An Integrated System Design and Safety Framework for Model-based Safety Assessment

by
Rahul Krishnan

A Dissertation
Submitted to the Faculty
of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
in the Systems Engineering Program
December 2021

Approved by:

Dr. Shamsnaz Virani Bhada (Advisor)
Worcester Polytechnic Institute

Dr. Thomas F. Gannon (Committee Member)
Worcester Polytechnic Institute

Dr. Jamie P. Monat (Committee Member)
Worcester Polytechnic Institute

Brian Bernier, AFSP (Committee Member)
Motional AD LLC

Date Approved: December 2021

Acknowledgement

First, I would like to extend my deepest gratitude to my advisor, Professor Shamsnaz Virani Bhada, for her guidance and support throughout the PhD program. She has helped me navigate personal and professional challenges throughout my time at WPI, and has been a mentor to me in the truest sense of the word.

I would also like to thank Professor Jamie Monat and Professor Tom Gannon for their valuable insight and encouragement throughout the PhD program as well as for taking the time to review my dissertation and providing their valuable feedback.

I'm extremely grateful to Brian Bernier for his guidance as one of my Ph.D. thesis committee members and for giving me the opportunity to intern on his team at Allegro Microsystems. The internship gave me a breadth of exposure on concepts in functional safety and also helped me leverage the incredible support and expertise of the functional safety team at Allegro Microsystems.

Many thanks to Jed Richards for taking the time to engage in several brainstorming sessions with me, which have been very useful in refining my research methodology.

I would also like to thank some of the friends I made in graduate school who are partially responsible for my success in completing this Ph.D. -Miguel, Norma, Vinayak, Niru, Bruno, Jayam, Kushboo, Rushab, Arjun, Becca, and Sophie. A special shout-out to my friend and roommate Lening Li, who in many ways has helped me successfully navigate my PhD program and a pandemic!

I would like to thank my closest friends, Rahul Chavan and Sheetal Tripathy, for their endless support and friendship. I would also like to say a big thank you to Ramya Singh- our shared passion and support for Liverpool FC has led to many cherished memories!

I would also like to thank my family in the US, Satish, Lara, Sanjay, Debbie, Matthew, and Julianna, thanks to whom my move to the US was seamless. I am eternally grateful for your support.

Finally, I'd like to thank my parents, Latha and Kongot Krishnan, and my brother Deepak Krishnan for their endless love, encouragement, and support throughout my life. My success in completing this Ph.D. program would not have been possible without them!

Abstract

The growing autonomy and complexity of modern-engineering systems has introduced novel challenges to assessing their safety. Traditionally, safety assessment is performed using a combination of safety analyses, safety verification and testing at various stages in the life cycle. While testing occurs in the later stages of the life cycle, safety analyses and safety verification can be performed early in the life cycle on the available design models and offer potential solutions to the need for early identification of safety-related design issues.

However, the manual nature of performing safety analysis and safety verification can introduce inconsistencies between the current system design model, the results from the safety analyses, and the results from safety verification. Additionally, due to the increased autonomy and complexity, small deviations caused by component degradation or unforeseen environment disturbances can lead to unanticipated system behavior. Current industry practice for safety assessment relies heavily on field-testing. However, field-testing is not a feasible solution to observe these rare events. Simulation testing offers a potential solution to observe such faulty system behavior caused by component degradation without relying on field-testing. While several approaches have adopted Model-based Development (MBD) for safety assessment, there is currently no framework or method that allows for feedback from safety analyses and safety verification to the system design model while also using simulation testing to observe faulty system behavior caused by degraded components or environmental disturbances.

This dissertation presents a model-based safety assessment framework, called the Integrated System Design and Safety (ISDS) framework, for assessing the safety of system design models early in the life cycle. The proposed framework combines a model-based safety analysis approach with a model-based safety verification approach to complete the safety assessment. The objective of the proposed framework is to eliminate sources of inconsistencies in the safety assessment process as well as to improve the safety of the system by using simulation testing to observe faulty system behaviors caused by component degradation. The ISDS framework is applied to a case study involving the development and safety assessment of a Forward Collision Warning (FCW) system in an autonomous vehicle. Results show that the feedback mechanism of the ISDS framework can eliminate the manual tasks that introduce inconsistencies in the safety assessment process as well identify a wider range of faulty system behavior compared to the current state-of-the-art. The key contributions of this research are, a) the feedback mechanism of the ISDS framework, which eliminated the need to manually update the system design model with the results from the safety analyses and safety verification, and b) the fault injection engine of the ISDS framework, which enables the use of simulation testing to identify faulty system behaviors caused by component degradation.

Contents

List of Tables **vii**

List of Figures **xi**

1 Introduction **1**
1.1 Thesis structure . . . 3

2 Literature review **4**
2.1 Model-based Systems Engineering (MBSE) methodologies . . . 4
2.1.1 Introduction to SysML . . . 5
2.2 Literature survey of model-based safety analysis approaches . . . 7
2.2.1 Current state-of-the-art in model-based safety analysis . . . 8
2.3 Literature survey of model-based safety verification approaches . . . 10
2.3.1 Current state-of-the-art in model-based safety verification . . . 11

3 Research Methodology **14**
3.1 Problem Statement . . . 14
3.2 Research Objective . . . 15
3.3 Research Approach . . . 15
3.3.1 Problem space examination . . . 15
3.3.2 Solution generation . . . 16
3.3.3 Solution evaluation . . . 16
3.4 Research Questions . . . 16
3.5 Case Study Description . . . 17

4 The Integrated System Design and Safety (ISDS) framework **19**
4.1 Overview of the ISDS framework . . . 19
4.1.1 CARLA Simulator . . . 21
4.1.2 Safety Profile for ISDS framework . . . 23
4.2 Model-based safety analysis approach . . . 25
4.2.1 Develop CONOPS and create failure operational scenarios . . . 27

4.2.2 Identify system requirements . . . 29
4.2.3 Perform system hazard and risk assessment (system HARA) . . . 29
4.2.4 Develop functional architecture . . . 32
4.2.5 Develop logical architecture . . . 32
4.2.6 Perform sub-system hazard and risk assessment (sub-system HARA) . . . 33
4.2.7 Generate safety artifacts . . . 35
4.2.8 Feedback from FMEA to SysML model . . . 38
4.2.9 Derive component safety requirements . . . 39
4.3 Model-based safety verification approach . . . 42
4.3.1 SysML model . . . 42
4.3.2 CARLA Simulator . . . 43
4.3.3 Fault Injection Engine . . . 43
4.3.3.1 Fault Injection Configurator (FIC) . . . 43
4.3.3.2 Fault Injection Campaign Manager (FICM) . . . 48
4.3.3.3 Safety Metric Evaluator (SME) . . . 50
4.3.4 Feedback from safety verification to SysML model . . . 50

5 Results **52**
5.1 Introduction to FCW systems . . . 53
5.2 ISDS framework applied to the FCW system case study . . . 55
5.2.1 Model-based safety analysis . . . 55
5.2.1.1 Develop CONOPS and create failure operational scenarios . . . 55
5.2.1.2 Identify system requirements . . . 55
5.2.1.3 Perform system hazard and risk assessment (system HARA) . . . 56
5.2.1.4 Develop functional architecture . . . 59
5.2.1.5 Develop logical architecture . . . 59
5.2.1.6 Perform sub-system hazard and risk assessment (sub-system HARA) . . . 61
5.2.1.7 Generate safety artifacts . . . 62
5.2.1.8 Feedback from FMEA to SysML . . . 66
5.2.1.9 Derive component safety requirements . . . 68
5.2.2 Model-based safety verification . . . 71
5.2.2.1 Results of fault injection simulation for FCW system . . . 72
5.2.2.1.1 Results for Scenario 1: Host vehicle encounters a stopped target vehicle on a straight road . . . 72
5.2.2.1.2 Results for Scenario 2: Host vehicle encounters a decelerating target vehicle on a straight road . . . 72
5.2.2.1.3 Results for Scenario 3: Host vehicle encounters a slower target vehicle on a straight road . . . 74
5.2.2.1.4 Results for evaluating mitigation strategy . . . 75
5.2.2.2 Feedback from safety verification to SysML model . . . 78
5.3 Discussion . . . 84
5.3.1 Research Questions . . . 84
5.3.1.1 Research Question 1 . . . 84

5.3.1.2 Research Question 2 85
5.4 Summary 86

6 Conclusion **87**
6.1 Research Contributions 89
6.2 Limitations 90
6.3 Future Work 91

A Risk assessment for hazard H2 and H3 **92**

B Failure modes of FCW system **94**

C FMEA for FCW system **98**

D Fault trees for safety goal SG2, SG3 **101**

E Updated system architecture of FCW system **104**

Bibliography **109**

List of Tables

1 Example of environment variables and their states for an automobile. 27
2 Classes of severity from [1, Tab. 1] 30
3 Classes of exposure from [1, Tab. 2] 30
4 Classes of controllability from [1, Tab. 3] 30
5 Determining the safety integrity level based on classification of severity, exposure, and controllability [1, Tab. 4] 31
6 Format and fields of the generated FMEA table 36
7 Templates for fault models currently supported by the ISDS framework 47
8 Risk assessment for system-level hazard H1: late engagement of FCW system for failure operational scenario 1. 57
9 Risk assessment for system-level hazard H1: late engagement of FCW system for failure operational scenario 2. 57
10 Risk assessment for system-level hazard H1: late engagement of FCW system for failure operational scenario 3. 58
11 Risk assessment for system-level hazard H2: Unexpected loss of FCW system for failure operational scenario 1. 92
12 Risk assessment for system-level hazard H2: Unexpected loss of FCW system for failure operational scenario 2. 92
13 Risk assessment for system-level hazard H2: Unexpected loss of FCW system for failure operational scenario 3. 92
14 Risk assessment for system-level hazard H3: Unexpected engagement of FCW system for failure operational scenario 1. 93
15 Risk assessment for system-level hazard H3: Unexpected engagement of FCW system for failure operational scenario 2. 93
16 Risk assessment for system-level hazard H3: Unexpected engagement of FCW system for failure operational scenario 3. 93

List of Figures

1 The Vee model representing the systems engineering life cycle, adapted from [2] 4
2 The Four Pillars of SysML, adapted from [3] 5
3 A simple example showing how a base SysML element, the "block" element, can be extended to create a new "vehicle" stereotype with a new set of tagged values or properties. The two elements on the right highlight how the vehicle stereotype is used to model vehicles in SysML.
Note: The properties and values selected for each vehicle are for illustration purposes only and are not intended to reflect real-world values. 6
4 Research approach used in this dissertation adapted from [4] 15
5 The Integrated System Design and Safety (ISDS) framework 20
6 Different layouts and weather conditions that can be simulated in CARLA 22
7 Safety profile of the ISDS framework 23
8 Sequence of steps for the model-based safety analysis approach 25
9 Sequence diagram representing the sequence of activities during system definition. The diagram highlights the responsible entity of an activity as well as the destination of the results from an activity. The iterative and collaborative nature that emerges from integrating the system and safety life cycle is clearly seen in this diagram. 26
10 Mapping used to define tagged values of the «scenarioConfiguration» stereotype to configure the failure operational scenario in CARLA 27
11 2-D map layouts and the corresponding 3-D rendering in CARLA illustrating the different roadway types and traffic zones that can be simulated in CARLA. The yellow dot on the 2-D maps identifies the current location of the vehicle on the map 28
12 System requirements modeled in SysML using a requirements diagram. 29
13 Safety goals modeled in SysML using a requirements diagram. 31
14 Functional architecture modeled in SysML using an activity diagram. 32
15 Logical architecture modeled in SysML using a block definition diagram. 32
16 Logical architecture modeled in SysML using a block definition diagram. 33
17 Failure modes modeled in SysML using an activity diagram and the «failureMode» stereotype. . . 34
18 Mapping of SysML diagrams and model elements to FMEA fields 35

19 Generic structure of the generated fault tree. 37
20 Mapping of SysML diagrams and model elements to fault trees 38
21 Component safety requirements modeled in SysML using a requirements diagram. 41
22 Traceability of safety integrity level from safety goals to component safety requirements 41
23 Overview of the safety verification approach used in the ISDS framework. 42
24 Overview of the diagram types used as well as the system design and safety data stored in the SysML model. 43
25 The SysML model elements and SysML diagrams from which each function extracts the system design and safety data 46
26 Sequence diagram elaborating the steps involved in performing the safety verification using the fault injection simulation. 49
27 Test scenarios used by NHTSA to evaluate the safety of FCW systems. 54
28 Failure operational scenario defined using the «scenarioConfiguration» stereotype for the FCW system case study. Tagged values for each failure operational scenario are defined such that it configures the CARLA simulator to recreate the test scenarios defined in the NHTSA FCW Confirmation Test. 56
29 System requirements for the FCW system case study. As eliciting comprehensive system requirements is not within the scope of this dissertation, the requirements identified in this figure represent a sufficient subset for this case study. 56
30 Safety goals of the FCW system defined using the «safetyGoal» stereotype in a requirements diagram. The safety goals are developed using the HAZOP technique and is the last step of system HARA. 58
31 Functional architecture for the FCW system modelled using an activity diagram. 59
32 Logical architecture for the FCW system modelled using a Block Definition Diagram (BDD). . . . 60
33 Logical architecture for the FCW system modelled using an Internal Block Diagram (IBD). 60
34 Failure modes for the *Determine host vehicle location* function of the Vehicle Dynamics Sensors block modelled defined using the «failureMode» stereotype in an activity diagram. Failure modes of the Target Vehicle Tracking module's *Determine target vehicle location* function are shown in blue. Failure mode combinations are identified using the "Combines with" dependency. 61
35 FMEA table for the *Determine host vehicle location* function exported as an image. Data in any cell can be updated by a safety engineer, if necessary. The "-" symbol signifies that the field was undefined in the SysML model and requires an engineer to update the spreadsheet. In this instance, the mitigation strategy is undefined in the SysML model. 62
37 FMEA table containing updates made by the engineer to the mitigation strategies for each failure mode of the *Determine host vehicle location* function as well as the modification to the safety goal violation of the "Host vehicle location lost intermittently" failure mode. Cells highlighted in red represent changes made to the FMEA compared to Figure 35. The table is exported as an image for better readability. 66
38 Updated SysML model showing changes made to the failure modes of the *Determine host vehicle location* function. The mitigation strategy for each failure mode has been updated with the data from the FMEA. The User Defined Safety Goal Violations for the "Host vehicle location lost intermittently" has been modified to not include SG1. 67

39 Component safety requirement allocated to the "Steering signal too high" and the "Steering signal too low" failure modes of the *Determine vehicle actuation* function which is allocated to the Vehicle Controller block 68
41 Results from safety verification for Scenario 1: Host vehicle encounters a stopped target vehicle on a straight road, showing the 12 out of 43 failure mode injections that resulted in a violation of one or more safety goals. The results show the nine behaviors of the FCW system where the faulty TTC, represented by the orange bars, is less than 2.1 seconds and two behaviors of the FCW system where it would pass NHTSA safety assessment but violate the system's own safety goals (i.e., orange bars where the TTC is greater than the golden run TTC value by at least 0.05 seconds). 73
42 Results from safety verification for Scenario 2: Host vehicle encounters a decelerating target vehicle on a straight road. The results show the 13 behaviors of the FCW system where the faulty TTC, represented by the orange bars, is less than 2.4 seconds and the three behaviors of the FCW system where it would pass NHTSA safety assessment but violate the system's own safety goals (i.e., orange bars where the TTC is greater than the golden run TTC value, i.e., the blue line, by at least 0.05 seconds). 74
43 Results from safety verification for Scenario 3: Host vehicle encounters a slower target vehicle on a straight road. The results show the ten behaviors of the FCW system where the faulty TTC, represented by the orange bars, is less than 2 seconds and the five behaviors of the FCW system where it would pass NHTSA safety assessment but violate the system's own safety goals (i.e., orange bars where the TTC is greater than the golden run TTC value, i.e., the blue line, by at least 0.05 seconds). 75
44 Results from safety verification of the redundant location sensor mitigation strategy for Scenario 1: Host vehicle encounters a stopped target vehicle on a straight road. The results show that the FCW system performs better with the mitigation strategy, shown in the gray columns, compared to the FCW system without the redundant sensor, shown in the red columns. With a redundant sensor, the FCW system engages at a TTC comparable to the TTC during the golden run (shown via the blue line) without violating a safety goal. In contrast, without the redundant sensor, the FCW system engages at a TTC with greater variations and four out the six failure modes violate safety goals. . 76
45 Results from safety verification of the redundant location sensor mitigation strategy for Scenario 2: Host vehicle encounters a decelerating target vehicle on a straight road. The results show that the FCW system performs better with the mitigation strategy, shown in the gray columns, compared to the FCW system without the redundant sensor, shown in the red columns. With a redundant sensor, the FCW system engages at a TTC comparable to the TTC during the golden run (shown via the blue line). In contrast, without the redundant sensor, the FCW system engages at a TTC with greater variations and four out of six failure modes violate safety goals. 77
46 Results from safety verification of the redundant location sensor mitigation strategy for Scenario 3: Host vehicle encounters a slower target vehicle on a straight road. The results show that the FCW system performs better with the mitigation strategy, shown in the gray columns, compared to the FCW system without the redundant sensor, shown in the red columns. With a redundant sensor, the FCW system engages at a TTC comparable to the TTC during the golden run (shown via the blue line). In contrast, without the redundant sensor, the FCW system engages at a TTC with greater variations and five out of six failure modes violate safety goals. 78

47 Failure modes of the *Determine host vehicle location* function updated with results from the safety verification. The "Single FM Simulation Results" and Combination FM Simulation Results" tagged values are updated with the safety goal violations and corresponding failure operational scenario during which the violation occurred. . . . 79
48 The *Yaw rate must not exceed yaw authority limits component safety requirement* updated with the results from the safety verification. The "Violation Simulation data" tagged value is updated with the failure operational scenario during which the requirement was violated as well as the failure mode that caused the violation. . . . 81
49 FMEA for the "Vehicle Dynamics Sensor" block that has been updated with the results from safety verification. The Simulation Data column is updated with the safety goal that was violated when the failure corresponding to that row was injected as well as the scenario during which the violation occurred. . . . 81

51 Failure modes for the *Determine host vehicle location* function . . . 94
52 Failure modes for the *Assess threat* function . . . 95
53 Failure modes for the *Determine host vehicle velocity* function . . . 95
54 Failure modes for the *Determine target vehicle location* function . . . 96
55 Failure modes for the *Determine target vehicle velocity* function . . . 96
56 Failure modes for the *Track target and host vehicle* function . . . 97
57 Failure modes for the *Determine host vehicle actuation* function . . . 97

58 FMEA for the *Determine host vehicle location* function . . . 98
59 FMEA for the *Determine host vehicle velocity* function . . . 98
60 FMEA for the *Determine target vehicle location* function . . . 99
61 FMEA for the *Determine target vehicle velocity* function . . . 99
62 FMEA for the *Assess Threat* and*Track target and host vehicle* function . . . 100
63 FMEA for the *Determine vehicle actuation* function . . . 100

64 Fault tree for violation of safety goal SG3: Prevent unexpected engagement of FCW system . . . 101

66 Updates made to the mitigation strategy and safety goal violation of the "Determine host vehicle location" function's failure modes . . . 104
67 Updates made to the functional architecture of the FCW system to include the *Determine host vehicle location* function . . . 105
68 Updates made to the logical architecture (BDD) of the FCW system to include the redundant location sensor . . . 106
69 Updates made to the logical architecture (IBD) of the FCW system to include the redundant location sensor . . . 107
70 Failure modes of the *Determine redundant host vehicle location* function . . . 108

Chapter 1

Introduction

In recent years, technological advances have led to the design of complex and sophisticated systems across multiple industries, such as autonomous vehicles, robotics, medical devices, and finance [5]. The growing complexity of these systems is evident in their design; modern-day systems have increasingly interdependent hardware and software architectures [6, 7]. This growth is driven by an increase in scale (number of elements in the system), diversity (number of distinct elements that make up the system), and connectivity (inter-relationships between elements in the system) [8]. While increased complexity has improved the performance and robustness of systems [9], it has also introduced new challenges [5]. One of the core challenges lies in assessing the safety of such complex systems [5, 10, 11].

The safety assessment of a system is a judgement made about the safety conformance and safety integrity achieved by every safety instrumented function within the system [12]. This judgment is based on a) the safety of the system development process (i.e., is the design process mature and in conformance with the criteria stated in the safety standards?) and b) safety of the system design (i.e., does the system design achieve the required safety integrity level?). While both factors are important for completing a safety assessment, this research will only focus on the latter (i.e., safety assessment based on the safety of the system design).

Academic researchers, industry experts and safety standards recommend assessing the safety of the system design using a combination of safety analyses, safety verification, and testing at various stages in the life cycle [12, 13, 14, 15, 16]. However, identifying safety-related design issues early in the life cycle can reduce rework, cost and schedule delays [17, 18, 19]. While testing occurs in the later stages of the life cycle, safety analyses and safety verification can be performed early in the life cycle on the available design models and offer potential solutions to the need for early identification of safety-related design issues.

Safety analysis is performed using a combination of one or more techniques, such as Failure Modes and Effects Analysis (FMEA), Fault Tree Analysis (FTA), and hazard analysis [13, 14, 15, 16]. These analyses are often performed using independent tools [20]. Each tool works off its own system design model; engineers must manually extract the relevant information from the original model and either import, or worse, re-create the system design model in each tool [20, 21]. Not only is this activity time-consuming and error-prone, but the existence of multiple system design models also creates a lack of traceability between the models and safety analyses [22]. Another limitation is that a failure to update the system design model in each tool may create an

inconsistency between the current design model and the safety analyses, leading to an incorrect safety assessment [20, 23, 24, 25, 26]. Similar limitations also plague activities performed during safety verification. In formal methods, which is one of the most prevalent techniques to perform safety verification early in the life cycle [27], specific model checkers are used to perform the safety verification. The model checking tool can use one of many available languages to describe the system design. Performing a safety verification requires a re-description or transformation of the original system design to the specific language used by the model checker [28]. This transformation may create an inconsistency between the current system design and the safety verification, and consequently lead to an incorrect safety assessment. The manual activities performed during safety assessment, as highlighted above, and the inconsistency they introduce is the first problem that this thesis seeks to address.

Another problem that arises during the safety assessment of complex systems lies in the increased unpredictability of the system's behavior, which can affect their safe and reliable operation [29]. Small deviations in the system, caused by component degradation or unforeseen environment disturbances, can lead to unanticipated system behavior [30]. This should not be confused with the concept of emergence [31] in systems thinking [32], where the system may not experience a fault or failure but complex interactions between the system's components result in unanticipated behavior at the system level. Assessing such emergent behavior is out of scope for this thesis. Instead, the focus is on unanticipated system behavior specifically due to component degradation and the interaction of the system with its environment after component degradation. Such phenomenon is observable in complex systems such as highly automated or autonomous vehicles and robots [7]. With the advent of increased autonomy in such systems, the importance of field-testing or on-road testing has been emphasized [33, 34, 35]. For example, the number of on-road miles accumulated is often presented as evidence for safety of highly automated vehicles [36]. Similarly, the National Highway Traffic Safety Administration (NHTSA) uses field-testing to determine the safety of complex systems like Forward Collision Warning (FCW) and Lane Departure Warning (LDW) in highly automated vehicles [37]. However, a safety assessment approach that relies heavily on field-testing is not feasible [36, 38, 39, 40]. Testing usually occurs later in the lifecycle, which hampers the early identification of safety-related design issues. Additionally, field-testing is an inefficient way to observe rare events like faulty system behavior due to component degradation or environmental disturbances; such situations may not arise during normal testing conditions [36]. Forcing component degradation during field-testing to observe how the system responds can be expensive and is not feasible either. As a result, faulty system behaviors that impact the safety of the system might go unrecognized. A system might be considered safe, but in reality would be unsafe if those behaviors are observed. Additionally, mitigation strategies in the system design might be incorrect or insufficient to handle those faulty behaviors. With the increased availability of high-fidelity simulators and increased computing power, simulation testing offers a potential solution to this problem [38, 36, 39]. Addressing the lack of simulation testing in current safety assessment approaches and the use of simulation to observe faulty system behavior caused by component degradation is the second problem that this thesis seeks to address.

To address the limitations and problems highlighted above, researchers have adopted a Model-based Systems Engineering (MBSE) approach to safety assessment [20, 41, 42]. The International Council on Systems Engineering (INCOSE) defines MBSE as "the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases". In MBSE, the model represents the single source of truth [43]. 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 [44]. Studies have shown that adapting an MBSE approach to system development, commonly referred to as Model-Based Development (MBD), improves the completeness and consistency in system development [45], fosters improved communication across design teams [45], provides added traceability between different models of the system [46], and enables easy integration with other engineering analysis tools [44]. The increased adoption of MBD also enhances the ability to perform safety analyses and safety verification early in the life cycle [43]. Consequently, an MBD approach to safety assessment offers a solution for resolving the inconsistencies that arise in the safety assessment process. MBD also enables verification activities like formal methods and simulation testing using early system design models, which helps to characterize and observe faulty system behavior in the context of safety assessments. This is because MBD allows more formal specification of a system's intended behavior with respect to its requirements and also enables the integration of a variety of analyses or simulations to be performed on the available system design model, such as formal methods or fault injection [27, 47, 48, 49, 11]. However, while MBD has been adopted for safety assessment, there is currently no framework or method that allows for feedback from safety analyses and safety verification to the system design model while also characterizing faulty system behavior to observe how the system deals with component degradation or environmental disturbances earlier in the life cycle. The need for feedback is important as it can facilitate consistency between the system design model, safety analyses, and safety verification.

This dissertation presents a model-based safety assessment framework, called the Integrated System Design and Safety (ISDS) framework, for assessing the safety of system architecture models (system design and safety) early in the life cycle. To narrow the scope of the thesis, the type of systems that the framework can assess the safety of is limited to automated or autonomous vehicles (i.e., automobiles). Such systems exhibit the increased autonomy and complexity that introduces the two challenges that this thesis aims to address. The proposed framework combines a model-based safety analysis approach with a model-based verification approach to complete the safety assessment. The objective of the proposed framework is to eliminate sources of inconsistencies in the safety assessment process as well as to improve the safety of the system by identifying faulty system behaviors that arise due to component degradation. The objectives are translated into two research questions that guide this dissertation. To evaluate the effectiveness of the ISDS framework as a safety assessment framework, it is applied to a case study involving the development and safety assessment of a Forward Collision Warning (FCW) system in an autonomous vehicle.

1.1 Thesis structure

The remainder of this dissertation is organized as follows. Chapter 2 reviews the relevant literature related to model-based safety assessment by discussing the current state-of-the-art in model-based safety analysis and model-based safety verification. Chapter 3 describes the research methodology and research questions guiding this dissertation. Chapter 4 describes the ISDS framework. Chapter 5 presents the application of the ISDS framework to the FCW system case study. Finally, Chapter 6 outlines the main conclusions and identifies recommendations for future research.

Chapter 2

Literature review

As the reader is now familiar with the two core problems in safety assessment this thesis seeks to address, and the potential of MBD as a promising solution to those problems, this chapter will review the current literature on approaches that use MBD for safety assessment. First, section 2.1 will review leading MBSE methodologies or frameworks for systems development. Next, Section 2.2 will review the current literature on model-based safety analysis approaches. Finally, section 2.3 will review the current literature on model-based safety verification approaches.

2.1 Model-based Systems Engineering (MBSE) methodologies

The systems engineering life cycle is often graphically represented using the Vee-model, as shown in Figure 1. Over the years, several MBSE methodologies have been developed to support the systems engineering life cycle. A comprehensive survey of prominent MBSE methodologies was conducted in 2008 and compiled in

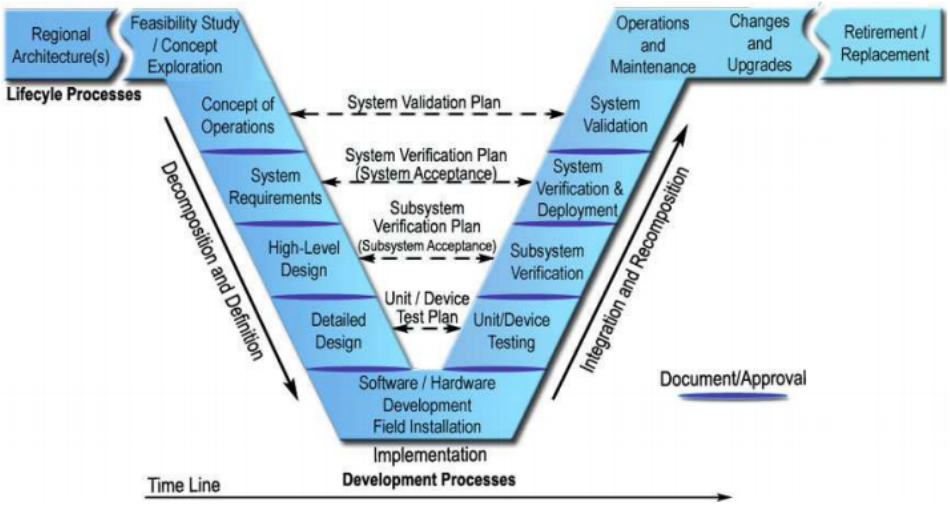


Figure 1: The Vee model representing the systems engineering life cycle, adapted from [2]

[50]. A list of these methodologies and any subsequent additions can be found in the repository maintained by OMG and INCOSE [51]. In [52], the author reviews the strengths and weaknesses of prominent methodologies like INCOSE Object-Oriented Systems Engineering Method (OOSEM) [53], CESAMES Systems Architecting Method (CESAM) [54], and IBM Harmony methodology for Systems Engineering [55]. In [56], the author reviews four additional methodologies like Weilkiens Systems Modelling Process (SYSMOD) [57], NASA JPL State Analysis [58], Vitech MBSE Methodology [59, 60], and MagicGrid (also known as MBSE Grid) [52]. In the list of MBSE methodologies, only MagicGrid explicitly augments the system design life cycle with safety activities. It includes a model-based safety analysis approach but does not currently support any verification activities.

Over the years, several modeling languages have been developed to implement MBSE; the Systems Engineering Book of Knowledge (SEBoK) provides a detailed list of modeling languages that have been developed for a variety of application such as the Systems Modelling Language (SysML) for descriptive models of systems, Modelica for analytical models of systems, Architecture Analysis and Design Language (AADL) for software design models, and Business Process Modelling Notation (BPMN) for business process models [61]. However, SysML is now the preferred modeling language for describing the system architecture [44] and has become the de-facto modeling language in systems engineering and MBD [62, 63, 64]. A majority of the prominent MBSE methodologies highlighted earlier in this section are based on SysML. Consequently, this thesis will review literature where SysML is used to model the system architecture. To familiarize the reader with the semantics and capabilities of SysML, the next section will provide a brief introduction to SysML.

2.1.1 Introduction to SysML

SysML is a general-purpose graphical modeling language which reuses and extends the unified modeling language (UML) for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities [65]. Being a standardized language with flexible semantics, it enables the creation of models customized for a specific application [43]. SysML allows for the

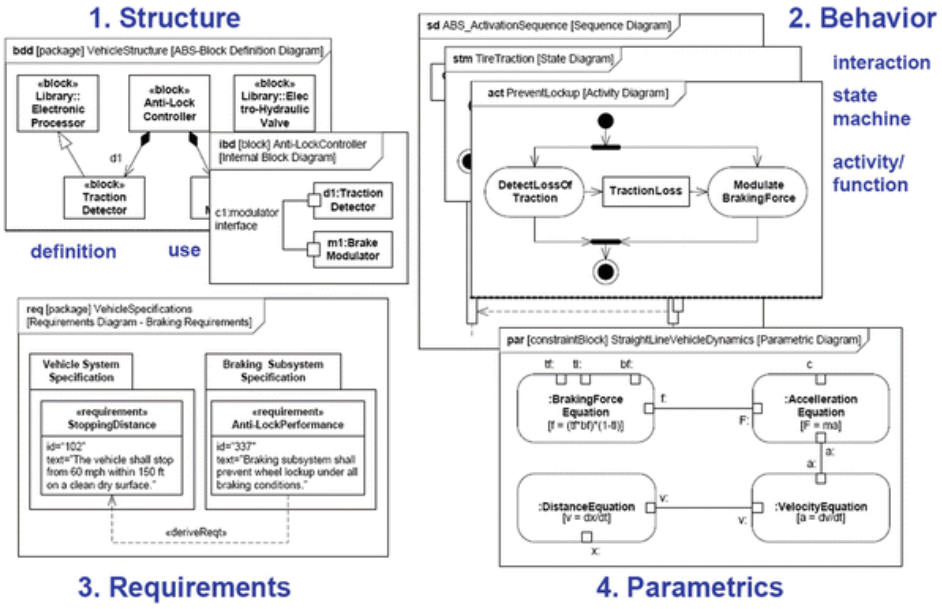


Figure 2: The Four Pillars of SysML, adapted from [3]

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.

SysML diagrams can be used to specify system requirements, behaviour, structure and parametric relationships. These are known as the four pillars of SysML [3], as shown in Figure 2. The system structure is represented by the Block Definition Diagram (BDD), Internal Block Diagram (IBD), and the package diagram. The BDD describes the system hierarchy, classification of the sub-systems or components, and the relationships between them. The IBD describes the internal structure of the system and emphasizes on the internal relationships between the sub-system or components, rather than the structural breakdown itself. The package diagram organizes the model i.e., it identifies and relates packages in the model. A package can hold a collection of one or more SysML diagrams, or other packages. Together, structural modelling defines the 'what' of a system, i.e., what it does, what it looks like, and what the relationships are [66].

The system behavior is represented by the use case diagram, activity diagram, sequence diagram and state machine diagram. The use case diagram provides a high-level description of system functionality to model requirements and contexts of a system. Activity diagrams model behavior within an operation and are used to model flow of data and control within activities or functions. Sequence diagrams model interactions between parts of a system by showing the messages/data passed between them with an emphasis on logical time or sequence of messages. Finally, state machine diagrams describe the transition in states of the system or its parts in response to events or actions. Together, behavioral modelling defines the 'how' of a system, i.e., describes the different interactions that take place within the system at different abstraction levels [66].

The system requirements are represented by the requirements diagram, which captures the requirements hierarchies and their relationships. According to [65], the requirements diagram is a cross-cutting construct between structure and behavior as elements of a requirements diagram can also appear in other diagrams. Finally,

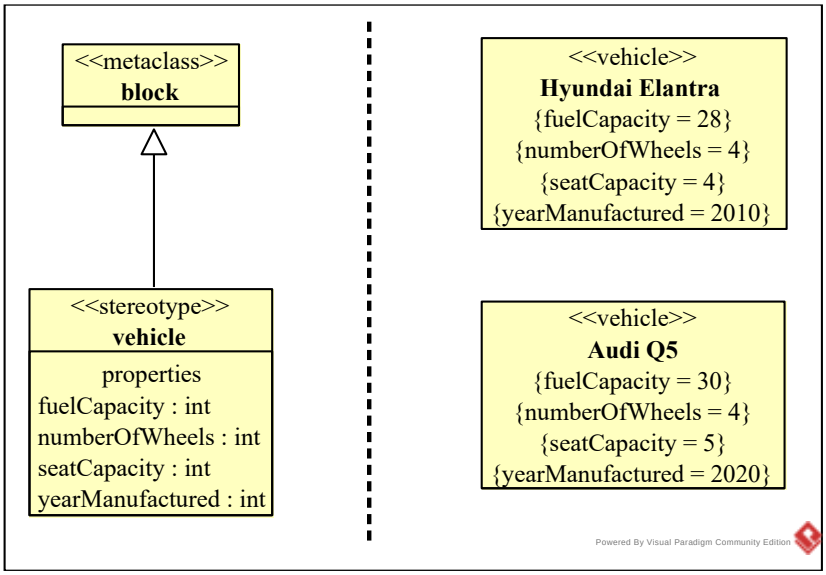


Figure 3: A simple example showing how a base SysML element, the "block" element, can be extended to create a new "vehicle" stereotype with a new set of tagged values or properties. The two elements on the right highlight how the vehicle stereotype is used to model vehicles in SysML.
Note: The properties and values selected for each vehicle are for illustration purposes only and are not intended to reflect real-world values.

parametric diagrams provides constraints on system parameters. The constraints represent rules that bound the properties of a system or define rules that a system conforms to [66].

The general-purpose nature of SysML allows it to be tailored for different domains, systems, and life cycle activities. The SysML Profiles mechanism allows existing SysML meta-classes (or elements) to be extended to adapt them for different purposes [65]. This is enabled by the use of stereotypes, which extend the current meta-class as a new model element that contains its own set of properties or tagged values. Figure 3 provides a simple example and illustrates how the base SysML block element can be extended to model vehicles in SysML.

Due to its prevalence as the preferred modeling language, this thesis only reviews literature that describes the system architecture using SysML. The remainder of this chapter is divided into two sections-the first section will review relevant literature related to model-based safety analysis, and the second section will review relevant literature related to model-based safety verification.

2.2 Literature survey of model-based safety analysis approaches

Bahr defines safety analysis as "the study of a system, identification of dangerous aspects of the system, and correction of them" [14]. While this definition is simplistic, it captures the objective of safety analysis, which is to assess the system safety and ensure that the designed system achieves the desired safety level. Safety standards like IEC 61508 [67] and ISO 26262 [12] recommend using traditional safety analysis techniques such as Preliminary Hazard Analysis (PHA), Functional Hazard Analysis (FHA), Hazard and Operability (HAZOP) studies, Failure Modes and Effects Analysis (FMEA), and Fault Tree Analysis (FTA). A comprehensive survey of safety analysis techniques can be found in [13, 14, 15, 16]. 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 and subsystem functions. Both PHA and FHA are qualitative, inductive approaches that are performed in the early design stages. Another qualitative hazard analysis technique is HAZOP, which uses guide words 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. Finally, FTA is a deductive, top-down approach used to determine the root cause of a certain undesired event. The analysis generates a fault tree, which 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 become a popular safety analysis technique 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 [29]. 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.) [29, 68, 69]. However, the traditional methods are still practiced in industry to analyze safety-critical systems. Sulaman et al. [70] compared STPA and FMEA safety analysis techniques on a collision avoidance system and found that both methods were able to identify all the different types of hazards, but each was better suited to identify specific types of hazards. Consequently, this research only focuses on traditional safety analysis techniques. As this research is primarily based on HAZOP, FMEA, and FTA for safety analysis, these techniques will be described in greater detail.

1. **Hazard and Operability Analysis (HAZOP):** HAZOP is an organized, structured, and methodical process for identifying hazards for a system throughout its life cycle [13]. The analysis uses guide words and system design representations to identify hazards. The approach is to brainstorm potential system deviations from the operational intent by combining guide words such as high, low, loss, delay etc. with system parameters such as speed, pressure, temperature etc. The resultant deviations capture potential hazards of the system. HAZOP analysis is based on the principle that experts from different backgrounds can brainstorm a more exhaustive list of system hazards.

2. **Failure Modes and Effects Analysis (FMEA):** FMEA is an inductive bottom-up approach used for evaluating the effect(s) of potential failure modes of subsystems, components, or functions [13]. The purpose of FMEA is to identify individual failures related to the design (component or function), evaluate the effects of those failures, and determine if design changes are necessary due to unacceptable reliability, safety or operation resulting from the failure mode. The FMEA is presented as a table with the element's failure mode, cause, effect, and mitigation strategy identified.

3. **Fault Tree Analysis (FTA):** FTA is a deductive top-down approach used for identifying the root causes and probability of occurrence of a specified top-level event [13]. The fault tree is a logical and graphical representation of the various combination of possible low-level events that can lead to the occurrence of an undesired top-level event. The combining low-level events are linked together using logical operators. The operators are primarily AND and OR relationships, but can also include NOT, priority AND, and exclusive OR. FTA can be for qualitative and quantitative analysis. A qualitative analysis can identify the minimal cut set, which represent the necessary and sufficient combination of events that can cause the undesired event. The minimal cut sets highlight the weakest link in the design. A quantitative analysis can identify the probability of occurrence of the top-level undesired event, given the individual probabilities of the low-level events.

2.2.1 Current state-of-the-art in model-based safety analysis

As the reader is now familiar with the various safety analysis techniques, this section will review the current progress in implementing these techniques using a model-based approach. Safety analyses are performed at various stages of the life cycle; different safety artifacts are created as the design progresses. Safety artifacts can be constructed from compositional (or structural) models as well as behavioral models of the system [71]. Over the years, several approaches have been developed that leverage a model-based approach 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 [72]. The approaches are classified into one of two groups: those that use SysML to model the system architecture and a different language to model the safety aspects of the architecture, and those that use SysML to model the system architecture as well as the safety aspects of the architecture.

Approaches that use different languages for the system architecture and the safety aspects of the architecture require an intermediate model transformation to create the safety model. The model transformation translates the SysML model to an equivalent safety model from which safety artifacts like fault trees and FMEA tables can be automatically generated. In [73] and [74], the authors demonstrate the ability to generate a single type of

safety artifact from SysML models. Xiang et al. [73] transformed SysML models into reliability configuration model (RCM) specifications from which static fault trees were generated. Hecht et al. [74] transformed SysML diagrams that modeled the system structure and behavior into AltaRica models from which an FMEA table was generated. Although both [73] and [74] 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 [75] 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 [73] and [74], 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. [41] identified this need for an integrated approach that covers all phases of the safety life cycle by developing a safety assessment framework called Sophia [76]. 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 [77] (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 common drawback of the model-based safety analysis approaches that requires a model-to-model transformation ([73, 74, 75, 41, 76]), lies in the challenge of maintaining consistency between the safety artifact, system design model, and safety model, as well as the increased possibility of data loss during each transformation [20, 78]. Based on the current literature, model transformation of the safety artifact back to the safety model and the system design model (i.e., system architecture) has not yet been implemented. Additionally, if the transformation is completed manually, it can be time-consuming and error prone.

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 [78] 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 [79, 80, 81]. Thramboulidis and Scholz [79] 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. [80] 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 information, 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. [81] present a hazard analysis profile that captures safety-related data by performing hazard analysis at each stage of the system design process. These approaches [79, 80, 81] highlight how SysML profies can be used to store relevant data from a hazard analysis in the same SysML model as the system design.

Mhenni et al. [20] 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 [82] where component redundancy is also captured in the system architecture model and dynamic fault trees are generated.

The common limitation of the approaches discussed in the section is the lack of feedback from safety analysis (or safety artifact) to the system design model. Changes made in the safety artifact are not automatically reflected in the SysML model and leads to an inconsistency between the safety artifact and SysML model (containing system design and safety data). This gap contributes to the inconsistency problem in safety assessment, which is described in Chapter 1. These limitations or gaps in the literature highlight the need to eliminate the manual activities in safety analyses by automatically generating safety artifacts from the SysML model and to automatically incorporate feedback from the safety artifact to the SysML model to prevent inconsistency.

2.3 Literature survey of model-based safety verification approaches

INCOSE defines verification as "a set of actions used to check the correctness of any element, such as a system element, a system, a document, a service, a task, a requirement, etc" [83]. For assessing the safety of the system design, the purpose of safety verification is to determine if the system design meets the safety requirements [12]. Safety standards like the ISO 26262 recommend using techniques such as formal methods and fault injection for safety verification [12]. Formal methods use formalisms and techniques based on mathematics for the specification, design and analysis of systems [14]. Commonly used formal method techniques include Automated Theorem Proving (ATP), static analysis, and model checking [84]. ATP determines if a certain theorem or statement that defines the system is valid or not based on the axioms of a certain theory [85]. Static testing involves code inspections, control flow analysis, data flow analysis etc. [14]. In model checking, the

system is modeled as a state transition system, and the specification is expressed as temporal logic formulae that define constraints to the system [86]. Finally, fault injection is used to assess the dependability of systems by using controlled experiments to observe the system's behavior in the presence of faults by explicitly injecting faults into the system [87]. Since model checking and fault injection are the most extensively used techniques for safety verification, these techniques will be described in greater detail.

1. **Model Checking**: The model checking technique is defined as follows: given a model of a system for which the set of finite state transitions have been encoded (system model) and a set of safety properties that the system must satisfy (safety requirements), the model checking problem is to identify all states in the system model that satisfy the requirements [86]. The model checking process consists of three steps: a) building the model of the system by converting the system design into a formal model that is accepted by a model checking tool (i.e., describing the state transitions possible for the system), b) describing the system specification by defining the properties the system must satisfy, and c) the verification process where all the states of the model are evaluated to check if it satisfies the defined properties.

2. **Fault Injection**: The fault injection technique consists of four parts: a) a set of faults to be injected into the system, b) a set of activations that represents the design elements of the system where the faults are to be injected, c) a set of readouts where the system behavior can be observed, and d) a set of measures that evaluate the dependability properties. The fault injection technique is effective at analyzing faulty system behavior in a controlled environment by forcing faults and other exceptional conditions and observing the resulting system behavior [88]. Fault injection techniques are broadly classified into two approaches– hardware-based and software-based approaches. Hardware-based approaches are accomplished at the physical level and use external sources to introduce faults into the system's hardware [89], while software-based (also referred to as simulation-based) approaches mimic faults through code changes or by injecting the effects of faults into the code [90].

2.3.1 Current state-of-the-art in model-based safety verification

As the reader is now familiar with the various safety verification techniques, this section will review the current progress in implementing these techniques using a model-based approach. The increased adoption of MBD has supported the use of techniques like model checking and fault injection for safety verification of complex systems [27]. The popularity of model checking arises from its ability to provide a formal guarantee that a given specification is obeyed [14]. Additionally, when a specification is not satisfied, a counterexample is generated and the offending behavior can be witnessed. In [47, 49], the authors developed the system design model in Simulink and used the SCADE design verifier to automate safety verification via model checking. In [20, 28], the authors developed the system design model in SysML and used the NuSMV symbolic model checker to perform safety verification. Overall, model checking has been shown to allow for the verification of system safety properties based on the requirements by using automated verification procedures [48]. However, the limitation with model checking is that it heavily relies on the knowledge and experience of the engineer to define the safety properties and system states [88], which can be challenging for complex systems that operate in diverse environments. Another challenge (and on-going research problem) with model checking is the state explosion problem, where the increase in system complexity leads to a state space that is too large for the technique to be computationally viable [86].

To overcome some of the model checking challenges, researchers have used fault injection for safety verification of complex systems [36]. Not only does fault injection make it easy to observe faulty system behavior but it also reduces the reliance on the expertise of engineers. The technique allows for automation when injecting faults into the system design model; the automatic nature of verification makes the high level of usability attractive to non-experts in the verification process [48]. Among the different fault injection approaches, simulation-based approaches are seen as a promising solution for safety verification. The advantages of simulation-based approaches are that they pose no risk of damage to the system, require less resources to execute in terms of time and effort, and have higher observability (how well the effects of faults on the system can be measured) and controllability (how well the location of faults in space and time can be controlled) of system behavior in the presence of faults [91, 92]. This approach is particularly suited for an MBD environment; it can provide timely feedback on the safety verification of the system design model early in the life cycle [89]. Additionally, high-fidelity simulators allow engineers to detect gaps in requirements, identify new test cases or observe unforeseen system behavior as a result of the injected faults in realistic operational environments earlier in the life cycle [36, 38, 39]. For example, Jha et al. [93] developed a fault injection tool called an Autonomous Vehicle Fault Injector (AVFI) that uses the CARLA simulator [94] for resilience assessment of autonomous vehicle systems. The tool can inject hardware, data, timing, and machine learning-related faults (source level fault models) using source code instrumentation at run time and compute resilience metrics such as mission success rate, traffic violations per kilometer, accidents per kilometer and time to traffic violations. Jha et al. [95] also developed the Kayotee fault injection tool, which uses the DRIVE Sim simulator [96] for safety and reliability assessment. Along with source level faults, the tool can inject faults modeled as bit-flips in different functional units of the GPU processor to identify unforeseen behavior under faulty conditions, such as safety envelope breaches and lane-centering breaches for autonomous vehicles. Li et al. [97] developed the AV-Fuzzer fault injection tool, which uses the LGSVL [98] simulator to find safety violations in autonomous vehicles under evolving traffic conditions. The tool uses a genetic algorithm to identify new operational scenarios or test cases that could lead to violations of requirements that are modeled as safety constraints. While such tools have shown success at identifying safety-related issues caused by faulty system behavior, they are not sufficient to perform a safety assessment since they do not consider the system design model as an input to safety verification. To leverage the benefits of MBD for early safety assessments, the safety verification must be integrated with the design and safety life cycle (i.e., the safety verification process should be based off the available system design model as well as the safety analyses that were performed on that design model).

Juez et al. [99, 42] overcome this problem by combining a model-based design process with a simulation-based fault injection technique. In [99], the authors use Simulink models to represent the system design and the Sabotage fault injection framework to perform safety verification early in the life cycle. The Sabotage framework uses the Simulink model to configure the type, location, and time for the faults to be injected and the Dynacar simulator [100] to perform the fault injection simulation. The framework uses source code instrumentation to inject faults and calculates the effect of the faults by computing the Fault Tolerant Time Interval (FTTI) (i.e., the time interval between injection of the fault and observation of the fault's system level effect). The limitations of this approach arise from the Simulink models–they are good for mathematically representing the system design, but cannot be used to store other design and safety data, such as requirements, use cases, failure modes etc. Additionally, the framework does not use the results from the safety analyses to configure the fault injection. This omission can cause an inconsistency between the results of the safety analyses performed during system definition and the safety verification. The authors overcome these limitations in [42] by developing the system

design models in SysML. The general-purpose nature of SysML allows different design and safety data to be stored in the same SysML model. The approach uses the SysML model and the results from the failure modes and effects analysis (FMEA) to configure the fault injection in a robot and environment simulator called Gazebo [101]. Using source code instrumentation, different failure modes are injected to simulate the system design in the presence of faults and the resulting behavior is evaluated against a predefined component level safety requirement stored in the SysML model.

While the approach by Juez et al. [42] is the most promising of the literature reviewed in this section, it has two limitations-

1. Lack of feedback from safety verification to system design and safety model: Results from the safety verification are not stored in the SysML model. This lack of feedback can introduce inconsistencies in the later stages of the life cycle.

2. Only verify safety requirements at the component-level: Safety requirements defined at the component level are verified using fault injection but system-level safety requirement violations are not. As a result, the effects of the component degradation (caused by injecting a fault in the component) cannot be traced to the system-level behavior observed (which is defined using system-level safety requirements).

These limitations or gaps in literature highlight the need for a model-based safety verification approach that closely aligns with design and safety life cycle by incorporating feedback between the SysML model, the safety analyses, and the safety verification as well as the need to verify safety requirements at different levels of abstraction.

In summary, the literature shows that an MBSE approach to safety assessment does help in eliminating the manual activities that introduce inconsistencies. Artifacts for hazard analysis, FMEA, and FTA can be automatically generated from a single SysML model. Safety verification techniques like fault injection can be configured using the system design and safety data from the same SysML model as well. Simulation-based fault injection approaches can leverage high-fidelity simulators to observe faulty system behavior caused by faults attributed to component degradation.

However, the gaps and limitations highlighted in section 2.2 and section 2.3 highlight the need for model-based safety assessment approach that can integrate the data from the system design and safety life cycle into a single SysML model, incorporates feedback between the SysML model, the results from the safety analyses (i.e., the safety artifact), and the results from the safety verification (i.e., results from the fault-injection simulation), and perform fault injection to assess safety violations at different levels of the system design, i.e., verify system-level safety requirements as well as component-level safety requirements.

Chapter 3

Research Methodology

The purpose of this chapter is to introduce the research methodology used in this dissertation for developing a model-based safety assessment framework. This chapter will introduce the problem statement and research objective, describe the research approach, formulate the research questions, and provide an overview of the case study used in this research. The answers to the research questions formulated in this chapter will demonstrate the ability of the framework to overcome the limitations in the current state-of-the-art. The chapter is divided into five sections-the first section introduces the problem statement, the second section introduces the research objective, the third section provides a detailed description of the research methodology, the fourth section states the research questions that this dissertation seeks to answer, and the fifth section provides an overview of the case study.

3.1 Problem Statement

Safety assessment of modern complex systems like highly automated or autonomous vehicles presents several challenges. This thesis will focus on two key challenges. The first challenge involves the inconsistencies that are introduced in the safety assessment when safety engineers manually perform tasks related to safety analysis or safety verification. Current model-based methods use SysML to model the system design or system architecture data such as requirements, functional and logical architecture, hazards etc. Data from the SysML model is used to perform safety analyses like FMEA or FTA and safety verification like simulation-based fault injection. While current methods (semi-) automate the transformation from SysML model to safety analysis and SysML model to safety verification, the feedback of results from safety analysis and/or safety verification to the SysML model is still a manual process that is performed by safety engineers. This manual feedback may introduce inconsistencies between the system design model (i.e., SysML model), the safety analysis, and the safety verification. These inconsistencies may lead to an incorrect estimation of the system design's safety and consequently, a flawed safety assessment. The second challenge deals with the lack of simulation testing used to observe faulty system behavior that is caused by component degradation or unforeseen environmental disturbances. Current safety assessment methods for highly automated or autonomous vehicles emphasize field-testing. However, testing is not a feasible approach to characterize such faulty behavior, as there is no guarantee that such behavior will arise

during normal testing conditions. Consequently, there is a need to characterize such faulty behavior and observe how the system design deals with such exceptional conditions without relying on field-testing.

3.2 Research Objective

This research aims to develop a model-based safety assessment framework for complex systems like highly automated or autonomous vehicles that can reduce the inconsistencies introduced by manual tasks performed by engineers during safety assessment, and can use simulation testing to characterize faulty system behavior due to component degradation early in the life cycle using just system architecture models. The framework is evaluated by applying it to a case study, which involves the development and safety assessment of a Forward Collision Warning (FCW) system in an autonomous vehicle. The data gathered from the case study is used to answer the research questions formulated later in this chapter.

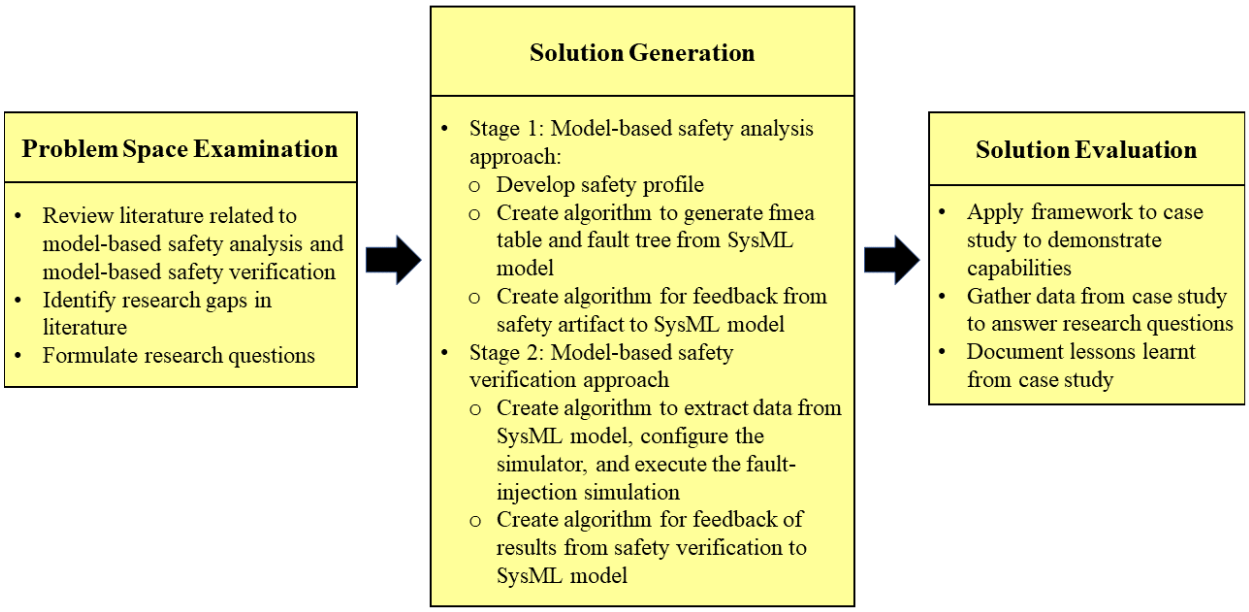


Figure 4: Research approach used in this dissertation adapted from [4]

3.3 Research Approach

The framework, called the Integrated System Design and Safety (ISDS) framework, is developed using the design research approach illustrated in Figure 4. A design research approach explores the problem space to define the research gaps, describes the design method developed to address the problems, evaluates the design method, and documents the outcome and lessons learned from the realized artifact [4]. The design research approach adopted in this dissertation consists of three steps, 1) Problem space examination, 2) Solution generation, and 3) Solution evaluation.

3.3.1 Problem space examination

The problem space for model-based safety assessment is explored by conducting a thorough literature review in the field of model-based safety analysis and model-based safety verification. For each field, the literature

survey concludes by highlighting the limitations in the current state-of-the-art. These limitations identify the research gaps in the literature and offer potential features that the ISDS framework should incorporate to extend the current state-of-the-art. The gaps are examined to identify metrics that can provide a quantitative measure for benefits of the framework. The research questions formulated in this dissertation use these metrics to provide clear, concise, and empirical evidence for the ability of the framework to overcome the limitations in the current state-of-the-art.

3.3.2 Solution generation

Next, the ISDS framework is developed as a solution to the identified research gaps. The framework is designed to integrate the system design and safety life cycle, to use a single SysML model to store system design and safety data, and to incorporate a model-based safety analysis and model-based safety verification approach. The development of the framework is divided into two stages. The first stage focuses on the approach used for performing model-based safety analysis. The key capabilities implemented for this stage includes a safety profile for the SysML model in order to the store system design and safety data in a single SysML model, the algorithm to generate FMEA tables and fault trees from the SysML model, and the feedback mechanism from the safety artifact to the SysML model (i.e., to update any changes made in the FMEA table in the SysML model). The second stage focuses on the approach used for model-based safety verification. The key capabilities implemented for this stage include the algorithm to extract information from the SysML model, configure the simulator, and perform a fault injection simulation, and the feedback mechanism from safety verification to the SysML model (i.e., to update the SysML model with the results from the safety verification).

3.3.3 Solution evaluation

The ISDS framework is applied to a case study to evaluate its effectiveness as a model-based safety assessment framework and its ability to overcome the limitations of the current state-of-the-art. The case study involves the development and safety assessment of a Forward Collision Warning (FCW) system in an autonomous vehicle. The case study is used to gather data for answering the research questions that will be formulated later in this chapter. Finally, the lessons learned from developing and applying the ISDS framework are documented.

3.4 Research Questions

The focus of this dissertation is to develop a model-based safety assessment framework that is effective at assessing the safety of a complex system and can overcome the limitations in the current state-of-the-art. As stated in Section 3.3.3, the framework is evaluated by applying it to a case study (i.e., development and safety assessment of an FCW system in an autonomous vehicle). As such, the research questions are closely tied to the case study. In particular, the two research questions that guided this dissertation are:

1. **In the FCW system case study, does the ISDS framework eliminate tasks that introduce inconsistencies compared to the current state-of-the-art?**

   Put another away, this question seeks to answer if the feedback mechanism in the ISDS framework reduces the number of tasks that a safety engineer must perform, especially tasks that introduce inconsistencies, compared to the mechanism used in the current state-of-the-art, which either uses a manual method for feedback or no feedback method at all.

2. **In the FCW system case study, does the ISDS framework assess the safety of the system for a wider range of behaviors than the current safety assessment method for FCW systems?**

   Currently, the safety of FCW systems is assessed without explicitly testing system behavior in faulty conditions (i.e., without forcing faults in the system and observing the resultant system behavior). This question investigates if the ISDS framework can use simulation testing to assess a system which is considered safe based on current assessment methods and identify behaviors of the system during which it is unsafe (i.e., by forcing faults in the system and observing the resultant system behavior).

3.5 Case Study Description

This section provides a brief overview of the case study used to evaluate the ISDS framework. The case study involves the development and safety assessment of an FCW system in an autonomous vehicle. An FCW system identifies the potential for an impending crash situation at the front of a vehicle and provides an alert or control signal to another vehicular sub-system to transition the vehicle to a safer state. The current industry practice to assess the safety of FCW systems is defined by the National Highway Traffic Safety Administration (NHTSA). The assessment involves observing the behavior of the vehicle equipped with the FCW system under conditions of a potential collision in different operational scenarios. In each scenario, the Time-To-Collision (TTC) metric is computed; TTC is defined as the time required for two vehicles to collide if they continue at their current speed and trajectory [102]. The system is considered safe if it issues an alert at a TTC greater than the pre-defined minimum TTC for the vehicle and test scenario.

In this case study, the ISDS framework is applied to the development and safety assessment of the FCW system. Once the system design and safety data is stored in the SysML model, the model-based safety analysis approach is used to generate the FMEA tables and fault trees. Any changes made to the safety artifacts are automatically updated in the SysML model. After the safety analysis is complete, the model-based safety verification approach is used to configure and execute the fault injection simulation. The TTC metric and operational scenario defined by the NHTSA is used in the fault injection simulation. First, the system behavior is observed without injecting any faults and the TTC is computed (i.e., the current industry practice for assessing the safety of FCW systems). Next, the system behavior in the presence of faults is observed; the failure modes identified in the FMEA are injected during the simulation and the TTC is computed. Once the verification is complete for the system, the results are updated in the SysML model. To answer the first research question, the study will look at the feedback mechanisms of the ISDS framework. The current state-of-the-art either uses a manual method for feedback (which could potentially introduce inconsistencies and errors) or no feedback at all (which definitely introduce inconsistencies). The study will highlight how the ISDS framework eliminates tasks that introduce inconsistencies. To answer the second research question, the study will look at the results of the model-based safety verification. The study will compare the number of TTC values computed using the current industry practice of assessing safety of FCW systems against the number of TTC values computed using the ISDS framework. The comparison will demonstrate how the framework can use simulation testing (based on fault injection) to assess safety for a wider range of behaviors than the current safety assessment method can. The results will highlight how the framework can assess a system which is considered safe based on current assessment methods and can identify behaviors of the system during which it is unsafe.

While the solution evaluation is only based off a single case study, the FCW system used in the case study represents the kind of complex system that will exhibit the problems that this dissertation seeks to address. The

research questions are also sufficiently narrow in scope to prevent any generalizations that is not grounded in data. It is assumed that a single case study is sufficient to demonstrate the capabilities of the ISDS framework and to collect enough data to answer the research questions formulated in this chapter. The next chapter provides an overview of the ISDS framework and a detailed description of the approach used for model-based safety analysis and model-based safety verification.

Chapter 4

The Integrated System Design and Safety (ISDS) framework

As system complexity increases, so does the challenge of assessing the safety of such systems. As discussed in chapter 2, there is a growing consensus in academia and industry that a holistic framework is required for safety assessment that does not rely heavily on field-testing. While several model-based safety assessment approaches have been developed to address this requirement, chapter 2 highlighted some of the limitations that still persist in the current state-of-the-art. To overcome the highlighted limitations, this dissertation aims to develop a model-based safety assessment framework. In this chapter, readers will be introduced to the ISDS framework and will be provided with a detailed description of the model-based safety analysis and model-based safety verification approach adopted in the framework. The chapter is organised into three sections-the first section provides an overview of the entire framework, the second section focuses on the model-based safety analysis approach, and the third section focuses on the model-based safety verification approach.

4.1 Overview of the ISDS framework

The ISDS framework uses a Model-based Design (MBD) approach based off the systems engineering Vee methodology [103] for safety assessment. The Vee methodology has been advocated by researchers as a useful template for developing a methodical approach to safety assessment - not only does it capture the iterative nature of model development, but it also traces the verification activity to a corresponding system design activity at different abstraction levels. [38]. Traditionally, the left side of the Vee deals with system definition, where the system evolves from basic requirements to detailed design models. During system definition, the system is incrementally broken down into smaller components until the system design is realized. The right side of the Vee deals with verification and validation, where the system is incrementally integrated and verified against the requirements defined at the corresponding system definition stage. Verification begins at the smallest component-level, continues up the right side of the Vee by integrating larger sections of the system, and ends with system-level verification activities.

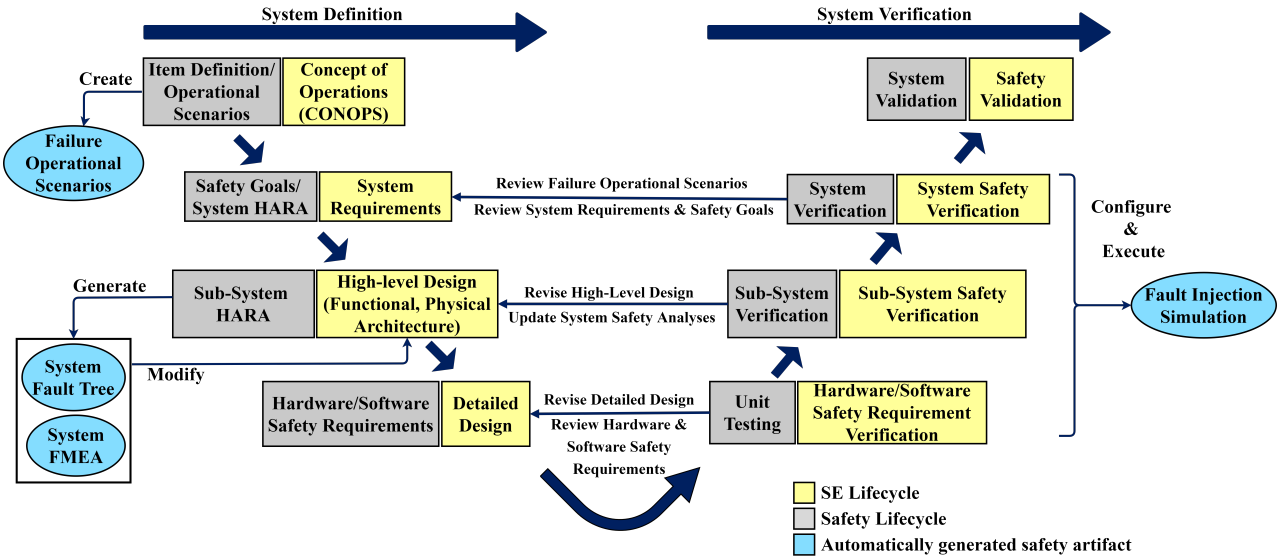


Figure 5: The Integrated System Design and Safety (ISDS) framework

The ISDS framework uses the Vee methodology within the MBD environment. The framework integrates the system design and safety life cycle via a one-to-one mapping of the different stages in both life cycles, as shown in Figure 5. SysML is used to model system design and safety data as the system progresses through its life cycle. After each stage in the ISDS framework, system design and/or safety data is captured in a SysML diagram. The framework leverages the ability of SysML to describe system structure and behavior using different diagrams to model system requirements, system structure, system behavior, and system safety. In Figure 5, the yellow items represent stages in the system design life cycle, the gray items represent stages in the safety life cycle, and the blue items represent the safety artifacts generated or configured by the framework. The framework captures the iterative nature of safety assessment by highlighting the feedback loops between a system definition activity and its corresponding system verification activity.

During system definition (i.e., the left side of the Vee), the focus of the ISDS framework is to incrementally develop the system architecture, store the system design and safety data in the SysML model, and perform the safety analyses. A SysML profile called the safety profile is developed to capture different types of data in the SysML model. The safety profile uses stereotypes and tagged values to modify or extend base SysML elements for the safety domain. A detailed description of the safety profile used in the ISDS framework can be found in section 4.1.2. By using a safety profile, the framework does not require separate models for system design and safety. Consequently, no model transformation is required, and the approach does not inherit the limitations associated with model-to-model transformation. As the framework progresses down the left side of the Vee, the analysis proceeds from system-level to component-level and the different activities related to the system design and safety life cycle are completed. To generate the safety artifacts (i.e., the FMEA tables and fault trees), a python script parses through the SysML model and extracts the required data. The safety artifacts are in a spreadsheet and graphical image file format, respectively. If required, a safety engineer can modify or update the data in the spreadsheet (FMEA table). The feedback mechanism of the ISDS framework updates the SysML model with any changes made to the artifact. A detailed description of the left side of the framework is provided in section 4.2.

During system verification (i.e., the right side of the Vee), the focus shifts towards verifying if the implemented system design satisfies the safety requirements defined during system definition. The framework uses a

simulation-based fault injection approach to perform this verification. The fault injection is performed in the CARLA simulator, which is an open-source simulator for automotive systems. The approach allows users to observe system behavior in the absence and presence of faults, and under a variety of operational scenarios. The system under development is recreated and configured in the CARLA simulator using data from the SysML model. First, the system is executed in the simulator to observe its behavior without any faults. Next, the system's failure modes are inserted during execution and the resulting faulty behavior is observed. The two executions (also called runs) are compared to determine if any safety requirements were violated. Finally, the feedback mechanism updates the results from the fault injection in the SysML model.

In contrast to the traditional Vee methodology, the framework does not incrementally integrate and verify as it progresses up the right side of the Vee. Instead, the SysML model configures the entire system in the simulator and verification of safety requirements is performed at the component-level as well as the system-level. While this approach does verify safety requirements in a similar vein to the traditional Vee methodology, it omits integration related issues. Since the safety assessment is performed using system architecture models and not the actual implementation of the system, identifying integration related safety issues is considered out of scope for this dissertation. Nevertheless, it is an important aspect of the life cycle and is a promising area for future research.

Now that the reader has an overview of the ISDS framework, it is important to understand two components of the framework that enable the model-based safety assessment i.e., the CARLA simulator (described in section 4.1.1) and the SysML safety profile (described in section 4.1.2). To perform safety verification, the ISDS framework adopts a simulation-based fault injection approach using the CARLA simulator. To configure and execute the fault injection in CARLA, the SysML model is enriched with simulation-specific properties. This is enabled by using the SysML safety profile, which converts general purpose SysML elements to domain specific elements using stereotypes and tagged values. To understand the safety profile and how the model-based safety analysis and model-based safety verification approach uses the safety profile, it is important for the reader to first be familiar with the CARLA simulator. The next section will introduce the simulator and provide a detailed description of its capabilities.

4.1.1 CARLA Simulator

The CARLA simulator is only used during system verification (i.e., right side of the framework). However, during system definition (i.e., left side of the framework), several SysML elements are defined to store simulation specific properties (i.e., to configure the CARLA simulator), which is enabled by the SysML safety profile. Hence, it is important for the reader to be familiar with the simulator before the rest of the framework. CARLA is an open-source simulator for autonomous automotive system. It provides support for a variety of sensor suites and driving environments, scenario generation, and the complete control of actors (pedestrian and vehicles) in the simulator. It is a high-fidelity simulator that can create detailed representations of different environments such as urban layouts, buildings, streets signs, highways, etc., for a range of weather and time of day conditions. Figure 6 (a), (b), and (c) illustrate some of the layouts available in CARLA and Figure 6 (d), (e), and (f) illustrate the different weather conditions that can be simulated in CARLA. The simulation platform allows user to setup a variety of sensors and gather data such as GPS coordinates, speed, acceleration, camera images, radar data, inertial measurement unit (IMU) data, etc., of an automotive system.

CARLA's architecture enables great flexibility and realism for graphical rendering and physics simulation. It is implemented as a layer over the Unreal Engine 4 (UE4) [104], which provides state-of-the-art rendering

(a) Layout for a highway environment (b) Layout for a rural environment (c) Layout for an urban environment

(d) Clear sunset weather conditions (e) Cloudy sunset weather conditions (f) Rainy sunset weather conditions

Figure 6: Different layouts and weather conditions that can be simulated in CARLA

quality and hyper realistic physics. CARLA can simulate a dynamic world and provides an interface between the world and an agent. The agent is usually an autonomous vehicle that is equipped with a set of sensors to observe its environment and a set of behaviors encoded to achieve a goal. To enable the interaction between the world and agent, the simulator uses a server-client system. The server (also called the world module), runs the simulation and renders the scene. The client module represents the agent that the user has developed (i.e., the system under development). The client communicates with the world to get information about its surroundings. The client sends commands to the world and gets sensor readings in return. The commands control the behavior of the automotive system such as throttle, steering and brake signals. After executing the command, the world returns the latest sensor readings. Additional commands can be used to control environmental properties like the weather, illumination, and friction of roadway surfaces. Finally, non-agent automotive systems and pedestrians can be added in the simulation as required.

The ISDS framework is closely coupled to the CARLA simulator. In its current form, the framework can only assess the safety of systems that can be represented in the simulator, i.e., automotive systems. A CARLA client is developed to represent the system under development using the framework- the client is a skeleton code of the system written in Python. The SysML model is a graphical representation of this client and the system design and safety data from the SysML model are used to configure the parameters in the client. The client is developed using the libraries of the CARLA simulator but contain parameters that are undefined and need to be configured by the SysML model. It should be noted that this research assumes that the SysML model is an accurate representation of the client in CARLA. Simulating the behavior of an automotive system requires complex python code, whereas the SysML model is developed at a much higher level of abstraction. However, it is the responsibility of the engineer to ensure that the functions and logical blocks defined in the SysML model are consistent with those used in the client. This does give rise to questions regarding model validation, i.e., *"how can one guarantee that the SysML model is consistent with the client?"*. Several researchers have developed methods for code generation from SysML system models [105, 106]. The SysML models are used to generate code that can be run on specific simulators. However, these approaches are limited to simple systems and require

support from the simulators. It is unknown if such approaches will scale for more complex systems, such as an automotive system within a high-fidelity simulating environment. Consequently, generating code for the entire CARLA client, i.e., code generation from SysML design models, is considered out of scope for this dissertation. However, the topic is closely tied to model validation and is a potential avenue for future research.

This section introduced the CARLA simulator and its capabilities. The CARLA client and world are configured based on the data and simulation-specific properties stored in the SysML model. The ability to store such domain specific data in a general purpose modeling language is enabled by using SysML profiles. The next section will describe the safety profile developed for ISDS framework and highlight how the profile helps to store system design data, safety data and simulation-specific properties in the SysML model.

4.1.2 Safety Profile for ISDS framework

The safety profile is a SysML profile used in the ISDS framework that allows data from different domains to be stored in the same SysML model, i.e., system design data, safety data, and simulation-specific properties. Native SysML elements are extended (i.e., customized) for storing specific types of data based on the domain and platform of the modeled system. Stereotypes and tagged values are used to extend the reference meta-class of a native SysML element. Figure 7 shows the safety profile of the ISDS framework. The safety profile extends the requirement, block, and use case meta-class in SysML to create new stereotypes that represent the data required for the ISDS framework. In Figure 7, the triangle connecting a parent block with the child blocks represents a generalization (or inheritance) relationship. It can also be read as "is a type of" relationship (i.e., the child block is a type of parent block).

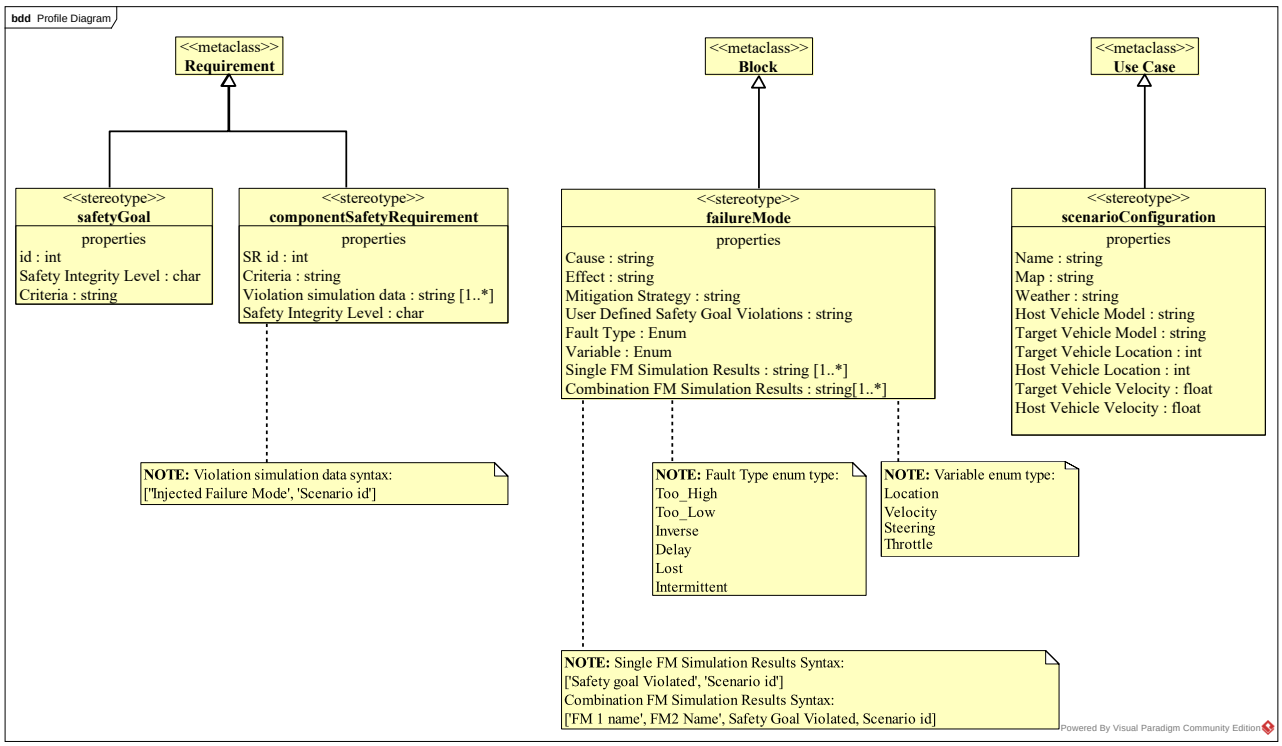


Figure 7: Safety profile of the ISDS framework

First, the requirement meta-class is extended to create two new types of requirements-safety goal and component safety requirement. The «safetyGoal» stereotype represents a system-level safety requirement and

has three properties-an id, a safety integrity level, and a criteria. The safety integrity level represents the risk associated with a safety goal and is evaluated using the risk assessment framework in ISO 26262 [1]. Based on this framework, a safety goal can have one of five values as its safety integrity level- QM, A, B, C, D (in increasing order of risk). The criteria represent a logical expression that constitutes the condition for violation of the safety goal. It is used to formalize the text-based requirement and configure the safety goal in the CARLA simulator for safety verification. The result of the criteria is a Boolean which returns true if the safety goal is violated.

The «componentSafetyRequirement» stereotype represents a low-level hardware or software safety requirement and has four properties-an id, a criteria, the violation simulation data, and a safety integrity level. Similar to the safety goal, the criteria is a logical expression that constitutes the condition for violation of the requirement and is used to formalize the text-based requirements and configure the requirements in the CARLA simulator. The violation simulation data acts as a placeholder to store the results from the fault injection simulation (i.e., after safety verification) in the SysML model. This tagged value stores the injected failure mode name that caused the requirement to be violated, and the failure operational scenario in which the violation occurred. The note attached to the stereotype highlights the syntax in which the results are stored. Finally, the component safety requirement inherits the safety integrity level of the safety goal it is derived from. The inheritance process will be described in detail in section 4.2.9, Figure 22.

Next, the block meta-class is extended to create the «failureMode» stereotype. The failure mode represents the manner in which a system element (in this case, a function) can fail. Each failure mode contains a cause, an effect, a mitigation strategy, user-defined safety goal violations (each are string data type), a fault type, a variable (each are enum data type) and a variable to store the results from the fault injection simulation, one for injection of single failure modes (i.e., one at a time), and one for the injection of multiple failure modes (i.e., more than one at a time). The cause, effects, and mitigation strategy are data that is commonly found in an FMEA, as highlighted in section 2. The user-defined safety goal violations, as the name suggests, contains the possible safety goals that the failure mode could violate according to the engineer performing the analysis. This is useful as it allows engineers to compare the user-defined safety goal violations against the results from the safety verification, which identify the safety goals violated using fault injection. In other words, it is a comparison of an analysis based on an engineer's intuition and experience against a more formal verification analysis. The variable and fault type are used to characterize the failure mode in the CARLA simulator. The variable identifies the sensor that the failure mode is linked to, i.e., location, velocity, actuator steering, actuator throttle or processing module. The fault type identifies the behavior of the sensor data - sensor data higher than expected (Too high), sensor data lower than expected (Too low), sensor data inversed or negated (Inverse), sensor data lost or delayed, or intermittent reception of sensor data. The variable and fault type together translate the failure mode defined in the SyML model to a failure mode that can be injected in the client developed to execute in CARLA. Section 4.3.3.1 explains failure mode characterization in greater detail. The single FM simulation results and combination FM simulation results are tagged values that act as placeholders to store the results from the fault injection simulation (i.e., safety verification). The single FM simulation results tagged value stores the safety goal violated when the failure mode was injected and the failure operational scenario in which the violation occurred. The combination FM simulation results tagged value stores the names of the failure modes that were injected together, the safety goal violated when the failure modes were injected, and the failure operational scenario in which the violation occurred. The note attached to the stereotype highlights the syntax in which the results are stored.

Finally, the use case meta-class is extended to create the «scenarioConfiguration» stereotype, which contains variables to configure the operational environment in CARLA. These variables are defined based on the operational scenario identified for the system and configure the environment variables in CARLA to recreate the operational scenario. The factors that influence the definition of these variables include the weather, the roadway type, traffic conditions, vehicle speeds, time of day etc. Section 4.2.1 describes how an operational scenario is translated to the tagged values of the «scenarioConfiguration» stereotype to recreate the operational scenario in CARLA.

In summary, this section described the SysML safety profile developed for the ISDS framework. It highlighted the extensions made to native SysML elements for storing system design data, safety data, and simulation specific properties in the same SysML model. Now that the reader is familiar with the CARLA simulator and its capabilities as well as the safety profile that enables the configuration of the simulator, the next section will describe the model-based safety analysis approach adopted in the system definition phase of the ISDS framework. It will highlight how the safety profile is used to to store system design data, safety data and simulation-specific properties in the SysML model, describe the algorithm for generating safety artifacts from the SysML model, and the algorithm for updating the SysML model with changes made to the safety artifact via the feedback mechanism.

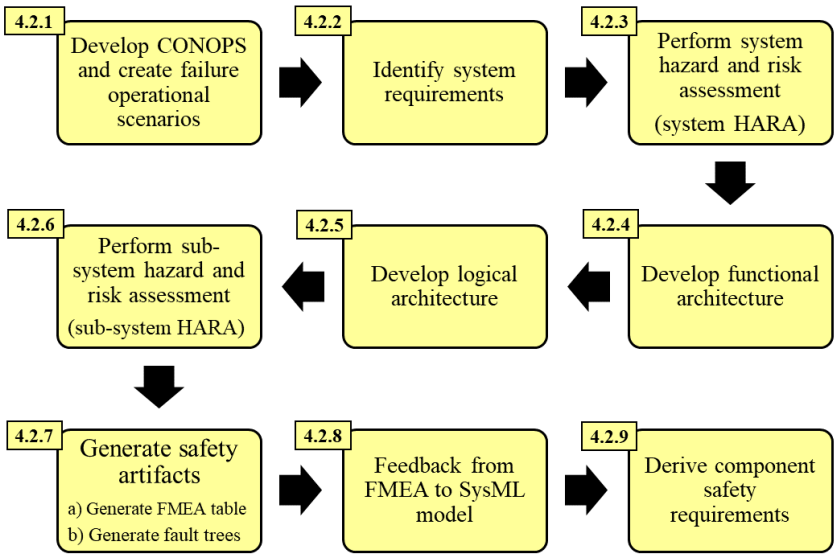


Figure 8: Sequence of steps for the model-based safety analysis approach

4.2 Model-based safety analysis approach

The model-based safety analysis approach is adopted in the left side of the ISDS framework. As previously highlighted, during system definition, the focus of the framework is to incrementally develop the system architecture, store the system design and safety data in the SysML model, and perform the safety analyses. The model-based safety analysis approach is comprised of nine steps, as shown in Figure 8. It is important to note that the purpose of Figure 8 is to show the steps that make up the model-based approach. To make it easy for the reader to visualize the individual steps of the approach, any feedback loops between steps have not been shown. A more detailed representation of the model-based safety analysis approach is provided in Figure 9.

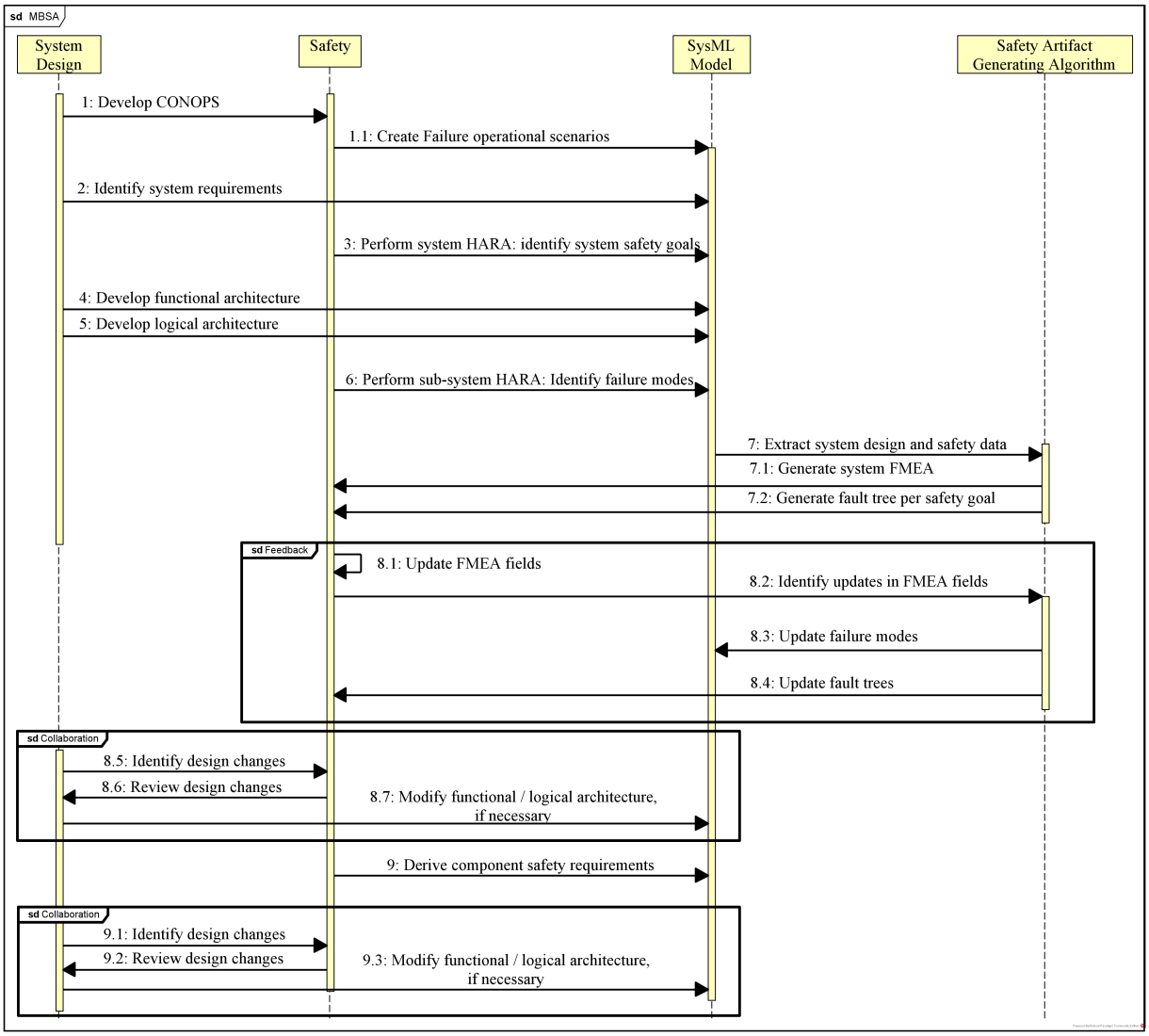


Figure 9: Sequence diagram representing the sequence of activities during system definition. The diagram highlights the responsible entity of an activity as well as the destination of the results from an activity. The iterative and collaborative nature that emerges from integrating the system and safety life cycle is clearly seen in this diagram.

The sequence diagram is able to provide a more nuanced description of the approach compared to Figure 5 and Figure 8 by showing the flow of data from source to destination, highlighting the feedback loops between individual steps in the approach, and by capturing the iterative and collaborative nature of system definition. As shown in Figure 9, the approach relies heavily on engineers (both design and safety engineers) to perform different safety analyses, add system design and safety data into the SysML model, review the generated safety artifacts, update the safety artifacts (if required), and modify the architecture (if necessary). Since the framework integrates the system design and safety life cycle, it encourages collaboration and communication between design and safety engineers as they incrementally add data into the SysML model. The next section describes each step of the approach in greater detail.

Environment Variable	Variable State
Weather	Clear, wind, rain, snow, fog
Roadway type	Interstate highway, undivided highway, local roads, etc.
Traffic zones	School zone, construction zone, residential area, tunnels, garages
Traffic conditions	Vehicle in front, vehicle behind, vehicle front and behind, no vehicles present nearby
Vehicle speed	Very high (>80mph), High (60mph < V $\leq$ 80mph), Medium (30mph < V $\leq$ 60mph), Low ($\leq$ 30mph)

Table 1: Example of environment variables and their states for an automobile.

4.2.1 Develop CONOPS and create failure operational scenarios

The system design and safety life cycle begin by identifying the set of system capabilities that the system must have. Developing the system's concept of operations and exploring its operational environment helps understand and refine stakeholder needs. For safety assessment, this step is crucial as it defines and captures the set of operational scenarios during which the system might encounter a hazard or unsafe conditions that could lead to an accident. The operational scenario is defined by identifying a set of environment variables (and its states) that cover the range of scenarios the system can encounter in its intended operational environment. Table 1 illustrates this definition process for an automotive system. The table provides a list of possible environment variables and states that an automobile could encounter in its operational environment. A variable state is selected for each environment variable and combined to create an operational scenario. For example, a potential operational scenario could be a vehicle is travelling on an interstate highway at high speed (60 mph $\leq$ V < 80 mph) with another vehicle in front on an icy surface with no pedestrians present.

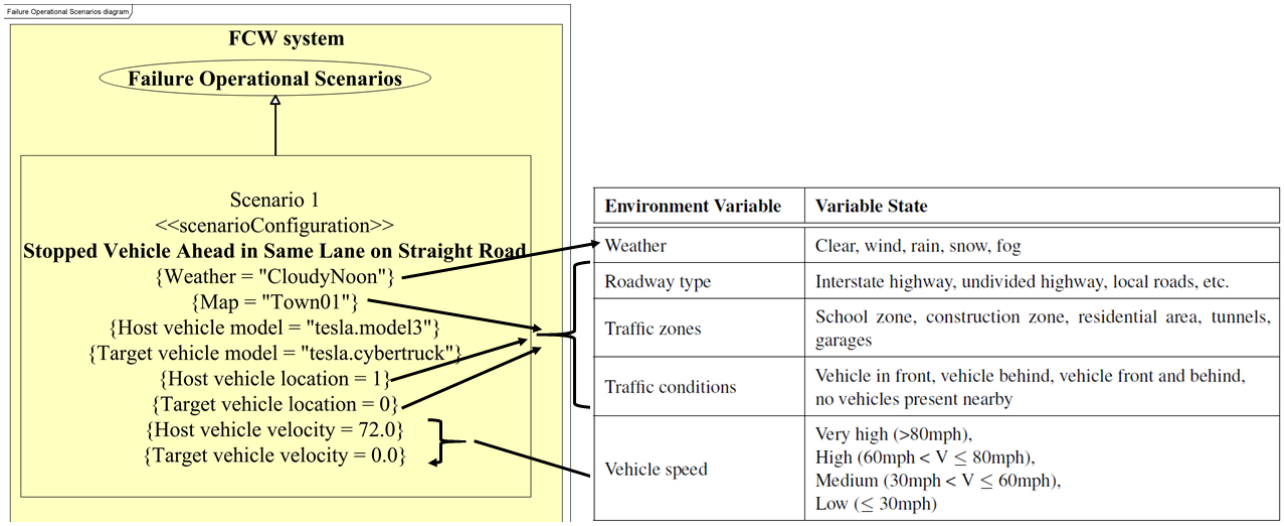


Figure 10: Mapping used to define tagged values of the «scenarioConfiguration» stereotype to configure the failure operational scenario in CARLA

These operational scenarios serve as a starting point for the hazard and risk assessment. However, they are also used to configure scenarios in the CARLA simulator during fault injection (i.e., the safety verification stage). The text-based operational scenario defined using the environment variable and its states are used to create a failure operational scenario (i.e., the operational scenario to be configured in CARLA for running fault injection) and is stored using the «scenarioConfiguration» stereotype in the SysML model. The tagged values of this stereotype represent the parameters of the simulator that are configurable; different failure operational scenarios can be configured by varying the tagged values. It should be noted that the type and value of the parameters are closely coupled to the features and capabilities of the simulator used for fault injection. Simulators differ in available capabilities and configurable parameters; the ability to closely re-create the operational scenario depends on these capabilities and parameters.

(a) Map containing 2-way lanes and T-intersections

(b) Map containing multi-lane roundabouts in an urban environment.

(c) Map containing exit ramp off a highway.

(d) 3-D rendering of map (a)

(e) 3-D rendering of map (b)

(f) 3-D rendering of map (c)

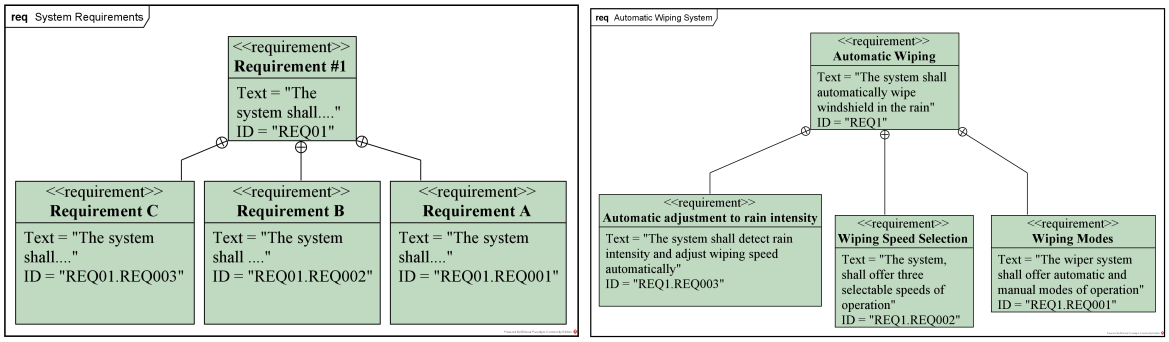
Figure 11: 2-D map layouts and the corresponding 3-D rendering in CARLA illustrating the different roadway types and traffic zones that can be simulated in CARLA. The yellow dot on the 2-D maps identifies the current location of the vehicle on the map

The tagged values in the «scenarioConfiguration» stereotype that configure the simulator are - Map name, weather, host and target vehicle model, host and target vehicle location, and host and target vehicle velocity. Since the these tagged values are fairly non-intuitive, Figure 10 highlights the mapping used to translate the text-based operational scenario into the failure operational scenario in greater detail. The *Weather* tagged value is straightforward - it is used to configure the weather conditions in CARLA. It determines the time of the day, which influences the lighting conditions (i.e., noon or sunset) and the precipitation (i.e., clear, cloudy, light rain, and heavy rain). Currently, the supported weather conditions include ClearNoon, CloudyNoon, WetNoon, WetCloudyNoon, SoftRainNoon, MidRainyNoon, HardRainNoon, ClearSunset, CloudySunset, WetSunset, WetCloudySunset, SoftRainSunset, MidRainSunset, and HardRainSunset. The *Map* tagged value is used to select the roadway type and traffic zones. Figures 11 (a), (b), and (c) show the 2-D layout of a few maps currently

available in CARLA and Figures 11 (d), (e), and (f) show the rendering of the corresponding maps in CARLA. Each map has a different set of possible traffic zones and roadway types- Map 02 contains residential zones with 2-way lanes and T-intersections, Map 03 contains multi-lane roundabouts, and Map 06 contains highways with ramp exits. The map chosen is based on the specific roadway type and traffic zones that needs to be configured in CARLA in order to re-create the operational scenario identified using the environment variables and its states. The *Host vehicle location* and *Target vehicle location* tagged value determine the start and end location of the vehicle and helps in controlling the route the vehicle follows. These values are selected such that the vehicle will encounter the specific roadway type, traffic zone, and traffic condition identified in the text-based operational scenario. The *Host and Target vehicle model* determines the type of vehicle that is configured in CARLA. The simulator has a wide range of available vehicle models, but they primarily differ in appearance. The type of vehicle model selected does not influence the results of the safety assessment itself. Finally, the *Host and Target vehicle velocity* determines the speed at which the vehicle(s) travels from the start location to the target location.

4.2.2 Identify system requirements

System requirements are captured by translating the system capabilities (identified by the concept of operation) and stakeholder needs into requirements. Requirements are stored in the SysML model using a requirements diagram, as shown in Figure 12.



(a) A generic requirements diagram (b) Requirements diagram for an automatic wiper system

Figure 12: System requirements modeled in SysML using a requirements diagram.

The system requirements do not directly influence the safety assessment (i.e., neither safety analysis nor safety verification). These requirements are considered non-safety related requirements - they focus on the functionality of the system rather than the safety. Nevertheless, they are an important aspect of the system design life cycle; the system architecture (functional and/or logical architecture) must satisfy these requirements and is an important step in the ISDS framework as well.

4.2.3 Perform system hazard and risk assessment (system HARA)

A system HARA consists of three steps: a) identify possible system-level hazards that the system might encounter, b) evaluate the risk that each hazard poses to the system and assign a safety integrity level to each hazard to determine the target level for risk-reduction strategies, and c) define system-level safety requirements, also called safety goals, to mitigate the hazards. The input to system HARA is a list of system capabilities and the output from system HARA is a list of system-level hazards and their corresponding safety goals. The following section provides a description of the three steps involved in performing a system HARA:

	Class			
	S0	**S1**	**S2**	**S3**
Description	No injuries	Light and moderate injuries	Severe and life-threatening injuries (survival probable)	Life-threatening injuries (survival uncertain), fatal injuries

Table 2: Classes of severity from [1, Tab. 1]

	Class				
	E0	**E1**	**E2**	**E3**	**E4**
Description	Incredible	Very low probability	Low probability	Medium probability	High probability

Table 3: Classes of exposure from [1, Tab. 2]

	Class			
	C0	**C1**	**C2**	**C3**
Description	Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable

Table 4: Classes of controllability from [1, Tab. 3]

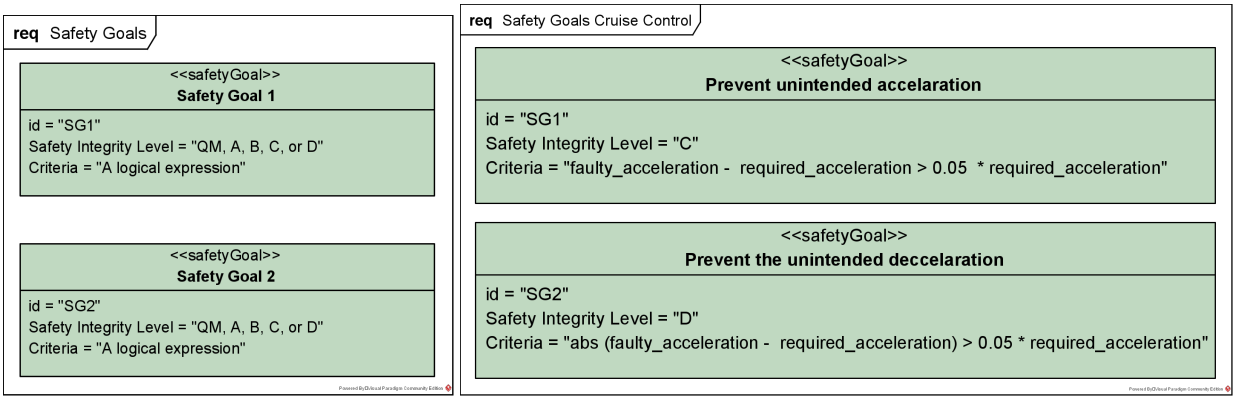
a) Identify system-level hazards: Hazard analysis is performed on the system capabilities to identify potential system-level hazards. These hazards represent states of the system that could pose significant risk to the system itself or its environment, such as loss of life or damage to property. The framework uses the HAZOP method (see chapter 2, section 1 for details on HAZOP) to perform the hazard analysis. Guide words such loss of function, too early, too late, not requested, more, less etc. are used to brainstorm possible system-level hazards. For example, applying the HAZOP technique on an automotive cruise control system, which helps maintain a steady velocity for the vehicle, can result in the following system-level hazards - excessive acceleration (using the "more" guide word), insufficient acceleration (using the "less" guide word), unexpected loss of cruise control (using the "loss" guide word), etc.

b) Evaluate risk and assign safety integrity level: The risk assessment framework of the ISO 26262 standard (Clause 6 in [1]) is used to identify the risk associated with system-level hazards. The framework evaluates the potential crash situation that arises when the system-level hazards occur in the context of different operational scenarios (i.e., scenarios developed using Table 1). For example, a potential crash situation is a vehicle travelling on an interstate highway at high speed (60 mph $\leq$ V < 80 mph) on an icy surface and the cruise control system malfunctions, causing the excessive acceleration hazard to occur. Given a crash situation defined as such, the standard uses three metrics to evaluate the associated risk - the exposure (i.e., the probability of encountering such a scenario), the severity (i.e., the extent of harm to individuals or the system if the hazard occurs in the given scenario), and controllability (i.e., the ability to avoid the specified harm by actions from individual or design control measures). Table 2, 3, and 4 highlight how the classes for severity, exposure, and controllability are defined in the ISO 26262 standard. Once the class for each of the three metrics are identified, Table 5 is used to determine the safety integrity level. If multiple safety integrity levels are applicable to a hazard, the worst-case level is selected.

c) Define safety goals: Safety goals represent high-level (system-level) safety requirements defined for each system-level hazard. For the cruise control system, the safety goal would be, *"The system shall prevent*

Severity class	Exposure class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Table 5: Determining the safety integrity level based on classification of severity, exposure, and controllability [1, Tab. 4]



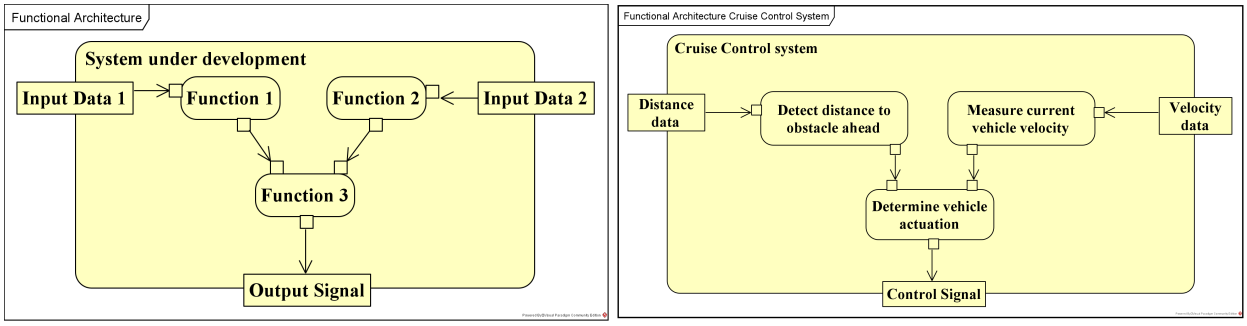
(a) A generic safety goals requirements diagram (b) Requirements diagram for safety goals of a cruise control system

Figure 13: Safety goals modeled in SysML using a requirements diagram.

unintended acceleration". The safety goal would have a safety integrity level based on the system HARA performed on the hazard. Safety goals are stored in the SysML model using a requirements diagram under the «safetyGoal» stereotype, as shown in Figure 13. The tagged values of this stereotype include the id, the safety integrity level, and the criteria. The id provides a unique identifier for the SysML element and is prefixed with the characters "SG" (i.e., SG1, SG2, etc.). The safety integrity level of the safety goal is stored as a character data type (i.e., QM, A, B, C or D). The criteria represent the logical expression that is used to formalize the text-based requirement and will be used to configure the safety goals in the CARLA simulator. The expression is defined to include variables or metrics that represent the safe behavior of the specific system under study. For a cruise control system, the variable could be acceleration. For a collision alert system, this variable could be time-to-collision. Finally, while defining this expression, care must be taken to ensure that the variable names match the naming convention used in developing the client algorithm in CARLA. An underlying assumption in the ISDS framework is that the naming convention in the SysML model match with those used by the simulator. Figure 13 (a) shows the generic format for defining safety goals in the SysML model. Figure 13 (b) illustrates a subset of safety goals for a cruise control system.

4.2.4 Develop functional architecture

The functional architecture is developed by translating the system requirements into functions and capturing the interactions between the functions. The functional architecture is defined in the SysML model using an activity diagram and interactions between functions are modeled via flow ports. Figure 14 (a) shows hows functions are modeled in an activity diagram to develop the functional architecture. Figure 14 (b) illustrates a simplistic view of a cruise control system's functional architecture modeled in SysML.

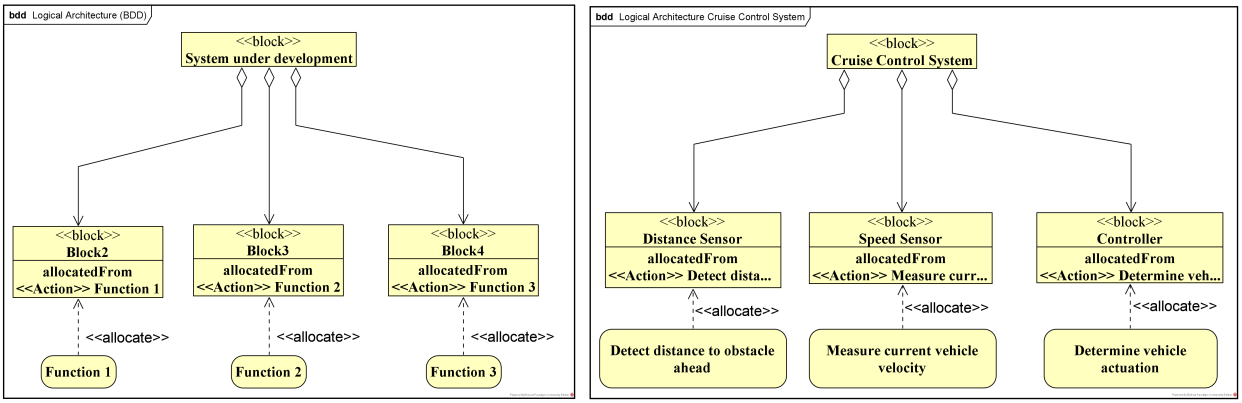


(a) A generic representation of a functional architecture (b) Functional architecture of a cruise control system

Figure 14: Functional architecture modeled in SysML using an activity diagram.

4.2.5 Develop logical architecture

The logical architecture represents the system's structural design and is developed by identifying the logical blocks of the system (i.e., sub-systems), allocating the system functions to the blocks, and identifying the interconnections or data flows between each block. The logical architecture is defined in the SysML model using a Block Definition Diagram (BDD) and an Internal Block Diagram (IBD).

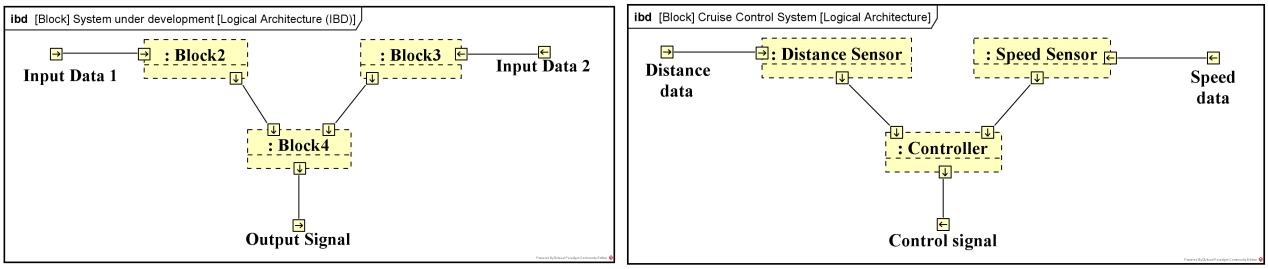


(a) A generic representation of a logical architecture (BDD) (b) Logical architecture (BDD) of a cruise control system

Figure 15: Logical architecture modeled in SysML using a block definition diagram.

The BDD captures the blocks or sub-systems of the system and highlights the allocation of functions, identified in the functional architecture, to their respective blocks, as shown in Figure 15 (a). Figure 15 (b) illustrates a simplistic view of a cruise control system's logical architecture modeled in SysML. An IBD captures

the interconnections and data flows between the blocks modeled in the BDD, as shown in Figure 16 (a). Figure 16 (b) illustrates a simplistic view of a cruise control system's IBD modeled in SysML.



(a) A generic representation of a logical architecture (IBD)

(b) Logical architecture (IBD) of a cruise control system

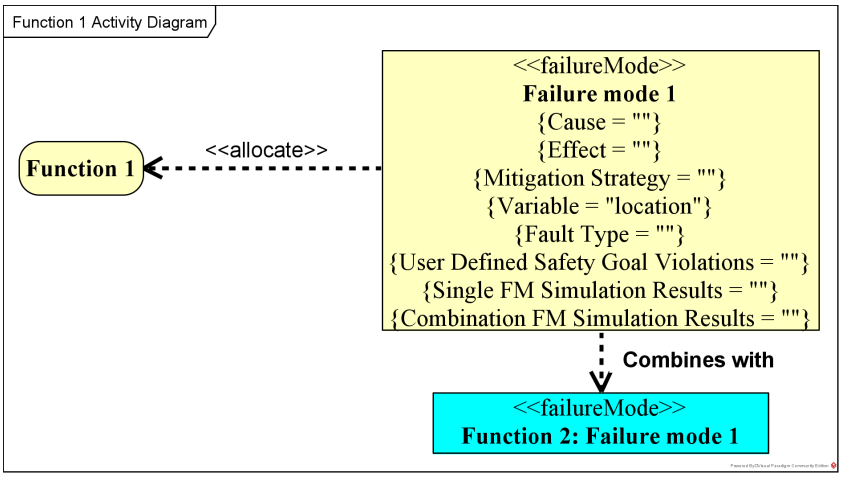
Figure 16: Logical architecture modeled in SysML using a block definition diagram.

4.2.6 Perform sub-system hazard and risk assessment (sub-system HARA)

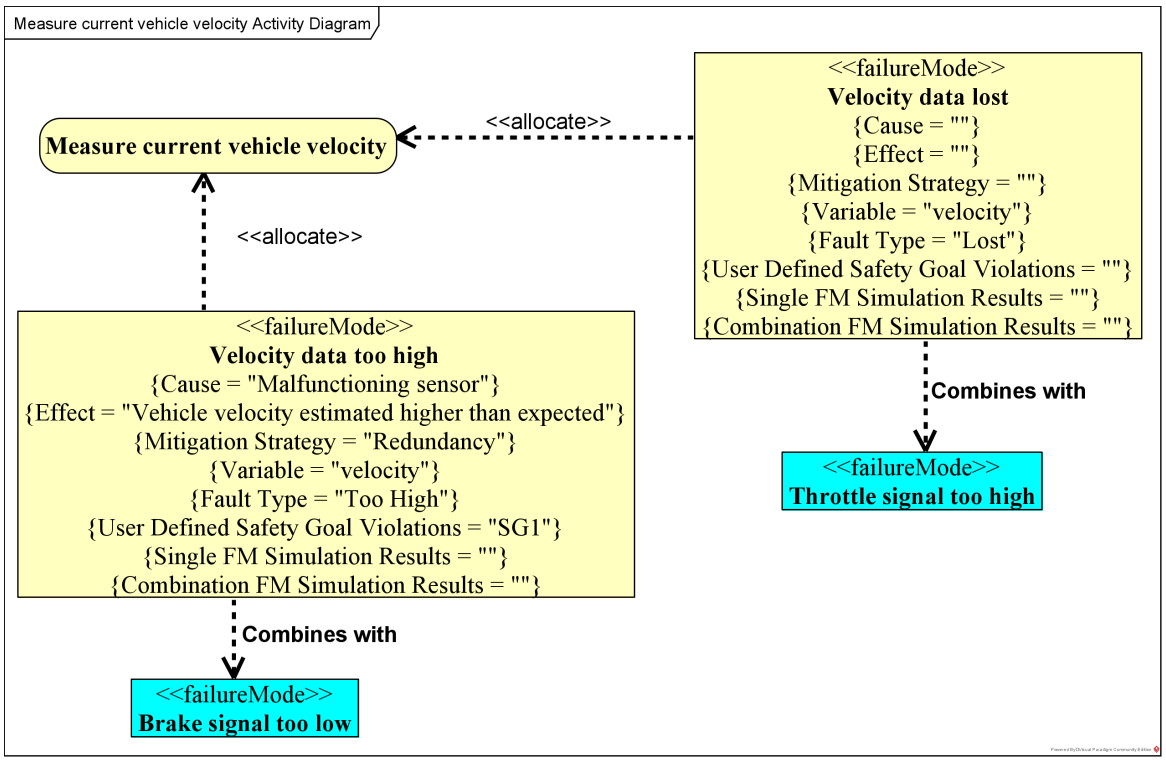
A sub-system HARA involves performing a hazard analysis on the functions allocated to the different blocks or sub-systems in the logical architecture. Similar to the system HARA, the sub-system HARA is performed using the HAZOP technique. Guide words such as loss, more than, less than, intermittent, wrong direction, and delay are combined with the function parameter to brainstorm possible functional hazards. These hazards represent the function's failure mode, i.e., the manner in which the function fails to provide the intended behavior. The failure modes are captured in a new activity diagram that is linked to the corresponding function in the functional architecture. This linking feature in SysML helps maintain consistency and traceability between model elements. Figure 17 (a) and (b) illustrates how failure modes are captured in the SysML model.

A function's failure mode is captured using the «failureMode» stereotype in the SysML model. The FMEA is performed to enter the data for the tagged value i.e., the cause and effect of the failure mode, a mitigation strategy to reduce the impact of the failure mode to an acceptable level, and the safety goals that will be violated when the failure mode occurs (estimated by an engineer). The tagged values can either be entered in the SysML model or be left blank in the SysML model and entered in the FMEA table that is generated in the the next step. The variable and fault type are defined to characterize the failure mode in the CARLA simulator i.e., "how can the failure mode's behavior be represented in the simulator and recreated during the fault injection simulation". The variable represents the type of data that the failure mode affects in the simulator such as location, velocity, throttle, steering, or timing. The fault type represents the behavior of the failure mode and is defined based on the HAZOP guide word of the failure mode. The fault types supported by the ISDS framework include too high, too low, lost, delay, intermittent, and inverse. The single FM simulation results and combination FM simulation results act as placeholders to store the feedback from the safety verification in the SysML model.

The failure modes are linked to each function using the «allocate» relationship. The source of the connection is the function, and the destination of the connection is the failure mode. Engineers can also model the failure modes that combine, i.e., multiple failure modes occurring at the same time. This relationship is modeled using the "combines with" dependency. In Figure 17 (a), function 1's failure mode 1 (shown in yellow) combines with function 2's failure mode 1 (shown in blue). Figure 17 (b) illustrates the failure modes for the "*Measure current vehicle velocity*" function of a cruise control system. It is allocated two failure modes - velocity data too high and velocity data lost. For the *Velocity data too high* failure mode, the cause, effect, mitigation strategy, and user-defined safety goal violation are added in the SysML model. To characterize the failure mode in CARLA,



(a) Failure modes defined for a generic function.



(b) Failure modes for the "*Measure current vehicle velocity*" function of the cruise control system.

Figure 17: Failure modes modeled in SysML using an activity diagram and the «failureMode» stereotype.

the variable is selected as *velocity* (as the failure mode impacts the vehicle velocity sensor) and the fault type is selected as *Too High*. It is also modeled to combine with the *Brake signal too low* failure mode from the *Determine vehicle actuation function* from Figure 15 (b). For the *Velocity data lost* failure mode, the tagged values are left empty and can be entered in the FMEA table generated in the next step. To characterize this failure mode in CARLA, the variable is selected as *velocity* (again, the failure mode impacts the vehicle velocity sensor) and the fault type is selected as *Lost*.

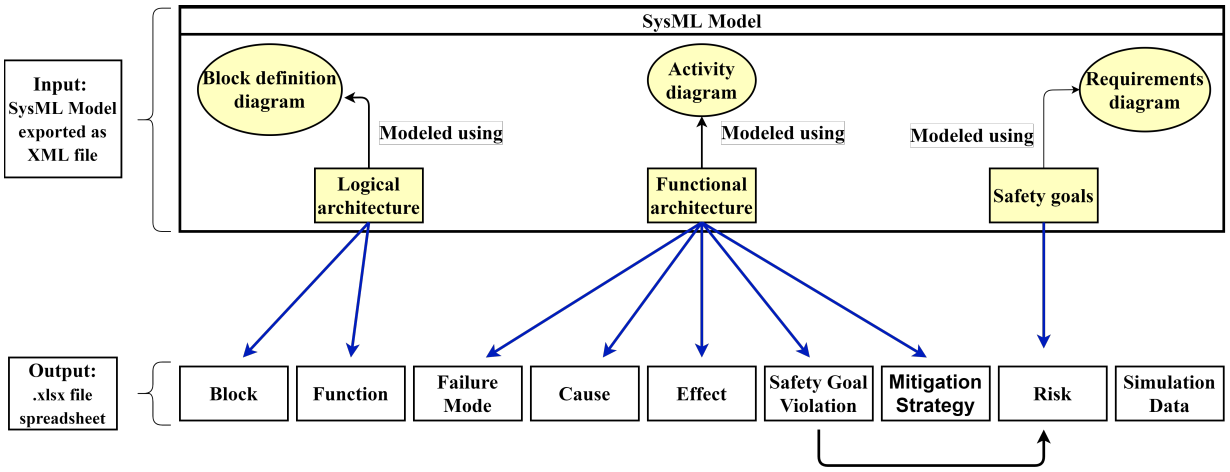


Figure 18: Mapping of SysML diagrams and model elements to FMEA fields

4.2.7 Generate safety artifacts

Once the failure modes are defined for all system functions and the data for the failure mode's tagged values are entered (partially or completely), the ISDS framework generates a single system FMEA table and multiple system fault trees, one fault tree for each safety goal. The SysML model is exported as an XML file and the framework's algorithms, which are developed in Python, parse through the file to generate the safety artifacts. The framework uses python's ElementTree XML API to parse the XML file.

The ISDS framework uses the XML file of the SysML model as opposed to the XMI file, which is billed as an inter-operable XML schema. In theory, the XMI schema allows tools to transfer SysML models between tools (i.e., different modeling tools can import the XMI file and re-generate the same SysML model). However, in practice the XMI schema is not truly inter-operable (i.e., tool support for the XMI schema is still limited). For example, the XMI file of the SysML model generated by Visual Paradigm [107] could not be imported by Enterprise Architect [108] to re-generate the SysML model, and vice-versa. In addition to the limited tool support for XMI files, XML files are also easier to parse since they are enriched with more model details. However, the limitation of using XML files is that it can only be imported by the tool that generates the file. An XML file generated by Visual Paradigm can only be read and imported back into Visual Paradigm. Consequently, the ISDS framework is dependant on the modeling tool used to create the SysML model. But, as tool support for the XMI schema improves to make it a truly inter-operable file format, a future avenue to improve the ISDS framework would be to update the framework's algorithm to handle XMI files as opposed to XML files. The next section describes how the framework generates the FMEA table and fault trees from the XML file of the SysML model.

1. Generate FMEA table: The algorithm to generate the FMEA table takes the XML file of the SysML model as input and generates a spreadsheet (.xlsx file) as an output with the contents of the FMEA. Figure 18 illustrates the diagram and model element from which data is extracted to generate the FMEA table. First, the algorithm iterates over the blocks in the BDD of the system's logical architecture and extracts each function allocated to the block. Next, the failure modes allocated to the functions in the functional architecture are identified and the required safety data is extracted, i.e., the cause, effect, mitigation strategy, and safety goal violations. The risk associated with the failure mode is determined by identifying the safety goal violations of the failure mode and extracting the safety integrity level of the corresponding safety goal from the requirements diagram

where the system safety goals are defined. If the failure mode violates multiple safety goals, the highest order safety integrity level becomes the failure mode's associated risk. Finally, the simulation data column is generated to store feedback from the safety verification (i.e., the results from the fault injection simulation). At this stage of the framework, the simulation data column is empty; after safety verification it will store the safety goals that were violated when the failure mode was injected during the fault injection simulation. If data is missing in the SysML model (i.e., a tagged value of the failure mode does not contain data), the corresponding cell in the FMEA table contains the '-' symbol and indicates that the information is to be filled by an engineer. The algorithm for generating the FMEA table is shown in Algorithm 1.

Block	**Function**	**Failure Mode**	**Cause**	**Effect**	**Safety Goal Violation**	**Risk**	**Mitigation Strategy**	**Simulation Data**

Table 6: Format and fields of the generated FMEA table

Algorithm 1: Generate System FMEA

Input: XML of SysML model
Output: Spreadsheet of system FMEA in .xlsx format

```
SET root = root node of XML
Find Blocks in BDD
for each SysML Block do
    Add Block name to spreadsheet
    Find all Functions allocated to the Block
    for each Function do
        Add Function name to spreadsheet
        Identify activity diagram of Function
        Extract Failure Modes of Function
        for each Failure Mode do
            Add Failure Modes name to spreadsheet
            if tagged value is empty then
                Add '-' in spreadsheet
            else
                Add Cause, Effect, Mitigation Strategy, Safety Goal Violations to spreadsheet
                Extract Safety Integrity Level of allocation safety goal
                SET Risk = Worst case safety integrity level
            end if
        end for
    end for
end for
```

2. Generate fault trees: The next safety artifact generated from the SysML model are the system fault trees. Like the artifact generation process for the FMEA table, fault tree generation is automatic, instantaneous, and consistent with the SysML model. In the ISDS framework, a unique fault tree is generated for each safety goal. The structure of the fault tree follows the generic format shown in Figure 19. The top-level event of the fault tree is the safety goal violation. The intermediate-level events are developed based on the system structure, i.e., the logical architecture. Each intermediate-level can be one of two categories: a) internal failure of the block, due to faults within the block itself, or b) failure in input to the block, due to faults in the neighbouring connected block, which in turn can be due to an internal failure or a failure in its input and

so on. The structure is repeated until the lowest-level blocks are traversed, i.e., blocks that do not have any inputs. The internal failure of these blocks represents the bottom-level events. At each intermediate-level, the internal failure of a block is represented using the failure modes of the block connected to an *OR* gate. The failure modes that represent an internal failure of the block are limited to those that violate the given fault tree's safety goal only. If failure modes combine, they connect to an *AND* gate, which in turn connects to the *OR* gate. The combination can either be of failure modes that each individually violate the safety goal but also combine to violate the safety goal or failure modes that individually do not violate the safety goal but together can combine to violate the safety goal.

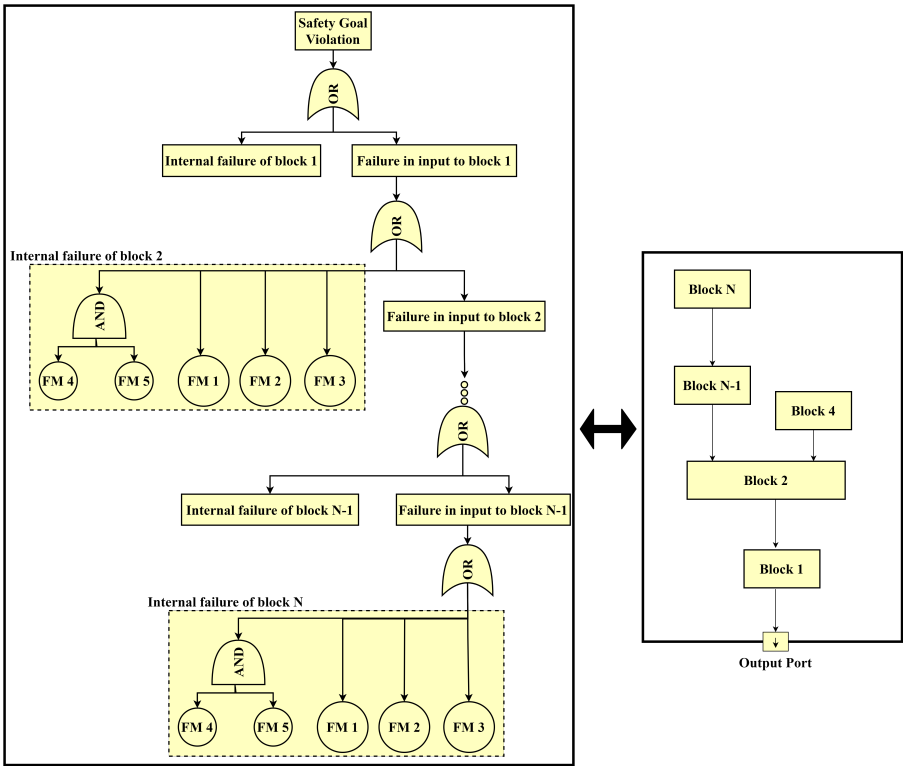


Figure 19: Generic structure of the generated fault tree.

The algorithm for generating fault trees, shown in Algorithm 2 and 3, takes the XML file of the SysML model as input and generates a unique fault tree for each safety goal violation. Figure 20 illustrates the SysML diagram and model element from which data is extracted to generate the fault tree. First, the algorithm extracts the safety goals from the requirements diagrams. Next, the output port is identified in the logical architecture's IBD. Starting at the output port, the algorithm traverses through all the blocks in the IBD and creates the equivalent tree structure in python, i.e., an N-ary tree. An N-ary tree is a tree structure where each node can have any number of children, as opposed to binary trees, in which each node can only have two children. The N-ary tree in python recreates the structure of the system from the IBD, where each node in the tree represents a block in the IBD. The algorithm uses the N-ary tree structure to create a fault tree based on the structure shown in Figure 19. Algorithm 2 shows the pseudo code for generating the N-ary tree.

Figure 20 shows the SysML diagrams and model elements used to create the different components of the fault tree. The N-ary tree is sufficient to create the fault tree as it contains the required information about the blocks and their interconnections; however, each event in the fault tree is label as *Internal failure of "block"*. The logical architecture and functional architecture are used to further refine the intermediate events,

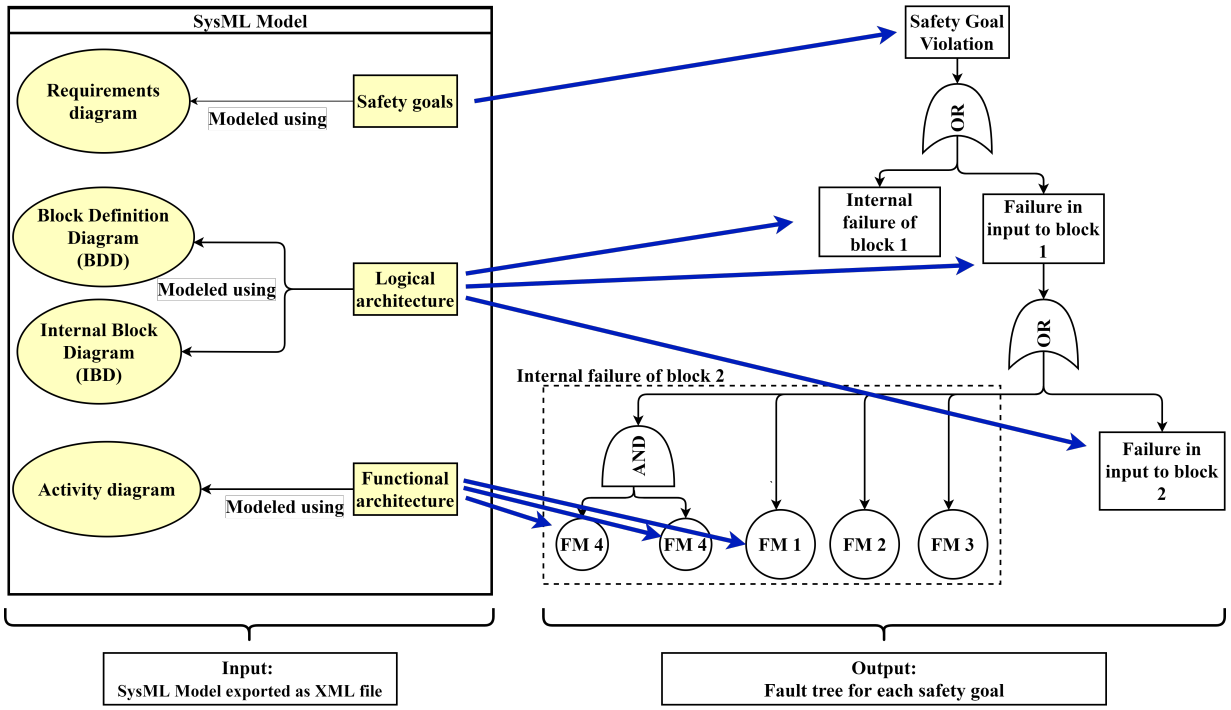


Figure 20: Mapping of SysML diagrams and model elements to fault trees

i.e., to highlight the failure modes of the block that violate the safety goal applicable to the fault tree. First, the algorithm iterates over the N-ary tree and for each node in the tree, it finds the corresponding block in the logical architecture's BDD. Next, the functions allocated to the block are identified and the failure modes allocated to the functions are extracted from the functional architecture. The *user-defined safety goal violations* tagged value is used to only select the failure modes that violate the given safety goal. Finally, the extracted data and N-ary tree are used to develop commands that can be read by software to create the fault tree. Algorithm 3 shows the pseudo code for generating the code for the fault tree. The framework uses the Open Reliability software [109] to read the generated fault tree code and visualize the fault tree.

4.2.8 Feedback from FMEA to SysML model

Once the safety artifacts are generated, the safety engineer can further analyze the FMEA table and update the cells in the spreadsheet as required. This is an opportunity for the engineer to fill in information that was missing or left empty in the SysML model. Once the FMEA table is updated, the changes are updated in the SysML model. Algorithm 4 shows how the data is updated in the SysML model. The algorithm parses the edited spreadsheet and iterates through the previously generated XML file of the SysML model to update cause, effect, mitigation strategy, safety goal violation, and risk for each failure mode to the SysML model. As a result, any changes made to the FMEA table by the engineer are automatically reflected in the SysML model. If any changes are made to the safety goal violations of a failure mode, the fault tree generation algorithm can be re-run on the updated SysML model to update the affected fault trees.

Based on the mitigation strategies identified in the FMEA for each failure mode, the design and safety engineers can modify the functional and/or logical architecture. Design and safety engineers collaborate to identify possible design changes required to realize the mitigation strategies. Changes could include adding a

Algorithm 2: Generate N-ary Tree

```
Input: XML of SysML model
Output: N-ary-Tree, List_of_safety_goals
SET root = root node of XML;
Function Find safety goals(root):
    SET list_of_safety_goals = []
    FIND requirement elements in root
    for each requirement element do
        if stereotype is Safety Goal then
            add Safety Goal to list_of_safety_goals
        end if
    end for
    return list_of_safety_goals
Function Generate-N-ary-Tree (list_of_safety_goals, root):
    Find system output port in IBD
    list_of_blocks = []
    Add output port to list_of_blocks
    while list_of_blocks > 0 do
        for each_block in list_of_blocks do
            new_list_of_blocks = []
            Find Block connected to block
            Add Block to new_list_of_blocks
            Create python tree node
            Set node_name to Block
        end for
        Set list_of_blocks = new_list_of_blocks
    end while
    return N-ary tree
```

new function in the functional architecture and allocating it to an either a new block or existing block in the logical architecture.

4.2.9 Derive component safety requirements

Component safety requirements represent hardware and software safety requirements that are derived to prevent or mitigate the effects of one (or more) failure modes of a given block or sub-system's function. These requirements are the lowest-level safety requirements and can be traced up to the safety goals. Component safety requirements are allocated to a function's failure mode in the activity diagram of the system's functional architecture and are defined using a requirements diagram under the «componentSafetyRequirement» stereotype. The tagged values of this stereotype include the id, the criteria, violation simulation data variable, and the safety integrity level. Similar to the safety goals stereotype, the criteria represent the logical expression that formalizes the text-based requirement and will be used to configure the component safety requirement in the simulation. The violation simulation data is a placeholder that stores the results of the feedback from the fault injection simulation. If the requirement was violated during safety verification (i.e., during the fault injection simulation run), the injected failure mode that caused the violation, the failure operational scenario during which the violation occurred, and the simulation time when the violation occurred are captured. Finally, the safety integrity level is inherited from the safety goal that is violated by the failure mode that the component

Algorithm 3: Generate System FTA

```
Input: N-ary tree, List_of_safety_goals, XML of SysML model
Output: System fault trees
SET root = root node of XML;
Function Generate-Fault-Tree (N-ary tree, root_XML, list_of_safety_goals):
  for each_safety_goal in list_of_safety_goals do
      create fault tree container F_TREE
      level = 1
      if N-ary tree is None then
        return
      else
        level = level + 1
        node_list = []
        Append N-ary tree root to node_list
        while size of node_list > 0 do
          Set node_children = length of node_list
          while node_children > 0 do
            current_node = first element in node_list
            remove current_node from from node_list
            Add OR gate at level in F_TREE
            Add event at level as failure in input to block in F_TREE
            Add Internal-Failure-of-Block (current_node, root_XML, each_safety_goal, level, F_TREE)
            Add all children of current_node to node_list
            node_children = node_children - 1
          end while
        end while
      end if
  end for
  return F_TREE
Function Internal-Failure-of-Block (current_node, root_XML, each_safety_goal, level, F_TREE):
  Find Block in BDD of logical architecture = current_node
  Find Functions and Failure Modes of Block
  if Failure Mode's safety goal violation = each_safety_goal then
      Add Failure Mode at level to OR gate
      if Failure Mode combination exists then
        Add AND gate to previous OR gate
        Add Failure Mode to AND gate
      end if
  end if
```

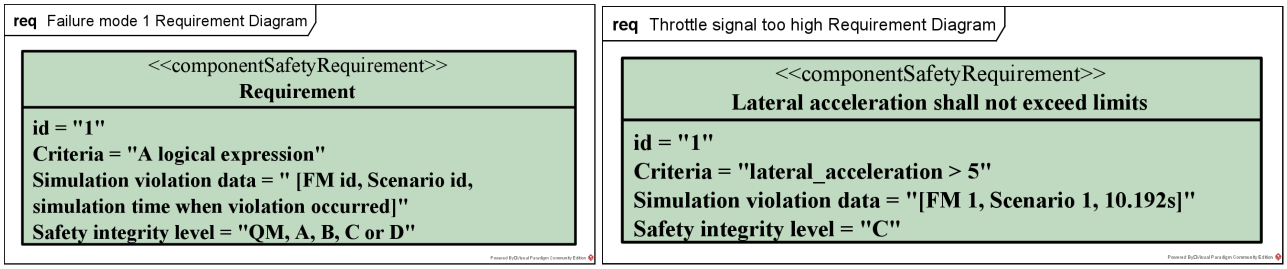
safety requirement is allocated to. If multiple safety goals are applicable, the worst-case safety integrity level is inherited. Figure 22 illustrates the traceability from safety goals to component safety requirements. Component safety requirements inherit the safety integrity level from the safety goal that is allocated to a given failure mode.

To satisfy the component safety requirements in the system architecture, the design and safety engineers can modify the functional and/or logical architecture, if necessary. Design and safety engineers collaborate to identify possible design changes required to realize the requirements. Changes could include adding a new function in the functional architecture and allocating it to an either a new block or existing block in the logical architecture.

This step marks the end of the model-based safety analysis approach and the completion of the system

Algorithm 4: Update System FMEA

Input: XML of SysML model, Edited Spreadsheet
Output: Updated Spreadsheet of System FMEA
for each row in spreadsheet **do**
Find *SysML Block* in XML 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 XML file
end for
end for



(a) A generic representation of a component safety requirement (b) Component safety requirement of a cruise control system

Figure 21: Component safety requirements modeled in SysML using a requirements diagram.

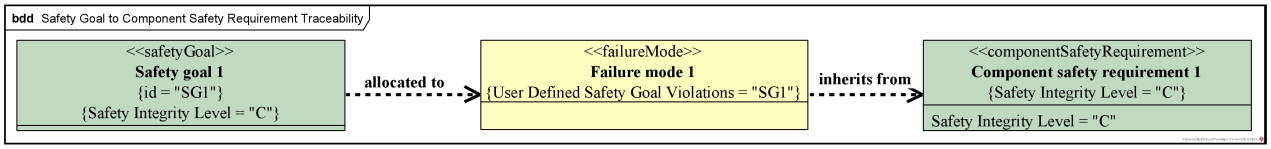


Figure 22: Traceability of safety integrity level from safety goals to component safety requirements

definition stage (or left side of the Vee) of the ISDS framework. To summarize the approach, as the framework traverses down the left side of the Vee, system design and safety data are incrementally added to the SysML model. FMEA tables and fault trees are automatically generated from the SysML model using the framework's algorithm and any changes made to the artifacts are instantly and automatically updated in the SysML model as well. Finally, component safety requirements are derived if required, to mitigate the effects of the functional failure modes. This step completes the system definition phase of the ISDS framework and the SysML model is enriched with the required system design data, safety data, and simulation-specific properties. The focus of the framework shits towards verifying the safety of the system design. The next section will describe the model-based safety verification approach used in the ISDS framework.

4.3 Model-based safety verification approach

The model-based verification approach is adopted in the right side of the ISDS framework (i.e., system verification phase). In this phase, the focus of the framework is to verify if the system design satisfies the safety requirements. The verification is performed using a simulation-based fault injection technique. The approach configures the system under development (or the client) in CARLA and observes system behavior in the absence and presence of faults for different operational scenarios. In contrast to the model-based safety analysis approach, the model-based safety verification approach is highly automated and does not include an engineer-in-the-loop. The automation is enabled by the use of a fault injection engine, which is a collection of algorithms for extracting system design and safety data from the SysML model, configuring the CARLA simulator, executing fault injection runs in the simulator, and analyzing the resulting data to determine if any safety requirements were violated. Figure 23 provides an overview of the model-based safety verification approach and highlights the components of the fault injection engine. The safety verification approach in the ISDS framework consists of three components:

1. SysML model, which contains the system design and safety data developed during system definition.
2. CARLA simulator [94], which is a high-fidelity simulator of autonomous automotive systems.
3. Fault injection engine, which used to configure, run, and analyze the simulation runs to compute safety metrics.

The next section describes the three components of the safety verification approach in the ISDS framework in greater detail.

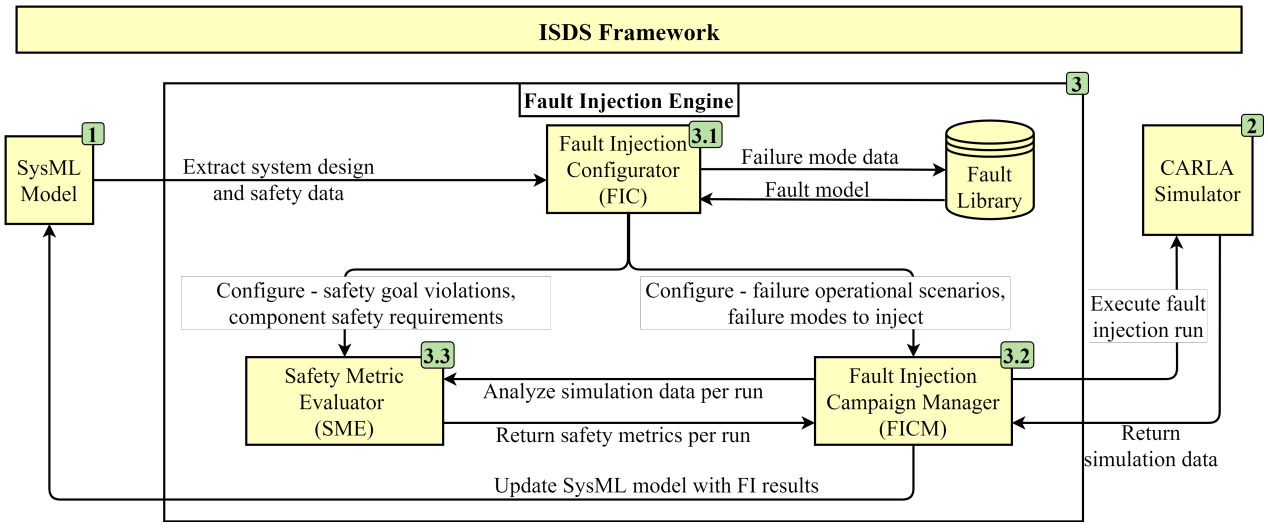


Figure 23: Overview of the safety verification approach used in the ISDS framework.

4.3.1 SysML model

The SysML model contains the design and safety data developed during the system definition phase (left side of the Vee) of the ISDS framework. The model consists of multiple diagrams, each representing a different view of the system, as shown in Figure 24. The SysML model is exported an an XML file and parsed by the Fault Injection Engine to configure the CARLA client. The system design and safety data stored in the SysML model includes the failure operational scenarios, safety goals, functional architecture, logical architecture, and

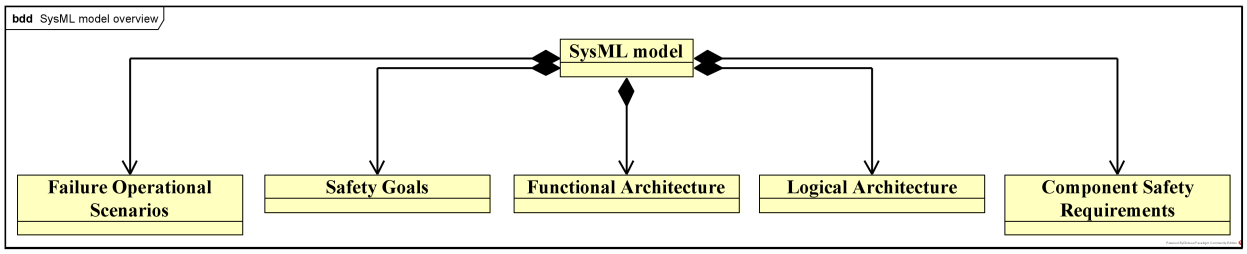


Figure 24: Overview of the diagram types used as well as the system design and safety data stored in the SysML model.

component safety requirements. For details on how the data is stored in the SysML model, please refer to section 4.2.

4.3.2 CARLA Simulator

CARLA is an open-source simulator for autonomous driving systems. Section 4.1.1 provides a detailed description of CARLA's architecture and capabilities. In CARLA, the world module runs the simulation and renders the scene, and the client module represents the system under development and controls the behavior of the system in the simulation. The client module, developed in python, contains parameters that are configured by the system design data, safety data, and simulation-specific properties stored in the SysML model.

4.3.3 Fault Injection Engine

The fault injection engine acts as a bridge between the SysML model and the CARLA simulator–it extracts the system design and safety data from the SysML model, configures the CARLA simulator based on the extracted data, executes the fault injection, verifies the system design against the safety requirements, and updates the SysML model with results from the fault injection. The Fault Injection Engine consists of three components: a) Fault Injection Configurator (FIC), b) Fault Injection Campaign Manager (FICM), and c) Safety Metric Evaluator (SME).

4.3.3.1 Fault Injection Configurator (FIC)

FIC is responsible for extracting system design and safety data from the SysML model and configuring the CARLA simulator to run the fault injection. FIC parses the XML file of the SysML model and extracts the data required to setup the configurable parameters in the CARLA client. Essentially, FIC acts as a bridge between the SysML model and CARLA and ensures that the fault injection reflects the current information in the SysML model. FIC configures the parameters in the simulator by performing six functions: model selecting, configuring failure operational scenarios during which the fault injection is performed, configuring safety goal violations, configuring component safety requirement violations, extracting the failure modes to be injected for each block, and translating failure mode to the fault model. Figure 25 illustrates the SysML model elements and SysML diagrams from which each function extracts the system design and/or safety data. The algorithm used in FIC to perform these functions are provided in Algorithm 5, 6. The next section will describe each of the six functions in greater detail.

1. Model selection: Once the SysML model is populated with the required design and safety data, it is exported as an XML file and parsed by FIC to extract data required to configure CARLA.

Algorithm 5: Algorithm for function 2, 3, and 4 of Fault Injection Configurator (FIC)

```
Input: XML of SysML model
Output: list of safety goals, component safety requirements, and failure operational scenarios.
SET root = XML Model
Function configure failure operational scenarios():
    SET list of failure operational scenarios = []
    find all use case elements in root
    for each use case do
        if stereotype is scenarioConfiguration then
            create scenarioConfiguration object
            for tagged value in use case do
                store tagged value in object member variable
            end for
            append object to list of failure operational scenarios
        end if
    end for
    return list of failure operational scenarios
Function configure safety goals():
    SET list of safety goals = []
    find all requirement elements in root
    for each requirement do
        if stereotype is Safety Goal then
            create SafetyGoal object
            for tagged value in requirement do
                store tagged value in object member variable
            end for
            append object to list of safety goals
        end if
    end for
    return list of safety goals
Function configure component safety requirements():
    SET list of component safety requirements = []
    find all requirement elements in root
    for each requirement do
        if stereotype is component safety requirement then
            create ComponentSafetyReq object
            for tagged value in requirement do
                store tagged value in object member variable
            end for
            append object to list of component safety requirements
        end if
    end for
    return list of component safety requirements
```

2. Configure failure operational scenarios: In the SysML model, failure operational scenarios are stored in a use case diagram. A failure operational scenario is defined using the «scenarioConfiguration» stereotype, as shown in Figure 10. The FIC module defines a python class named *scenarioConfiguration* with member variables identical to the tagged values of the «scenarioConfiguration» stereotype. The module calls the configure failure operational scenarios function, which instantiates an object of the *scenarioConfiguration* class, parses the XML file to find use case elements with the «scenarioConfiguration» stereotype, and

Algorithm 6: Algorithm for function 5 of Fault Injection Configurator (FIC)

```
Input: XML of SysML model
Output: list of failure modes
SET root = XML Model
Function configure failure modes(list of safety goals):
    SET list of failure modes = []
    find all SysMLBLock elements in logical architecture
    for each SysMLBLock element do
        find allocated function and store address
        for action elements in functional architecture do
            if stereotype is Failure Mode and
                ↪ address is equal then
                find allocated failure modes to function
                for each failure mode do
                    create Failure Mode object
                    for tagged value in Failure Mode do
                        store tagged value in object
                    end for
                    append object to list of failure modes
                    find all dependency relationships
                    if dependency is of type Combines with
                        ↪ then
                        Store destination address
                        Find corresponding failure mode
                        Set failure mode combination
                            ↪ for source and destination
                    end if
                end for
            end if
        end for
    end for
    for each failure mode in list of failure modes do
        find failure mode combinations
        create tuple of failure mode combinations
        append to list of failure modes
        remove duplicate tuples
    end for
    return list of failure modes
```

initializes the object's member variables with the data from the corresponding tagged value, i.e., the map name, the host and target vehicle type, the host and target vehicle starting location, the host and target vehicle desired speed, and weather type. This process is repeated for each failure operational scenario defined in the SysML model. This function returns a list of objects that represent the different failure operational scenarios of the system. The CARLA world and client modules are configured using the values of the object variables.

3. Configure safety goal violations: Safety goals are stored in a requirements diagram and are defined using the «safetyGoal» stereotype, as shown in Figure 13. The FIC module defines a python class names *safetyGoal* with member variables identical to the tagged values of the «Safety Goals» stereotype. The module calls the configure safety goal violations function, which instantiates an object of the *safetyGoal* class, parses the

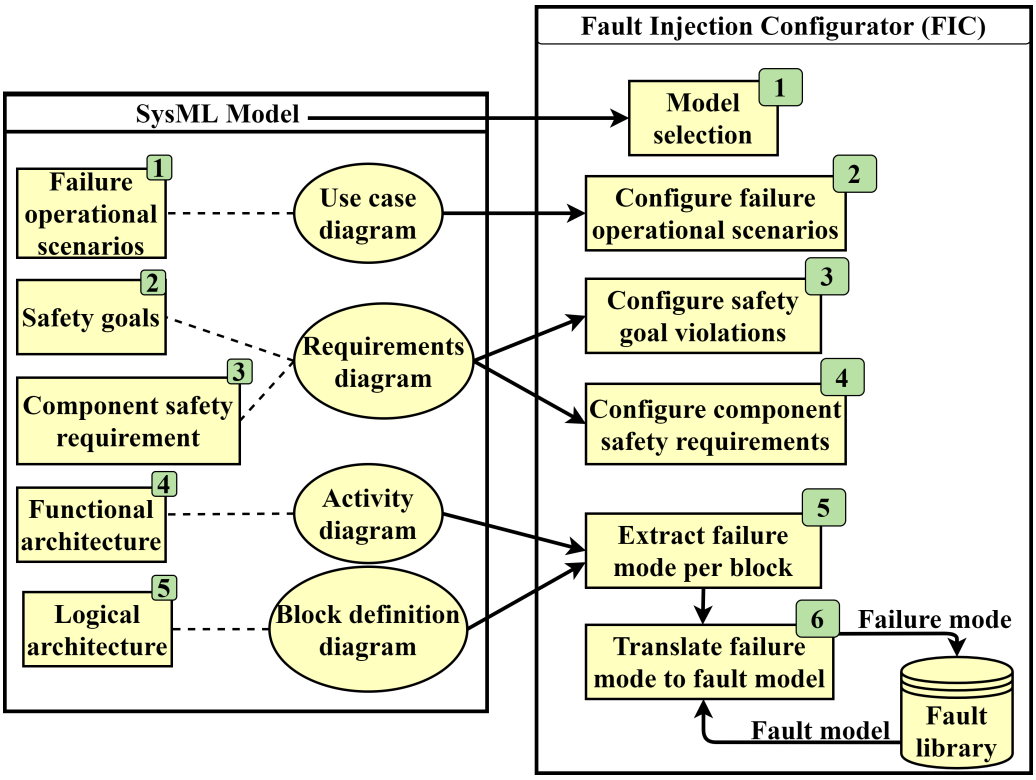


Figure 25: The SysML model elements and SysML diagrams from which each function extracts the system design and safety data

XML file to find requirements defined with the «safetyGoal» stereotype and initializes the object's member variables with the data from the corresponding tagged value, i.e., the name, id, criteria, and safety integrity level. This process is repeated for each safety goal defined in the SysML model. The function returns a list of objects that represent the safety goals of the system. The CARLA client module uses the criteria defined in each safety goal object to evaluate whether the safety goal was violated in the simulation.

4. Configure component safety requirement violations: In the SysML model, components safety requirements are stored in a requirements diagram and are defined using the «componentSafetyRequirement» stereotype. These requirements are allocated to a function's failure mode. The FIC module defines a python class named *componentSafetyRequirement* with member variables identical to the tagged values of the «componentSafetyRequirement» stereotype. The module calls the configure component safety requirement violations function, which instantiates an object of the *componentSafetyRequirement* class, parses the XML file to find requirements defined with the «componentSafetyRequirement» stereotype, and initializes the object's member variables with the data from the corresponding tagged value, i.e., the name, id, criteria, and safety integrity level. This process is repeated for each component safety requirement defined in the SysML model. The function returns a list of objects that represent the component safety requirements of the system. The CARLA client module uses the criteria defined in each component safety requirement object to evaluate whether or not the requirement was violated in the simulation.

5. Extract failures mode per block: The faults injected in CARLA to recreate the faulty behavior of the system are based on the failure modes identified in the FMEA. The FIC module defines a python class named *failureMode* with member variables identical to the tagged values of the «failureMode» stereotype. The configure failure modes function extracts the failure modes in a series of steps: first, the XML file is parsed

to identify the blocks (or sub-systems) defined in the logical architecture. Next, the functions allocated to each block are traced to their activity diagram in the functional architecture. Finally, the failure modes allocated to each function are identified, an instance of the *failureMode* class is created (i.e., an object), and the object's member variables are initialized with the failure mode's corresponding tagged values i.e., the failure mode name, allocated block and function, fault type, variable to be corrupted, and user-defined safety goal violations. The failure mode combinations are extracted by checking if the failure mode block has a "Combines with" dependency relationship with other failure modes. This function returns a list of objects that represent the failure modes and failure mode combinations of the system. The CARLA client uses the failure mode objects to configure and inject faults in the simulation.

6. Translate failure mode to fault model: The last function performed by the FIC module is to translate the failure mode into a mathematical model, called the fault model, that can recreate the intended faulty behavior of the failure mode in CARLA. This translation is based on two parameters - the failure mode "fault type" and the failure mode "variable". As highlighted in section 4.1.2, these two parameters are tagged values for the «failureMode» stereotype and defined while performing sub-system HARA during system definition. The variable represents the type of data that the failure mode affects (i.e., location, velocity etc.) and the fault type represents the behavior of the failure mode based on the HAZOP guide words (i.e., too high, too low, delay, lost etc.).

   To characterize the first parameter for translating the failure mode, i.e., the fault type, the module defines a python class called the *faultLibrary* (represented as "Fault library" in Figure 25), which contains a collection of templates for the different fault types or HAZOP guide words. The class takes the failure mode's fault type as input and matches it with the template defined for that type. For example, if a failure mode's fault type is defined as "Too High", the mathematical model is defined as -

$$data = data + x\%data \tag{4.1}$$

   where data represent the value to be corrupted and x represents the percentage it should be increased. The fault models currently supported by the ISDS framework are shown in Table 7.

Fault Type	Template
Too High	$data = data + x\,\%data$
Too Low	$data = data - x\,\%data$
Inverse	$data = -1 * data$
Intermittent loss	$x = random\ number\ generator\ between\ 0\ to\ 1$ $if\ x > 0.5\ then\ data = 0;\ \ if\ x < 0.5\ then\ data = data$
Lost	$x = data\ (Store\ value\ of\ current\ data)$ $for\ 50\ frames\ \ data = x$ $after\ 50\ frames\ \ data = data$
Delay	$x = data\ (Store\ value\ of\ current\ data)$ $delayed\_data = [data]\ (append\ data\ in\ a\ list)$ $for\ 50\ frames,\ \ data = x\ and\ delayed\_data = [data, data, ...]$ $for\ next\ 50\ frames,\ \ data = delayed\_data\ \ (data\ retrieved\ from\ list\ in\ order)$ $after\ 100\ frames\ \ data = data$

Table 7: Templates for fault models currently supported by the ISDS framework

Similar to the too high fault type, the too low fault type reduces the value of data by a certain percentage. The inverse fault type negates the current value of data. The intermittent loss fault type randomly fluctuates the value of data to be equal to either the actual value or zero. The lost fault type drops data for a certain amount of time, i.e., data is not recoverable. The delay fault type pauses data for a certain amount of time and transmits the data with a time delay, i.e., data is recoverable but with a time delay. The lost and delay fault type use the simulation's frames per second (fps) to determine the time for which the fault occurs. A simulation step occurs after every frame, i.e., after each frame data is refreshed. Using the fps, a time can be assigned to the delay and lost fault type. For example, if the simulation fps is fixed at 25 fps, a delay 50 frames will correspond to a delay for 2s. The lost fault type fixes the value of data to a given value spanning 50 frames. After 50 frames, the data resumes with the correct values. The delay fault type fixes the value of data to a given value for 50 frames but also continuously records the updates to the value after each frame in a list. After 50 frames, the data from this list is retrieved, in order, for the next 50 frames. After a total of 100 frames, the data resumes with the correct values.

The second parameter for translating the failure mode, i.e., the variable, decides the faults that can be represented in CARLA. The variables or faults for which fault injection is currently supported by the ISDS framework are -

a) Sensor Faults: Sensory faults are injected by manipulating the measurements from sensors (camera, GPS, IMU, etc.). Sensory faults represent scenarios where faulty sensor data are obtained from sensor errors, the environment, noise, etc. Based on the sensor type, these faults alter the host vehicle velocity (faulty IMU sensor), host vehicle location (faulty GPS sensor), target vehicle velocity (faulty radar/camera/lidar sensor), and target vehicle location (faulty radar/camera/lidar sensor).

b) Actuator Faults: Actuator faults are injected by manipulating the output data sent by the controller to the system actuators (throttle, steering, and brake). Actuator faults represent real-world scenarios where faulty actuators signals lead to incorrect control commands of a system, such as the unintended acceleration of a vehicle. Based on the actuator type, these faults alter the throttle percentage (faulty accelerator), steering angle (faulty steering system), and brake percentage (faulty brake system).

c) Timing Faults: Timing faults are injected by manipulating the communication of information between systems (sensor to controller or vice versa). Such faults can be injected for sensor, actuator, and processing blocks. Timing faults are only applicable with the delay and loss fault types. Timing faults represent real-world scenarios where the communication of data between modules is affected due to hardware or software causes.

In summary, the FIC module is responsible for extracting system design and safety data from the SysML model and configuring CARLA to run fault injection. The module is responsible for configuring the CARLA world and client with the failure operational scenario, the safety goals, component safety requirements, and failure modes of the system under development. Once the FIC module configures the CARLA client module with data from the SysML model, the focus shifts to running the fault injection experiment in CARLA.

4.3.3.2 Fault Injection Campaign Manager (FICM)

FICM is responsible for executing the fault injection experiment in CARLA. Once the FIC module has configured the CARLA client according to the information from the SysML model, FICM begins executing the fault injection. In general, fault injection involves analyzing the response of the system after artificially or forcefully inserting

faults or errors [110]. It consists of two types of runs (i.e., test cases): the golden-run and the faulty-run. The golden run involves observing the behavior of the system in its intended operational scenario in the absence of faults [92, 110]. The golden run acts as a reproducible reference or baseline for system behavior in the specific operational scenario and captures information about the system during execution. Then, the faulty run is executed for the same operational scenario. The faulty run involves inserting faults into the system, observing its behavior, and capturing the same information about the system during execution. For the given operational scenario, the data from the golden-run and faulty-run are compared to determine the effects of the faults.

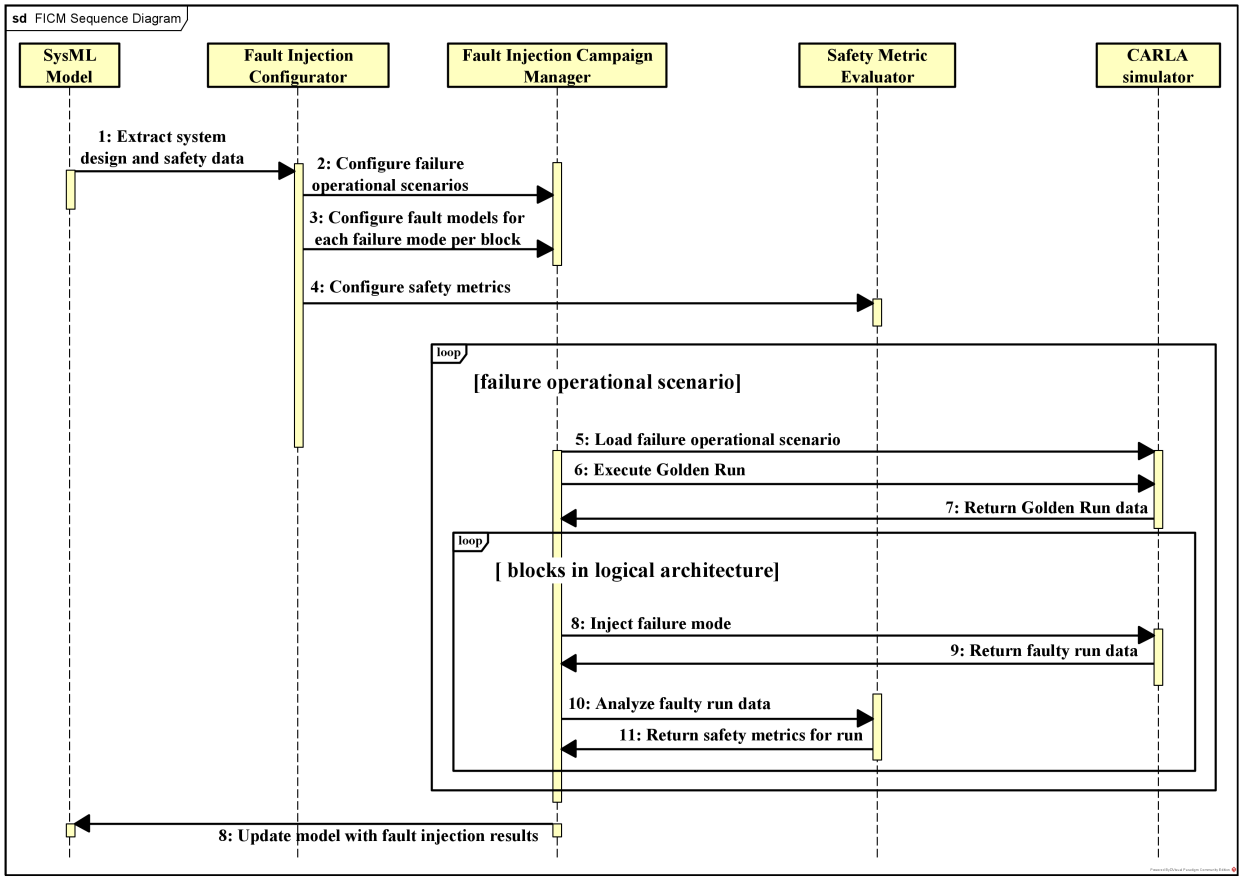


Figure 26: Sequence diagram elaborating the steps involved in performing the safety verification using the fault injection simulation.

FICM uses a similar approach to execute the fault injection in CARLA, as shown in Figure 26. The FIC module forwards the list of failure operational scenario, list of failure modes, list of safety goals and list of component safety requirements to the FICM module. These lists contain objects of the corresponding elements that were instantiated by the FIC module. For each failure operational scenario in the list of operational scenarios, a single golden-run and multiple faulty-runs (one for each failure mode injected) are executed. First, FICM sets up the configuration of the failure operational scenario in CARLA. Next, it executes the golden-run, i.e., the system is executed for the given failure operational scenario without injecting any faults in the system. Data of the golden-run is stored by the module and acts as the baseline for system behavior. The data stored is limited to the variables used in evaluating a safety goal violation, i.e., the variables used in defining the logical expression in the criteria for each safety goal violation. Next, FICM executes the faulty-runs. For the same operational scenario, the FICM module iterates through the list of failure modes, which contains objects of the *failureMode*

class and represent the fault models of the system's failure modes. The faulty run can involve the injection of single failure modes as well as failure mode combinations. Similar to the golden-run, after each faulty-run, system data corresponding to the safety goals are stored by the FICM module. After each faulty-run, the FICM module sends the golden run data and faulty run data to the Safety Metric Evaluator (SME) to compute the safety metric and assess if a safety goal or component safety requirement was violated for that run. The results from the SME are stored in the failure mode object. This process repeats for each failure mode and failure mode combination identified for the system. Once all the failure modes have been injected for the current operational scenario, the next operational scenario is loaded, and the process is repeated. When the FICM module completes the fault injection runs for all the failure operational scenarios defined in the SysML model, the results from the fault injection are updated in the SysML model.

4.3.3.3 Safety Metric Evaluator (SME)

SME is responsible for receiving the data corresponding to each faulty run from FICM, comparing the data with that of the golden run and assessing the safety metrics for the system. The safety metrics computed are:

1. Safety goal violations: This is a Boolean value that indicates if the logical expression that represents the violation of the safety goal, as defined in the SysML model, was met. It returns true if the safety goal was violated. All safety goals defined for the system are evaluated at the end of each faulty run by comparing the golden run data with the faulty run data.
2. Component safety requirement violations: This is a Boolean value that indicates if the logical expression that represents the violation of the component safety requirement, as defined in the SysML model, was met. It returns true, along with the simulation time, if the component safety requirement was violated. All the component safety requirements are continuously evaluated during each faulty run.

4.3.4 Feedback from safety verification to SysML model

Once the fault injection simulation is complete, the SysML model is updated by the FICM module with the results from the verification. The results from the safety verification are updated in the XML file of the SysML model. This XML file can be imported into the SysML tool to be viewed by the engineer. The updates made in the XML file (i.e., the SysML model) are -

1. Failure Modes: Failure modes are defined in the SysML model with the «failureMode» stereotype and contain placeholders for the results from the safety verification i.e., the "Single FM simulation data" and "Combination FM simulation data" tagged values (refer to section 4.1.2 for details). After each faulty run, the results of the assessment by the SME are stored in the object of the *failureMode* class. If a single failure mode was injected, then the safety goals violated and the failure operational scenario during which the violation occurred are stored. If multiple failure modes were injected, then the names of each injected failure mode, the safety goals violated, and the failure operational scenario during which the violation occurred are stored. This data is updated in the XML file of the SysML model.
2. Component Safety Requirements: Component safety requirements are defined in the SysML model with the «componentSafetyRequirement» stereotype and contain a placeholder for storing the results from the safety verification, i.e., the "Violation simulation data". Feedback of safety verification results related to component safety requirements is similar to the process followed for failure modes- after each faulty run, the name of the failure mode that caused the violation, the failure operational scenario during which the

violation occurred, and the simulation time at which the violation occurred is stored in the corresponding object of the *componentSafetyRequirement* class. This data is updated in the XML file of the SysML model.

The feedback of results from safety verification to the SysML model marks the end of the model-based safety verification approach. The updated SysML model is used to generate a new FMEA table and fault trees of the system. The simulation data column in the FMEA is updated with the results from the safety verification. The fault tree generation algorithm is re-run to generate fault trees based on the safety goal violations found during the fault injection simulation, i.e., safety verification. Engineers can compare the original fault trees with the updated fault trees and review design decisions and failure mode mitigation strategies. This step marks the end of the model-based safety verification approach. However, as highlighted in the literature review, safety assessment is an iterative process. After each safety assessment, system designs are updated, and the safety assessment is repeated. The feedback mechanism of the ISDS framework ensures that consistency is maintained with each iteration and enables safety assessment with each incremental design change per iteration.

In summary, this chapter introduced the ISDS framework and provided a detailed description of the app aches used to perform the model-based safety assessment. Section 4.1 provided an overview of the ISDS framework. Section 4.1.1 introduced the CARLA simulator, its architecture, and capabilities. Section 4.1.2 provided a detailed description of the safety profile developed for the framework. Section 4.2 described the model-based safety analysis approach adopted during the system definition phase of the framework. Finally, section 4.3 described the model-based safety verification approach adopted in the system verification phase of the framework. The next chapter will focus on the solution evaluation phase of the research methodology by applying the ISDS framework on a case study.

Chapter 5

Results

This chapter applies the ISDS framework, described in chapter 4, to a case study to evaluate its effectiveness as a model-based safety assessment framework. The case study involves the development and safety assessment of a Forward Collision Warning (FCW) system in an autonomous vehicle. Data from this case study is used to answer the research questions formulated in chapter 3, section 3.4. To recap, the first research question investigates if the ISDS framework can eliminate tasks that introduce inconsistencies in the safety assessment process compared to the current state-of-the-art. The discussion in chapter 2 (i.e., literature review) highlighted that the current state-of-the-art uses a manual method to update the SysML model with changes made to the safety artifact (i.e., from safety analysis) as well as a manual method to update the SysML model with results from safety verification, which are sources of inconsistency in the safety assessment process and can lead to an incorrect safety assessment. This case study will highlight how the feedback mechanism in the ISDS framework can automatically update the SysML model with changes made in the safety artifact as well as with the updates from safety verification, thereby eliminating the need for safety engineers to perform these tasks. The second research question investigates if the ISDS framework can assess the safety of the system for a wider range of behaviors than the current safety assessment method for FCW systems. Section 5.1 in this chapter will highlight the current method and metric used to assess the safety of FCW systems. The case study will highlight how the ISDS framework can identify additional unsafe behaviors (i.e., behaviors that cause the system to fail safety assessment) in a system that is considered safe (i.e.,the system passes safety assessment) based on current safety assessment methods and metrics. The answers to these research questions will demonstrate the effectiveness of the ISDS framework as a model-based safety assessment framework.

The FCW system case study was selected for several reasons: a) since the ISDS framework is closely coupled to the CARLA simulator, the possible case studies were limited to automotive systems or a safety-critical sub-system within an automotive system, and b) FCW algorithms were first developed in the 1990s [111, 112] and are now ubiquitous in automotive systems; US government agencies are even looking to mandate new vehicles to be equipped with such systems [113]. As a result of the widespread adoption, mature methods have been developed by the National Highway Traffic Safety Administration (NHTSA) to assess the safety of FCW systems. The ISDS framework can be compared with these methods to evaluate the effectiveness of its safety assessment. The chapter is organized as follows: Section 5.1 introduces the reader to FCW systems.

Section 5.2 applies the ISDS framework towards the development and safety assessment of an FCW system case study. Finally, Section 5.3 ties the results of the case study to the research questions formulated in this dissertation.

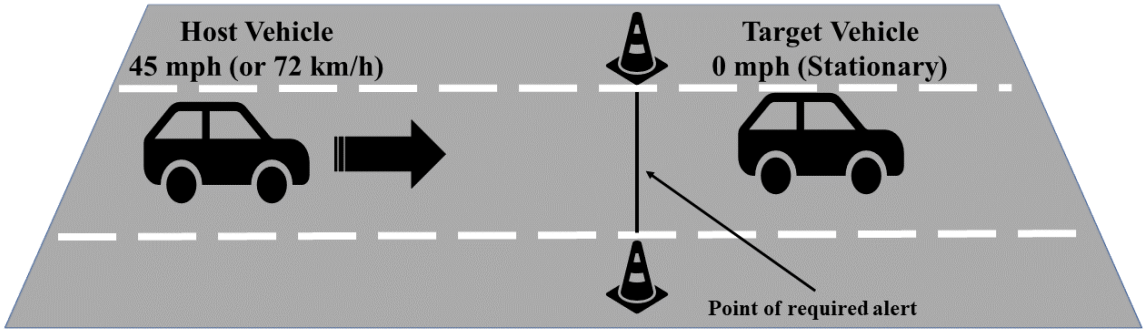
5.1 Introduction to FCW systems

An FCW system identifies the potential for an impending crash situation at the front of a vehicle and either provides an alert to the driver of the vehicle or sends a control signal to activate another vehicular sub-system that can prevent the crash [114]. In 2019, the NHTSA reported that approximately 42% of passenger car related crashes were due to forward collisions with other vehicles [115]. To prevent or reduce the impact of such crashes, driver assist technologies such as FCW systems have been developed. A 2017 study evaluating the effectiveness of forward collision warning systems in reducing front-to-rear crash rates found that FCW systems alone reduce crash involvement rates by up to 27% [116]. Due to its effectiveness, FCW systems are becoming widely adopted by automotive manufacturers; a voluntary commitment by 20 automakers, representing 99% of car manufacturers, will see all new passenger vehicles equipped with FCW systems by September 1, 2022 [117]. The impact of FCW systems on society and businesses was a key factor in selecting the case study.

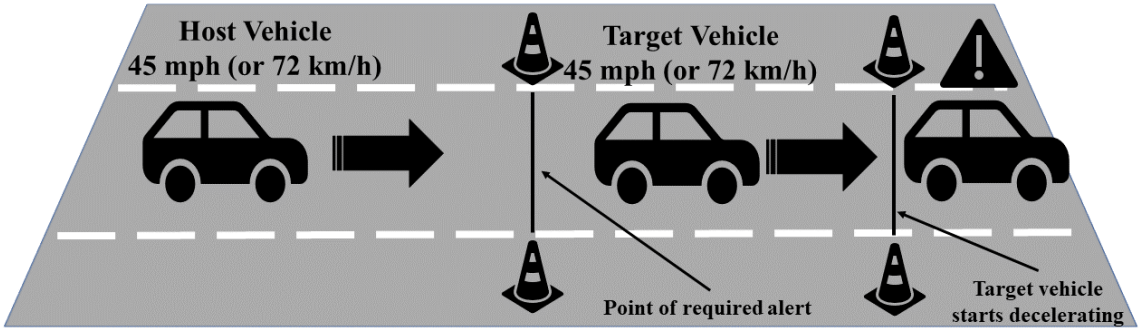
Given the widespread adoption of FCW systems, the NHTSA has developed a series of test procedures, called the Forward Collision Warning Confirmation Test, to evaluate the safety of FCW systems [37]. These tests evaluate the ability of the FCW system to detect and provide an alert for a potential crash under three driving scenarios, as shown in Figure 27. For each scenario, the Time-To-Collision (TTC) metric is used to assess the risk of collision. TTC is defined as the time required for two vehicles to collide if they continue at their current speed and remain on the same path [102]. The test scenarios developed for the Forward Collision Warning Confirmation test are -

1. **Host vehicle encounters a stopped target vehicle on a straight road**: As shown in Figure 27 (a), the host vehicle is travelling at 45 mp/h or 72 km/h and encounters a stationery target vehicle in its path on a straight road. The FCW system of the host vehicle is required to provide an alert when the TTC is $\geq$ 2.1 seconds.
2. **Host vehicle encounters a decelerating target vehicle on a straight road**: As shown in Figure 27 (b), the host vehicle and target vehicle are travelling at 45 mp/h or 72 km/h, 30 meters apart. At a certain point, the target vehicle begins decelerating. For this test, the FCW system of the host vehicle is required to provide an alert when the TTC is $\geq$ 2.4 seconds.
3. **Host vehicle encounters a slower target vehicle on a straight road**: As shown in Figure 27 (c), the host vehicle is travelling at 45 mp/h or 72 km/h and encounters a slower moving target vehicle in its path, travelling at a constant speed of 20 mp/h or 32 km/h. The FCW system of the host vehicle is required to provide an alert when the TTC is $\geq$ 2 seconds.

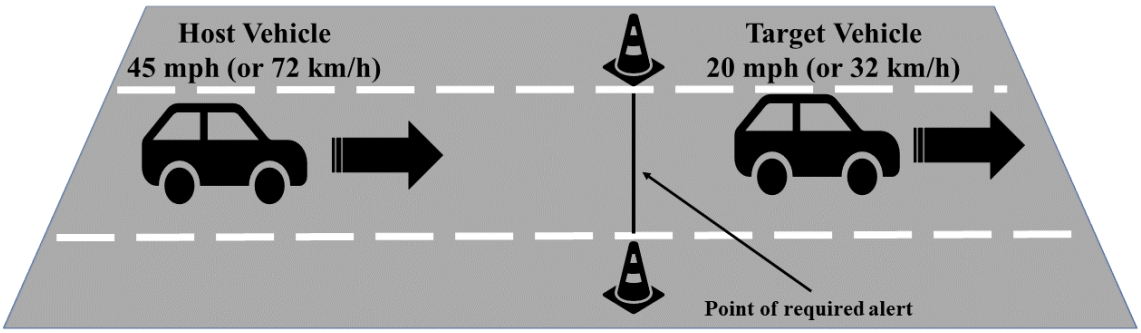
These test procedures have been used by the NHTSA to evaluate the safety of FCW systems since 2013. The existence of mature evaluation methods for FCW systems was another key factor in selecting this case study. Over the past two decades, several algorithms have been developed for forward collision warning, as highlighted in [112]. Since the purpose of this case study is to only assess the safety of an FCW system, the case study will implement the Honda algorithm [118] due to its simplicity, ease of implementation, and extensive use in the field. The Honda algorithm computes the minimum distance required between a host vehicle (equipped with the



(a) Host vehicle encounters a stopped target vehicle on a straight road.



(b) Host vehicle encounters a decelerating target vehicle on a straight road.



(c) Host vehicle encounters a slower target vehicle on a straight road.

Figure 27: Test scenarios used by NHTSA to evaluate the safety of FCW systems.

FCW system) and a target vehicle in its path. If the actual distance between the two vehicles is less than this minimum distance, an alert is triggered.

The equation to compute this distance is:

$$Distance_{warning}(in\ meters) = 2.2 * v_{rel} + 6.2 \tag{5.1}$$

where $Distance_{warning}$ represents the critical distance at which the alert is issued, v_{rel} represents the relative velocity between the two vehicles, and 6.2 is a constant factor that represents the minimum distance that must exist between the two vehicles.

This section introduced the reader to forward collision warning systems, described the test procedures used by the NHTSA to evaluate the safety of such systems, and highlighted the specific FCW algorithm that will be

used in this case study. The next section will apply the ISDS framework towards assessing the safety of an FCW system that uses the Honda algorithm.

5.2 ISDS framework applied to the FCW system case study

In this section, the ISDS framework is applied towards the development and safety assessment of an FCW system that uses the Honda algorithm. The case study will begin with the model-based safety analysis approach (i.e., the left side of the framework) and define the SysML model with system design data, safety data, and simulation-specific properties. The model-based safety analysis approach is complete once the safety artifacts are automatically generated (FMEA tables and fault trees), the changes made to the FMEA are updated in the SysML model, and component safety requirements are defined, if required. Next, the case study will highlight the application of the model-based verification approach (i.e., right side of the framework) and describe the results of the safety verification, illustrate the feedback of results to the SysML model, and highlight the updates made to the safety artifacts.

5.2.1 Model-based safety analysis

As highlighted in Section 4.2 and Figure 8, the model-based safety analysis approach is comprised of nine steps. This sections highlights the SysML model elements developed and/or results after each step of the model-based safety analysis approach for the FCW system case study. Section 5.2.1.1-5.2.1.9 shows the ability of the feedback mechanism in the ISDS framework to automatically update the SysML model with changes made to the safety artifact and highlight how the framework eliminates a task that introduces inconsistencies. Data from this section will contribute in answering research question 1. The remainder of this section will guide the reader through the application of each step in the model-based safety analysis approach to the FCW system case study.

5.2.1.1 Develop CONOPS and create failure operational scenarios

This step develops the CONOPS to identify the set of capabilities that the system must have. In this case study, the capabilities are inspired from [119]. These capabilities represent high-level needs such as the ability to detect road lanes, detect obstacles in the lane, assess possible threats etc. The capabilities feed into the system requirements and subsequently the functional architecture. Next, the failure operational scenarios are created using the «scenarioConfiguration» stereotype to configure the test scenario (which were highlighted in Figure 27) in CARLA. Using Table 1 and Figure 27 as a reference, three failure operational scenarios are defined, one for each test scenario. The failure operational scenarios are shown in Figure 28.

5.2.1.2 Identify system requirements

The system requirements translate the system capabilities and stakeholder needs into requirements. Since the case study focuses on the safety assessment of the FCW system, the system requirements do not directly influence the safety assessment. However, since these requirements are used to realize the system architecture, they are also a part of the ISDS framework. Figure 29 highlights the requirements for the FCW system.

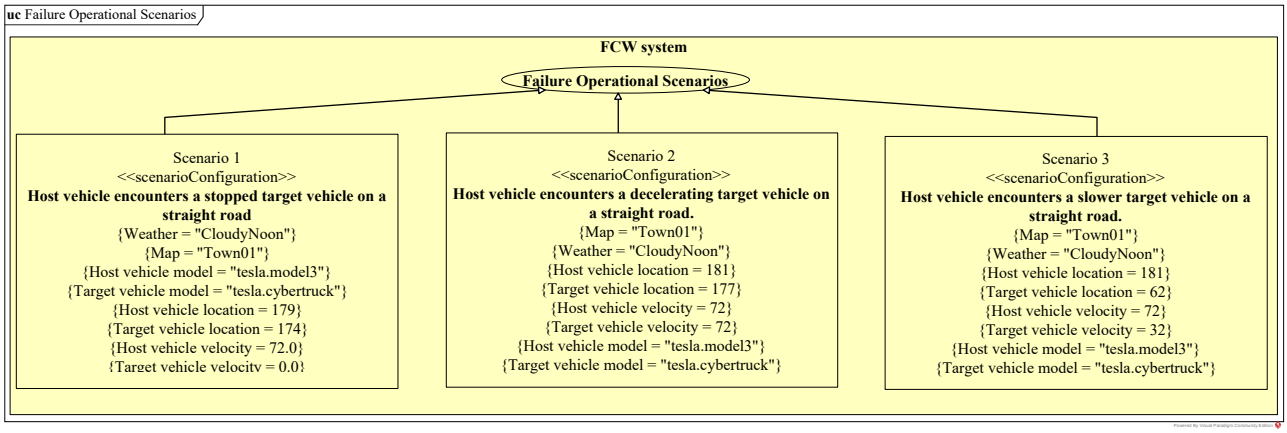


Figure 28: Failure operational scenario defined using the «scenarioConfiguration» stereotype for the FCW system case study. Tagged values for each failure operational scenario are defined such that it configures the CARLA simulator to recreate the test scenarios defined in the NHTSA FCW Confirmation Test.

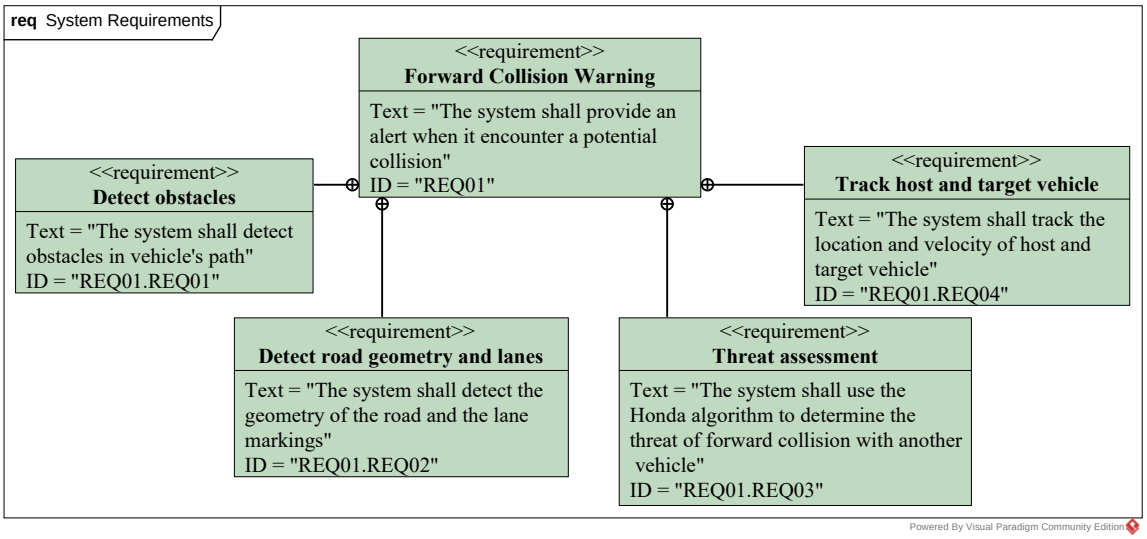


Figure 29: System requirements for the FCW system case study. As eliciting comprehensive system requirements is not within the scope of this dissertation, the requirements identified in this figure represent a sufficient subset for this case study.

5.2.1.3 Perform system hazard and risk assessment (system HARA)

The system HARA consists of three steps: a) identify possible system-level hazards that the system might encounter, b) evaluate the risk that each hazard poses to the system and assign a safety integrity level to each hazard, and c) define safety goals to mitigate the hazards. The following section will describe how each step is performed in the FCW system case study.

a) Identify system level hazards: The system-level hazard analysis is performed using the HAZOP technique. Guide words like early, late, loss, and not requested are used to identify four system-level hazards:

   1) H1: late engagement of FCW system.
   2) H2: unexpected loss of FCW system.
   3) H3: unexpected engagement of FCW system.

b) Evaluate risk and assign safety integrity level: The risk associated with each hazard is assigned based on the risk assessment framework of the ISO 26262 standard. The framework evaluates the potential crash

Assessment Variable	Description
Hazard	H1: late engagement of FCW system
Failure operational scenario	Scenario 1: Host vehicle encounters a stopped target vehicle on a straight road
Crash Situation	Host vehicle is traveling at 45 mph or 72 km/h on a straight road in an urban zone with no pedestrians present on a clear day, encounters a stationery target vehicle in its path. The vehicle's FCW system engages later than it should but avoids a crash.
Severity	S2: Light and moderate injuries
Exposure	E4: High Probability
Controllability	C2: Normally Controllable
Safety Integrity Level	B

Table 8: Risk assessment for system-level hazard H1: late engagement of FCW system for failure operational scenario 1.

situation that arises when the system-level hazards occur during the three test scenarios identified in Figure 27. Table 8 shows the risk assessment process when hazard H1: late engagement of FCW system hazard occurs during failure operational scenario 1. Based on the crash situation description, the class for severity, exposure, and controllability are selected using the classification Table 2, 3, 4 respectively. Given that the vehicle is travelling at only 45 mph and avoids the crash since it engages earlier than it should, the severity of the hazard is classified as S2, where the injuries are minor. The exposure is classified as E4 since it is highly likely for vehicles in urban environments at 45 mph to encounter such a situation. Finally, the controllability is classified as C2 since the speed of the vehicle and late engagement of the FCW system would not cause the vehicle behavior to be uncontrollable. Using Table 5, the safety integrity level is classified as B. Table 9, 10 follow a similar process to assess the risk for the late engagement of FCW system during failure operational scenario 2 and 3 respectively. The risk assessments for hazards H2 and H3 are shown in Appendix A.

Assessment Variable	Description
Hazard	H1: late engagement of FCW system
Failure operational scenario	Scenario 2: Host vehicle encounters a decelerating target vehicle on a straight road.
Crash Situation	Host vehicle is traveling at 45 mph or 72 km/h on a straight road in an urban zone with no pedestrians present on a clear day and encounters a target vehicle travelling at 45 mph or 72 km/h which begins to decelerate. The vehicle's FCW system engages later than it should but avoids a crash .
Severity	S2: Light and moderate injuries
Exposure	E4: High Probability
Controllability	C2: Normally Controllable
Safety Integrity Level	B

Table 9: Risk assessment for system-level hazard H1: late engagement of FCW system for failure operational scenario 2.

Assessment Variable	Description
Hazard	H1: late engagement of FCW system
Failure operational scenario	Scenario 3: Host vehicle encounters a slower target vehicle on a straight road.
Crash Situation	Host vehicle is traveling at 45 mph or 72 km/h on a straight road in an urban zone with no pedestrians present on a clear day and encounters a target vehicle travelling at 20 mph or 32 km/h. The vehicle's FCW system engages later than it should but avoids a crash.
Severity	S2: Light and moderate injuries
Exposure	E4: High Probability
Controllability	C2: Normally Controllable
Safety Integrity Level	B

Table 10: Risk assessment for system-level hazard H1: late engagement of FCW system for failure operational scenario 3.

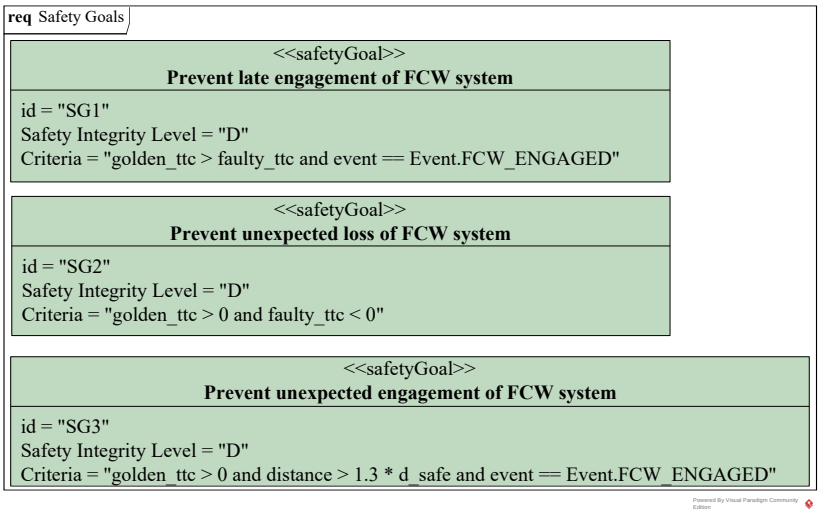


Figure 30: Safety goals of the FCW system defined using the «safetyGoal» stereotype in a requirements diagram. The safety goals are developed using the HAZOP technique and is the last step of system HARA.

c) Define safety goals: The safety goals represent system-level safety requirements that are defined to mitigate the system-level hazards identified in the previous step. In the SysML model, they are defined using the «safetyGoal» stereotype. Figure 30 illustrates the safety goals for the FCW system. For each safety goal, the safety integrity level is assigned based on the results from the risk assessment as shown in Table 8, 9, 10, and Appendix A. The criteria tagged value in the «safetyGoal» stereotype is used to formalize the requirement and configure the safety goal in CARLA. For example, the criteria for SG1: Prevent late engagement of FCW system is *golden_ttc* $\geq$ *faulty_ttc AND event == Event.FCW_ENGAGED*. This expression checks if the TTC at which the FCW system engaged during the faulty run is lesser than the TTC at which the FCW system engaged during the golden run (for details on golden and faulty runs, please refer back to Section 4.3.3.2). The criteria for SG2 checks if the FCW system failed to engage during the entire test scenario and did not record a value. Finally, the criteria for SG3 checks if the FCW system engaged at a distance greater than 30% of d_safe, which is the distance at which the FCW system was supposed to engage.

5.2.1.4 Develop functional architecture

The functional architecture is developed by translating the system requirements into functions and capturing the interactions between the functions, as shown in in Figure 31. Perception data, from camera systems or lidar systems, is used to detect lanes by identifying the lane marking on the roadway. Radar data is used to detect obstacles in front of the vehicle, as well as the obstacle's location and velocity. The perception data and radar are combined to determine if the obstacle, now called the target vehicle, is in the host vehicle's pathway. On-board sensors are used to determine the host vehicle's data such as velocity and location. The host and target vehicle data are tracked relative to each other and forwarded to the assess the threat of collision. Based on the assessment, which will be performed using the Honda algorithm, the vehicle actuation is determined i.e., the level of throttle, steering and brake. The levels are translated into control signals and forwarded to the respective actuation sub-systems, external to the FCW system.

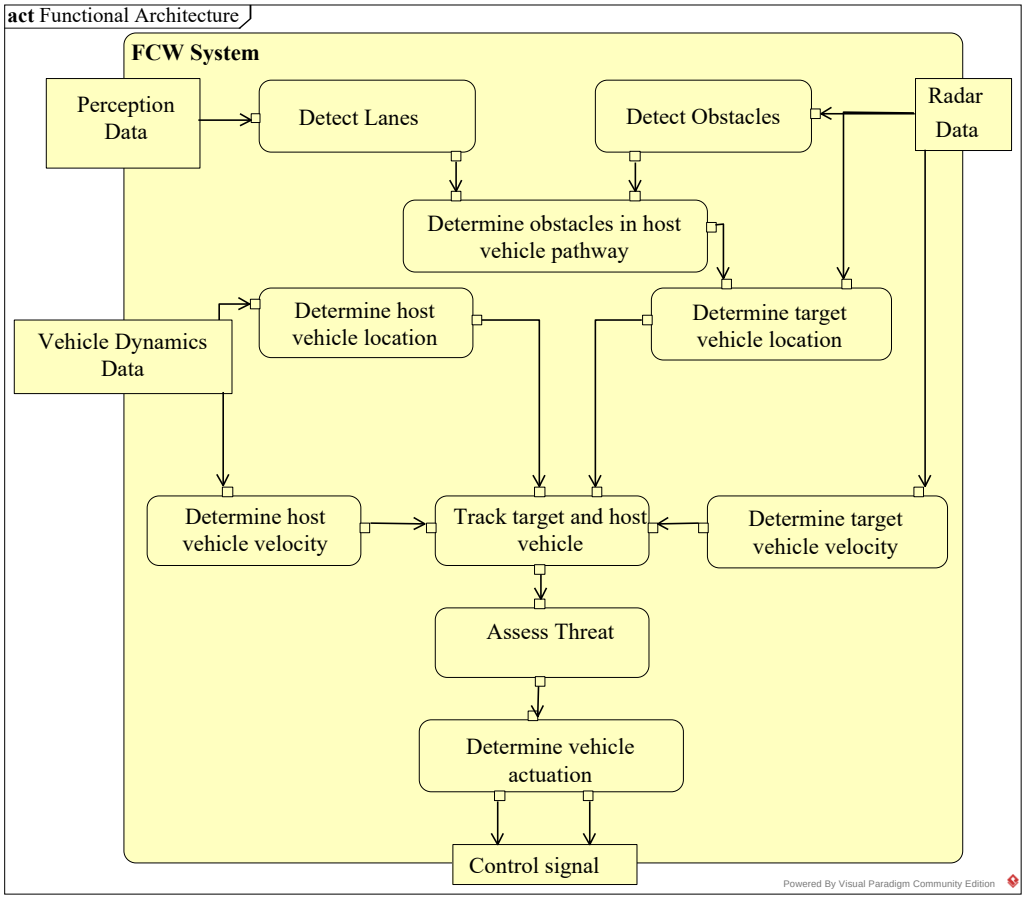


Figure 31: Functional architecture for the FCW system modelled using an activity diagram.

5.2.1.5 Develop logical architecture

The logical architecture identifies the logical blocks of the system (i.e., sub-systems), allocates the system functions to the blocks, and identifies the interconnections or data flows between each block. The FCW system's logical architecture contains seven sub-systems with functions from the functional architecture allocated to it, as shown in Figure 32. The lane detection sensors and obstacle detection sensors are responsible for detecting the lane marking on the roadway and obstacles in front on the host vehicle, respectively. The object detecting

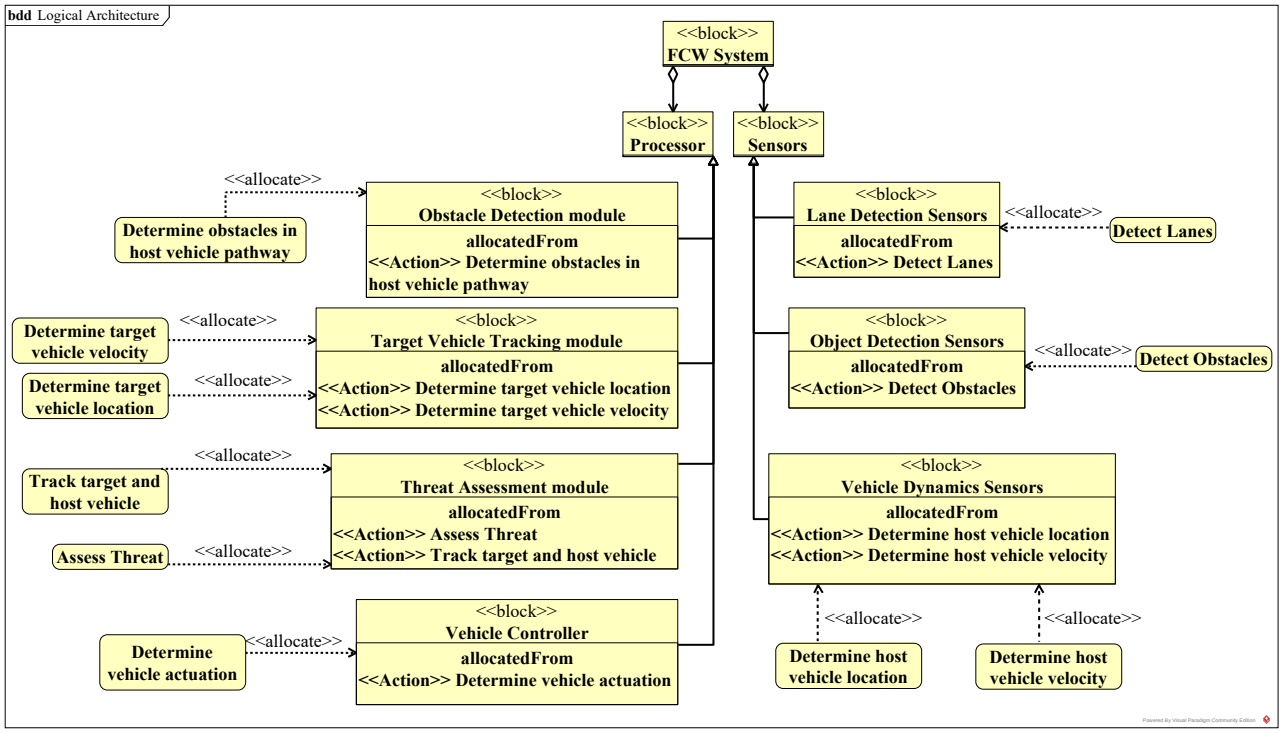


Figure 32: Logical architecture for the FCW system modelled using a Block Definition Diagram (BDD).

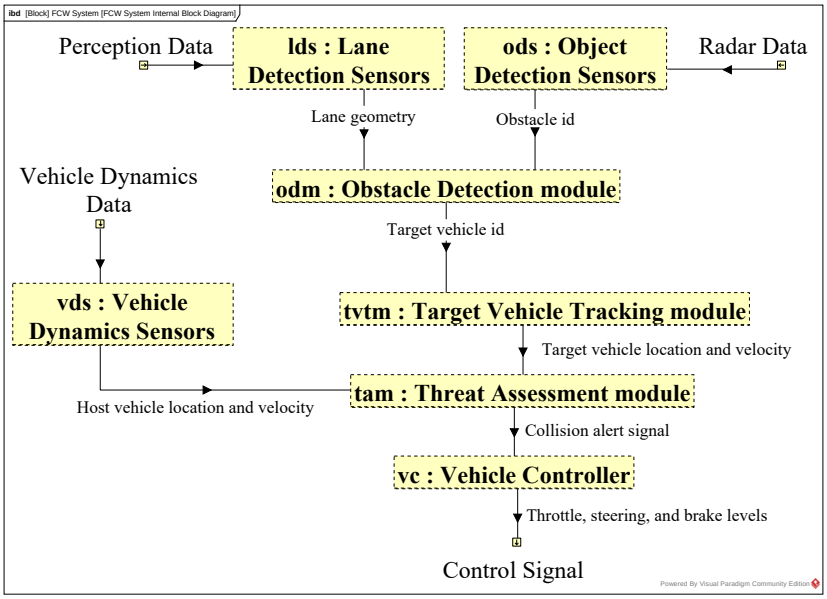


Figure 33: Logical architecture for the FCW system modelled using an Internal Block Diagram (IBD).

module filters the obstacles to those within the lane of the host vehicle. The target vehicle tracking module computes the target vehicle's location and velocity. The vehicle dynamics sensor computes the host vehicle's location and velocity. The threat assessment module tracks the relative distance and velocity between the host and target vehicle. The module also uses the Honda algorithm to determine the threat of a potential collision. Finally, the vehicle controller determines the level for throttle, steering and brake signals required based on the threat assessment and forwards the control signals to the external actuation sub-systems. Figure 33 represents the IBD of the logical architecture and illustrates the interconnections and flow of data between the sub-systems.

5.2.1.6 Perform sub-system hazard and risk assessment (sub-system HARA)

The sub-system hazard analysis is performed on the functions allocated to the different sub-systems in the logical architecture using the HAZOP technique. For each function guide words like too high, too low, lost, delay, intermittent, and inverse are used to brainstorm functional hazards. These hazards represent the function's failure modes and are captured using the «failureMode» stereotype. Figure 34 shows the failure modes for the *Determine host vehicle location* function of the *Vehicle Dynamics Sensor* sub-system. For each failure mode, the cause, effect, and user-defined safety goal violations have been defined. The mitigation strategy tagged value is undefined and will be updated in the FMEA spreadsheet generated in the next step. Finally, the variable and fault type have been defined to characterize the failure mode in CARLA. A subset of the failure modes combine with the failure modes of the *Determine target vehicle location* function of the "Target Vehicle Tracking module" sub-system and are defined using the "Combines with" dependency relationship.

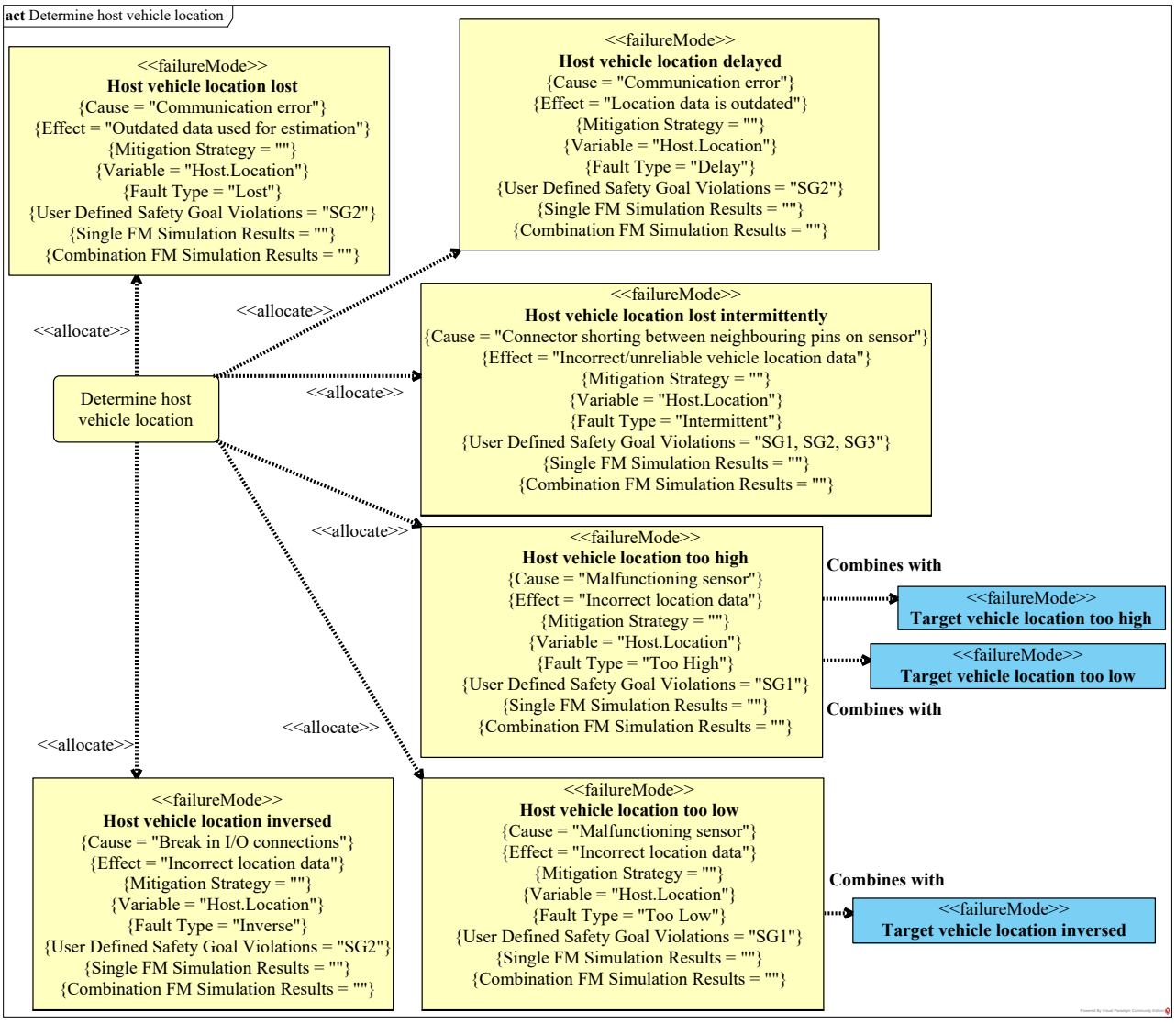


Figure 34: Failure modes for the *Determine host vehicle location* function of the Vehicle Dynamics Sensors block modelled defined using the «failureMode» stereotype in an activity diagram. Failure modes of the Target Vehicle Tracking module's *Determine target vehicle location* function are shown in blue. Failure mode combinations are identified using the "Combines with" dependency.

Failure modes for the remaining functions of the FCW system are shown in Appendix B. It is important to note that as highlighted in Section 4.3.3.1, the ISDS framework does not currently support camera/lidar/radar related faults. The Lane Detection Sensors, Object Detection Sensors, and Obstacle Detection module relies on perception data to perform its functions. Consequently, sub-system HARA is not performed on these three sub-systems, i.e., no failure modes are identified for these sub-systems since they cannot be characterized in CARLA.

5.2.1.7 Generate safety artifacts

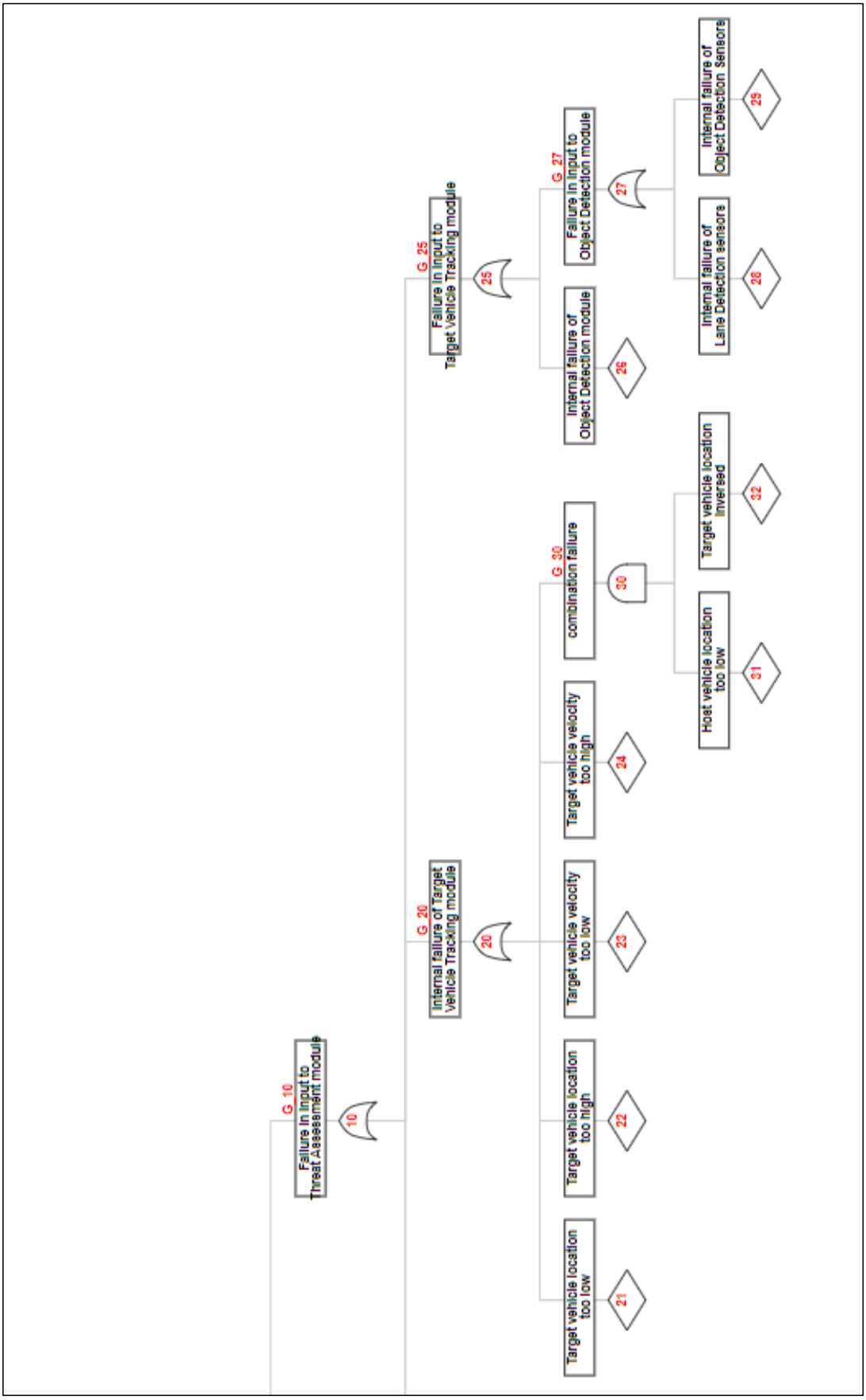
To automatically generate the FMEA table and fault trees, the SysML model is exported as an XML file. The framework's algorithm parses the file to generate the safety artifacts, as described in Section 4.2.7. The first safety artifact generated is the FMEA table spreadsheet. The algorithm extracts failure mode data and creates a spreadsheet of the system FMEA. If the failure mode contains tagged values that have not been defined, the corresponding cell in the spreadsheet contains the "-" symbol, which denotes that an engineer has to enter this information in the spreadsheet. Figure 35 shows a small section of the FMEA table, which represents the FMEA for the *Determine host vehicle location* function (refer to Figure 34 for description of function's failure modes in SysML). The simulation data column is left empty and is only updated once the results from safety verification are available. The system FMEA for all functions of the FCW system is shown in Appendix C.

Block	Function	Failure Mode	Cause	Effect	Safety Goal Violation	Risk	Mitigation Strategy	Simulation Data
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location lost	Communication error	Outdated data used for estimation	SG2	D	-	
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location delayed	Communication error	Location data is outdated	SG2	D	-	
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location lost intermittently	Connector shorting between neighboring pins on sensor	Incorrect/unreliable vehicle location data	SG1, SG2, SG3	D	-	
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location too high	Malfunctioning sensor	Incorrect location data	SG1	D	-	
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location too low	Malfunctioning sensor	Incorrect location data	SG1	D	-	
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location inversed	Break in I/O connections	Incorrect location data	SG2	D	-	

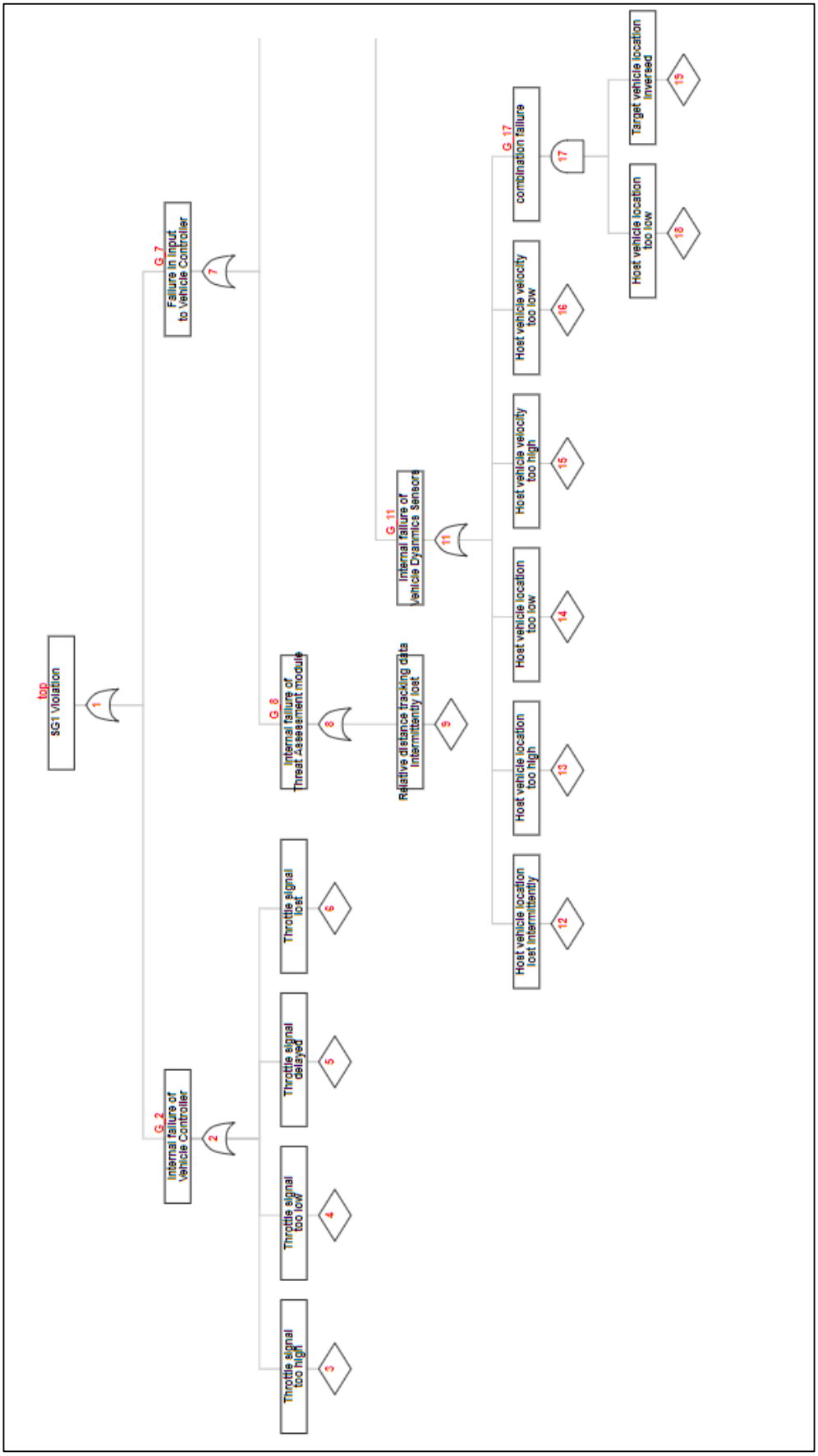
Figure 35: FMEA table for the *Determine host vehicle location* function exported as an image. Data in any cell can be updated by a safety engineer, if necessary. The "-" symbol signifies that the field was undefined in the SysML model and requires an engineer to update the spreadsheet. In this instance, the mitigation strategy is undefined in the SysML model.

The next safety artifacts automatically generated by the ISDS framework are the system fault trees, one for each safety goal. Figure 36 illustrates the fault tree for violation of the safety goal *SG1: Prevent late engagement of FCW system*. The fault tree follows the structure shown in Figure 19. Based on the IBD of the FCW system's logical architecture, the fault tree traverses from output to input. Each intermediate level is either

an internal failure of the block or a failure in the input to the block, which is caused due to an internal failure of its neighboring connected block. Internal failure of each block is represented using only those failure modes of the block that violate SG1. Since the fault tree for SG1 is large, the failure mode combinations identified during sub-system HARA are limited to the combinations shown in Figure 34. The *Determine host vehicle location* function's failure modes are shown to combine with the failure mode of the *Determine target vehicle location* function. The "Host vehicle location too high" failure combines with the "target vehicle location too high" and the "Target vehicle location too low" failure modes. Since all three failures individually violate SG1, they are not shown to combine using an AND gate in the fault tree. Instead, each failure mode is only present within the internal failure of their respective blocks. The "Host vehicle location too low" failure mode combines with the "Target vehicle location inversed" failure mode. While the "Host vehicle location too low" failure mode does violate SG1, the "Target vehicle location inversed" failure mode only violates SG2. In this instance, the failure mode combination is shown using the AND gate in the fault tree, as illustrated in Figure 36. The fault trees for SG2 and SG3 are shown in Appendix D.



(a) Right side of the fault tree for violation of SG1



(b) Left side of the fault tree for violation of SG1

Figure 36: Fault tree for violation of safety goal SG1: Prevent late engagement of FCW system

5.2.1.8 Feedback from FMEA to SysML

The feedback mechanism presented in this section is one of the key contributions of this research as it eliminates the need for safety engineers to manually update the SysML model with any changes made to the FMEA table. Once the SysML model is updated, the fault trees are re-generated to be consistent with the latest changes. Data from this section contributes in answering research question 1.

Once the FMEA table and fault trees are generated, the safety engineer further analyzes the safety artifacts and makes changes as required. For the FMEA of the Vehicle Dynamics Sensor block, shown in Figure 35, the safety engineer uses the data currently available from the FMEA and fault trees to identify a mitigation strategy and update the FMEA. Additionally, the remaining FMEA fields, i.e., cause, effect, and safety goal violation, can also be updated. For example, in the FMEA for the Vehicle Dynamics Sensor block, the engineer identifies "Redundant location sensor" as the mitigation strategy and updates the mitigation strategy field. To realize this update in the system architecture, a redundant location sensor must be added in parallel to the primary location sensor. This sensor acts as a backup to the primary sensor and becomes activated if the primary sensor fails. Another update made to the FMEA is that the safety goal violation for the "Host vehicle location lost intermittently" failure mode is modified to no longer include SG1. The updated FMEA spreadsheet is shown in Figure 37.

Block	Function	Failure Mode	Cause	Effect	Safety Goal Violation	Risk	Mitigation Strategy	Simulation Data
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location lost	Communication error	Outdated data used for estimation	SG2	D	Redundant location sensor	-
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location delayed	Communication error	Location data is outdated	SG2	D	Redundant location sensor	-
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location lost intermittently	Connector shorting between neighboring pins on sensor	Incorrect/unreliable vehicle location data	SG2, SG3	D	Redundant location sensor	-
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location too high	Malfunctioning sensor	Incorrect location data	SG1	D	Redundant location sensor	-
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location too low	Malfunctioning sensor	Incorrect location data	SG1	D	Redundant location sensor	-
Vehicle Dyanmics Sensor	Determine host vehicle location	Host vehicle location inversed	Break in I/O connections	Incorrect location data	SG2	D	Redundant location sensor	-

Figure 37: FMEA table containing updates made by the engineer to the mitigation strategies for each failure mode of the *Determine host vehicle location* function as well as the modification to the safety goal violation of the "Host vehicle location lost intermittently" failure mode. Cells highlighted in red represent changes made to the FMEA compared to Figure 35. The table is exported as an image for better readability.

After the engineer completes the modification to the FMEA spreadsheet, the framework updates the SysML model with the changes made to the spreadsheet. The updates made to the mitigation strategy and safety goal violation of the "Determine host vehicle location" function's failure modes are illustrated in Figure 38. The mitigation strategy in each failure mode has been updated with the data from the FMEA spreadsheet, i.e., Redundant location sensor. Additionally, the User Defined Safety Goal Violations tagged value of the "Host vehicle location lost intermittently" failure mode is updated to only include SG2 and SG3 (as highlighted by the red circle in the updated FMEA, Figure 37).

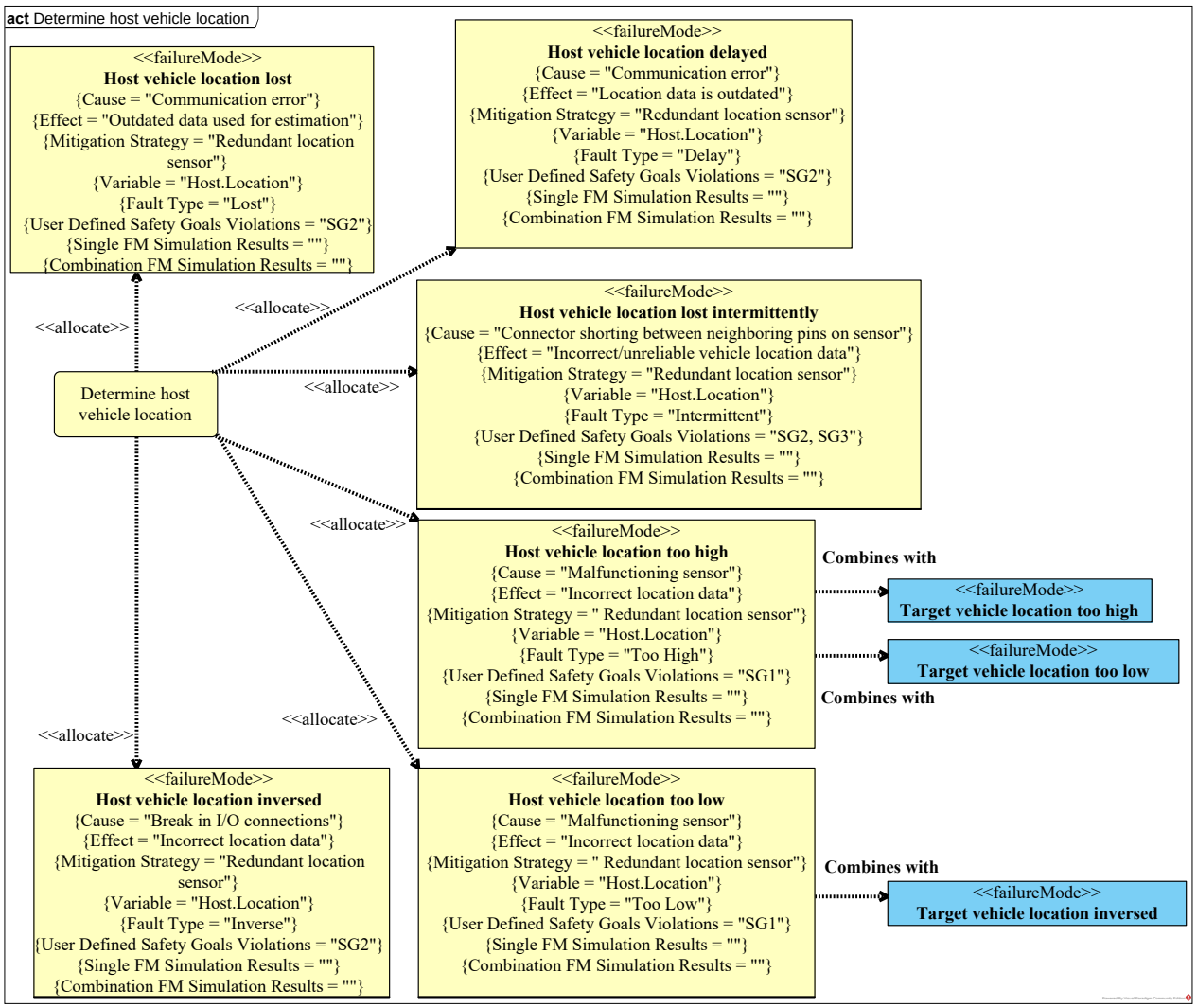


Figure 38: Updated SysML model showing changes made to the failure modes of the *Determine host vehicle location* function. The mitigation strategy for each failure mode has been updated with the data from the FMEA. The User Defined Safety Goal Violations for the "Host vehicle location lost intermittently" has been modified to not include SG1.

At this stage, as highlighted in Section 4.2.8, the functional and logical architecture of the FCW system are modified by the system design and safety engineers to realize the mitigation strategy. This step is not a part of the feedback mechanism; the engineers must manually modify the functional and logical architecture in the SysML model to realize the mitigation strategy and is indicative of the iterative nature of the system design and safety life cycle. Appendix E, Figure 67, 68, 69 illustrate the functional architecture, logical architecture BDD, and logical architecture IBD that are updated with the redundant sensor data. Appendix E, Figure 70 shows the failure modes for the redundant location sensor. It is important to note that the failure mode's safety

goal violation is undefined. Since the sensor is a mitigation strategy, it does not directly violate the safety goal. This feature allows the framework's algorithm to determine that the block is a mitigation strategy and places the block's failure modes in the correct location in the fault tree. Based on the updates to the SysML model, new fault trees are generated. Figure 40 shows the internal failure of redundant location sensor event added to the fault tree for SG1 as well the removal of the "Host vehicle location lost intermittently" failure mode from the fault tree. In Figure 40, the failure modes of the redundant location sensor are contained within the "Internal failure of redundant location sensor" event. This event combines with any failure mode of the *Determine host vehicle location function* with the AND logic gate since both the primary location sensor and the mitigation strategy must fail for the safety goal to be violated, as highlighted in Figure 40.

5.2.1.9 Derive component safety requirements

The model-based safety analysis approach started by identifying system capabilities, failure operational scenarios and safety goals, followed by developing the functional and logical architecture, automatically generating the system FMEA and fault trees, and updating the SysML model with changes made to the FMEA via the feedback mechanism. The last step in the model-based safety analysis approach of the framework is to derive component safety requirements, if necessary. For ease of demonstration, a single component safety requirement is defined in this case study. The *Determine vehicle actuation* function of the Vehicle Controller block is responsible to computing the level of throttle, brake and steering signal required based on the input from other sub-systems. The failure modes allocated to the function include "Steering signal too high" and "Steering signal too low", "Steering signal lost", "Steering signal delayed", "Steering signal lost intermittently", and "Steering signal inversed", as shown in Appendix B, Figure 57. According to [120], a vehicle yaw rate must not exceed 4 degrees per second at a speed close to 50km/hr, called the yaw authority limit. To mitigate the effects of the failure modes related to the steering levels of the Vehicle Controller block, a component safety requirement is defined to prevent the vehicle from exceeding the yaw authority limit. Figure 39 illustrates the component safety requirement. The component safety requirement is allocated to the "Steering signal too high" and the "Steering signal too low" failure modes. The component safety requirement's criteria formalizes the requirement in the CARLA client.

This step completes the model-based safety analysis approach (i.e., the system definition phase) of the ISDS framework and the SysML model is now enriched with the required system design data, safety data, and simulation-specific properties to perform safety verification. The next section will highlight the results from the model-based safety verification.

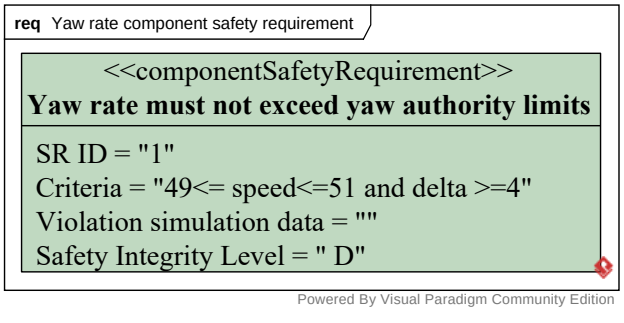
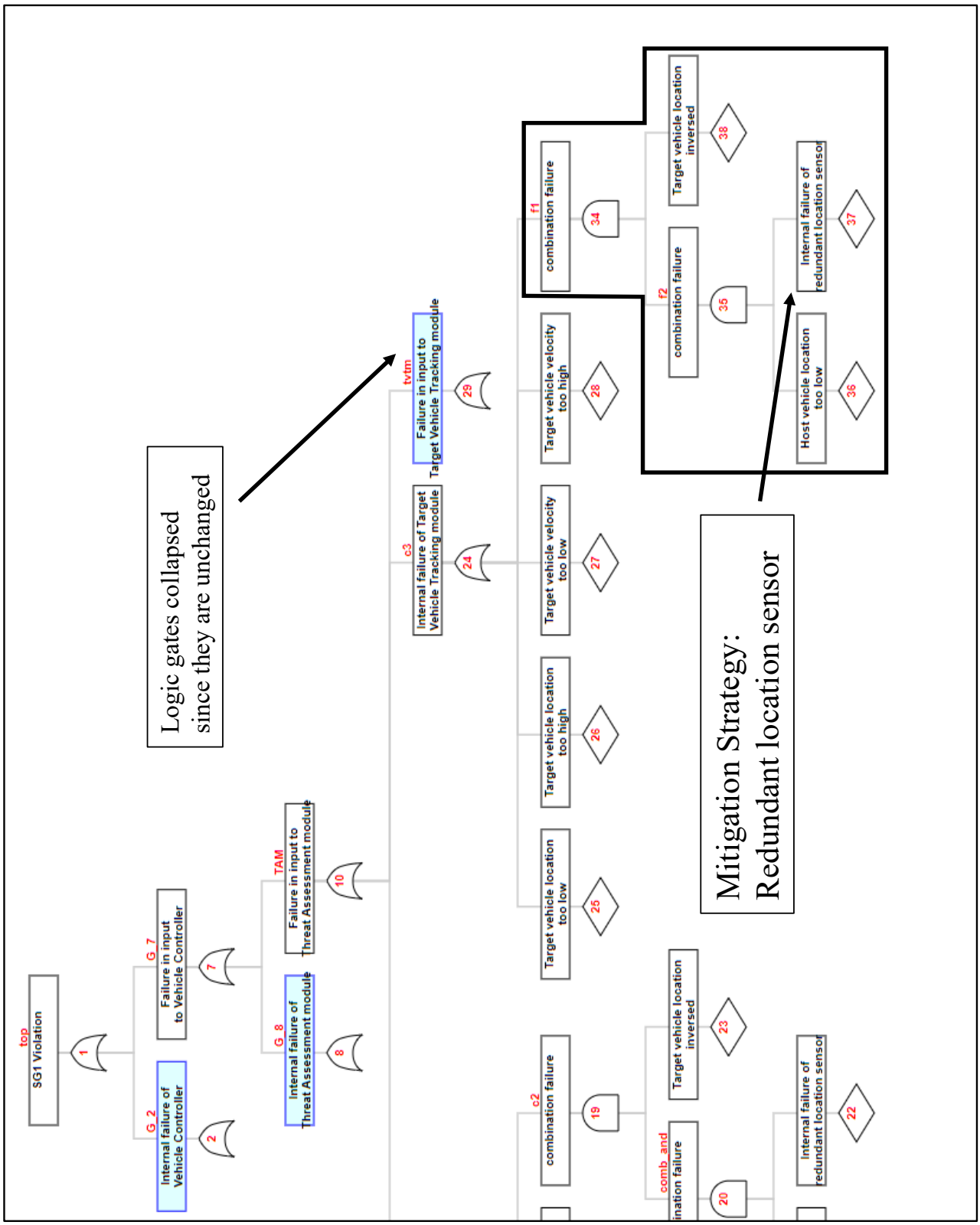
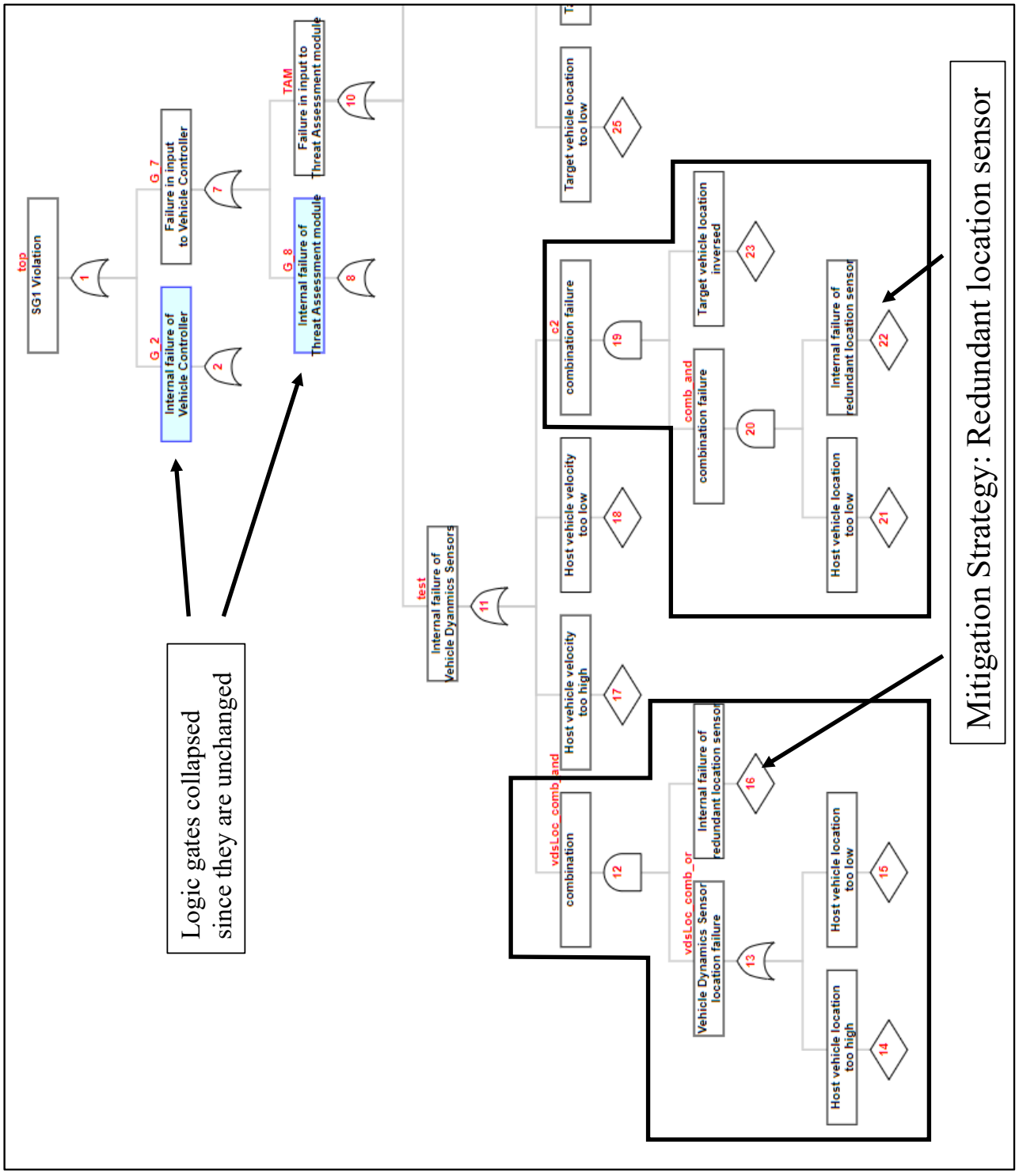


Figure 39: Component safety requirement allocated to the "Steering signal too high" and the "Steering signal too low" failure modes of the *Determine vehicle actuation* function which is allocated to the Vehicle Controller block



(a) Right side of the updated fault tree for violation of SG1



(b) Left side of the updated fault tree for violation of SG1

Figure 40: Updated fault tree for violation of safety goal SG1: Prevent late engagement of FCW system containing the new mitigation strategy for the redundant location sensor and the removal of the "Host vehicle location lost intermittently" failure mode from the Internal failure of Vehicle Dynamics Sensor block. Note that the internal failure of the redundant location sensor event combines with any failure mode of the Determine host vehicle location with an AND logic relationship.

5.2.2 Model-based safety verification

This section will illustrate the results of the model-based safety verification. The ISDS framework uses a simulation-based fault injection technique to verify if the system design satisfies the safety requirements (safety goals and component safety requirements). A CARLA client is developed to represent the FCW system designed in the left side of the framework (i.e., during system definition). As highlighted in Section 4.1.1, it is the responsibility of the system design and safety engineer to ensure that the SysML model is an accurate representation of the developed CARLA client. The model-based safety verification approach is highly automated and does not require any engineering intervention. The automated safety verification using the CARLA simulator is enabled by the use of the fault injection engine, which is one of the key contributions of this research. The fault injection engine consists of three components: the Fault Injection Configurator (FIC), Fault Injection Campaign Manager (FICM), and Safety Metric Evaluator (SME). The FIC module is responsible for extracting system design data, safety data, and simulation-specific properties from the SysML model and configuring the CARLA client with that data. The SME module is responsible for analyzing the results from fault injection to determine if any safety goals or component safety requirements were violated. The FICM module is responsible for executing the fault injection simulation for each failure operational scenario, forwarding the data to be analyzed by the SME module, and finally updating the SysML model with the results from safety verification (i.e., fault injection). Once the feedback to the SysML model is complete, the FMEA and fault tress are updated to reflect any changes.

In the FCW system case study, the FIC module of the fault injection engine extracts three failure operational scenario defined using the «sceanarioConfiguration» stereotype, three safety goals defined using the «safetyGoal» stereotype, 40 failure modes defined using the «failureMode» stereotype, three failure mode combinations identified using the "Combines with" dependency, and one component safety requirement defined using the «componentSafetyRequirement» stereotype. For each failure operational scenario, the FICM module completes one golden run and 43 faulty runs (40 for single failure modes and three for failure mode combinations). At the end of the golden run, the FICM module computes the TTC at which the FCW system engaged; this value acts as a reproducible reference or baseline for system behavior in the specific operational scenario and represents the current state-of-the-art safety assessment for FCW systems. This is the data that would be obtained by the NHTSA Forward Collision Warning Confirmation Test to assess the safety of the FCW system. For each of the 43 faulty runs, the FICM module injects the failure mode or failure mode combinations and computes the TTC at which the FCW system engages. The data from the golden and faulty runs (i.e., the TTC) is used to evaluate the criteria defined for the three safety goals and one component safety requirement. This analysis determines if any safety goal or component safety requirements was violated during the faulty run. Once the fault injection is complete for the three failure operational scenarios, the FICM module updates the results in the SysML model. Section 5.2.2.1.1 - section 5.2.2.1.4 shows the ability of the ISDS framework to assess a wider range of behaviors in the FCW system than the current safety assessment methods. Data from this section will contribute to answering research question 2. Section 5.2.2.2 shows the ability of the feedback mechanism in the ISDS framework to automatically update the SysML model with the results from safety verification and highlights how the framework eliminates a task that introduces inconsistencies. Data from this section will contribute to answering research question 1.

5.2.2.1 Results of fault injection simulation for FCW system

In total, 40 failure modes and three failure mode combinations were injected for each of the three failure operational scenarios. Figure 41, 42, 43 illustrate the results of the fault injection for each failure operational scenario. In each figure, the blue horizontal line represents the TTC for the golden run, called the golden TTC. It indicates the time at which the FCW engaged during the fault-free run. The orange columns represent the TTC for the faulty run, called the faulty TTC. They represent the time at which the FCW system engaged when the corresponding failure mode(s), shown along the x-axis, were injected. The label above each orange column highlights the safety goal violated and the TTC at which the FCW system engaged. The negative faulty TTC values represent instances when the FCW system failed to engage and did not register a TTC value. The next section will discuss the results for each scenario in greater detail.

5.2.2.1.1 Results for Scenario 1: Host vehicle encounters a stopped target vehicle on a straight road
Figure 41 illustrates the results from the fault injection for failure operational scenario 1. In this scenario, the host vehicle equipped with the FCW system encountered a stationary vehicle in its path on a straight road. The TTC for the golden run, indicated by the horizontal blue line, was 2.49 seconds. The NHTSA assessment requires this value to be $\geq$ 2.1 seconds. Hence, this system would pass safety assessment based on the current state-of-the-art. Of the 43 faulty runs for this scenario (40 faulty runs with single failure mode injection and three faulty runs with dual failure mode injection), 12 runs resulted in a violation of one or more safety goals. Figure 41 only highlights data from those 12 runs. The seven negative faulty TTC values, shown in Figure 41, represent instances when the FCW system failed to engage and did not register a TTC value. During the injection of these 7 failure modes, the FCW system encountered the H2: unexpected loss of FCW system hazard and violated SG2: Prevent unexpected loss of FCW system.

Since the NHTSA assessment criteria only requires the TTC $\geq$ 2.1 seconds, only nine out of the 12 faulty runs would fail its safety assessment. Since there is no upper limit to the criteria, it is difficult to classify if runs with a large faulty TTC value would pass or fail safety assessment based on the NHTSA assessment. In Figure 41, these runs correspond to the injection of the *Relative distance tracking data intermittently lost* failure mode which violates SG3: unexpected engagement of FCW system with a TTC of 8.17 seconds, and the combination of the *Host vehicle location too high* and *Target vehicle location too low* failure modes, which violate SG3: unexpected engagement of FCW system, with a TTC of 3.35 seconds. Hence, for scenario 1, in an FCW system that passes the NHTSA safety assessment (based on the golden run data), the ISDS framework is able to identify at least nine behaviors (characterized by the system's failure modes) where the FCW system would fail the NHTSA safety assessment and two other behaviors where it would pass NHTSA safety assessment but violate the system's safety goals, as defined during system definition.

5.2.2.1.2 Results for Scenario 2: Host vehicle encounters a decelerating target vehicle on a straight road
Figure 42 illustrates the results from the fault injection for failure operational scenario 2. In this scenario, the host vehicle equipped with the FCW system encountered a decelerating target vehicle in its path on a straight road. The TTC for the golden run, indicated by the horizontal blue line, was 2.67 seconds. The NHTSA assessment requires this value to be $\geq$ 2.4 seconds. Hence, this system would pass safety assessment based on the current state-of-the-art. Of the 43 faulty runs for this scenario (40 faulty runs with single failure mode injection and three faulty runs with dual failure mode injection), 20 runs resulted in a violation of one or more safety goals. Similar to the previous section, Figure 42 only highlights data from those 20 runs and the negative faulty TTC values

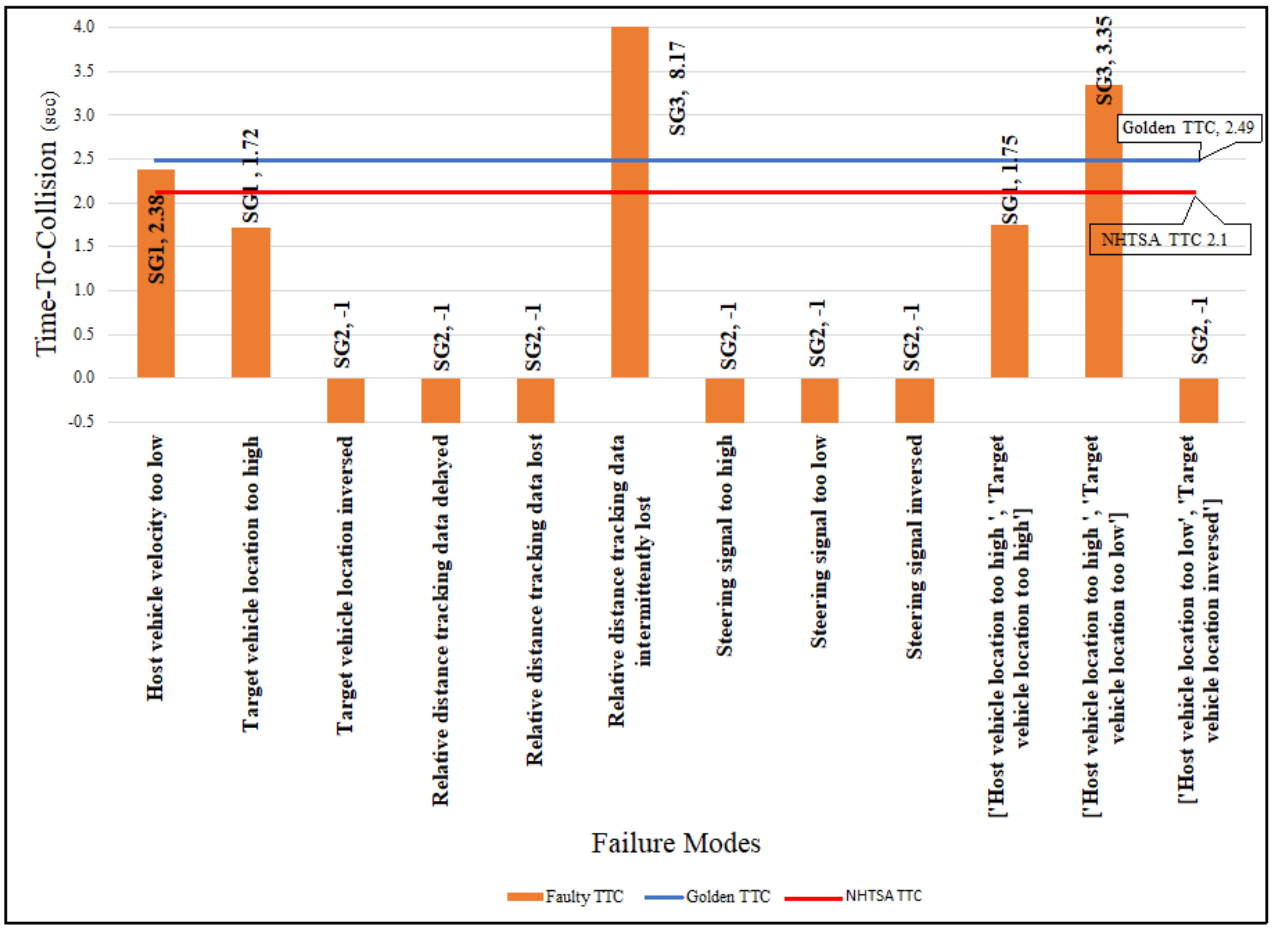


Figure 41: Results from safety verification for Scenario 1: Host vehicle encounters a stopped target vehicle on a straight road, showing the 12 out of 43 failure mode injections that resulted in a violation of one or more safety goals. The results show the nine behaviors of the FCW system where the faulty TTC, represented by the orange bars, is less than 2.1 seconds and two behaviors of the FCW system where it would pass NHTSA safety assessment but violate the system's own safety goals (i.e., orange bars where the TTC is greater than the golden run TTC value by at least 0.05 seconds).

shown for six failure modes represent instances when the FCW system failed to engage and did not register a TTC value. During the injection of these six failure modes, the FCW system encountered the H2: unexpected loss of FCW system hazard and violated SG2: Prevent unexpected loss of FCW system.

Since the NHTSA assessment criteria only requires the TTC $\geq$ 2.4 seconds, only 13 out of the 20 faulty runs would fail its safety assessment. Since there is no upper limit to the criteria, it is difficult to classify if runs with a large faulty TTC value would pass or fail safety assessment based on the NHTSA assessment. In Figure 42, these runs correspond to the injection of the *Target vehicle location lost* failure mode (violates SG3 with a faulty TTC of 10.7 seconds), *Target vehicle location delayed* failure mode (violates SG3 with a faulty TTC of 10.65 seconds), and the combination of *Host vehicle location too high* failure mode with *Target vehicle location too low* failure mode (violates SG3 with a faulty TTC of 3.67 seconds). Hence, for scenario 2, in an FCW system that passes the NHTSA safety assessment (based on the golden run data), the ISDS framework is able to identify at least 13 behaviors (characterized by the system's failure modes) where the FCW system would fail the NHTSA safety assessment and three other behaviors where it would pass NHTSA safety assessment but violate the system's safety goals, as defined during system definition.

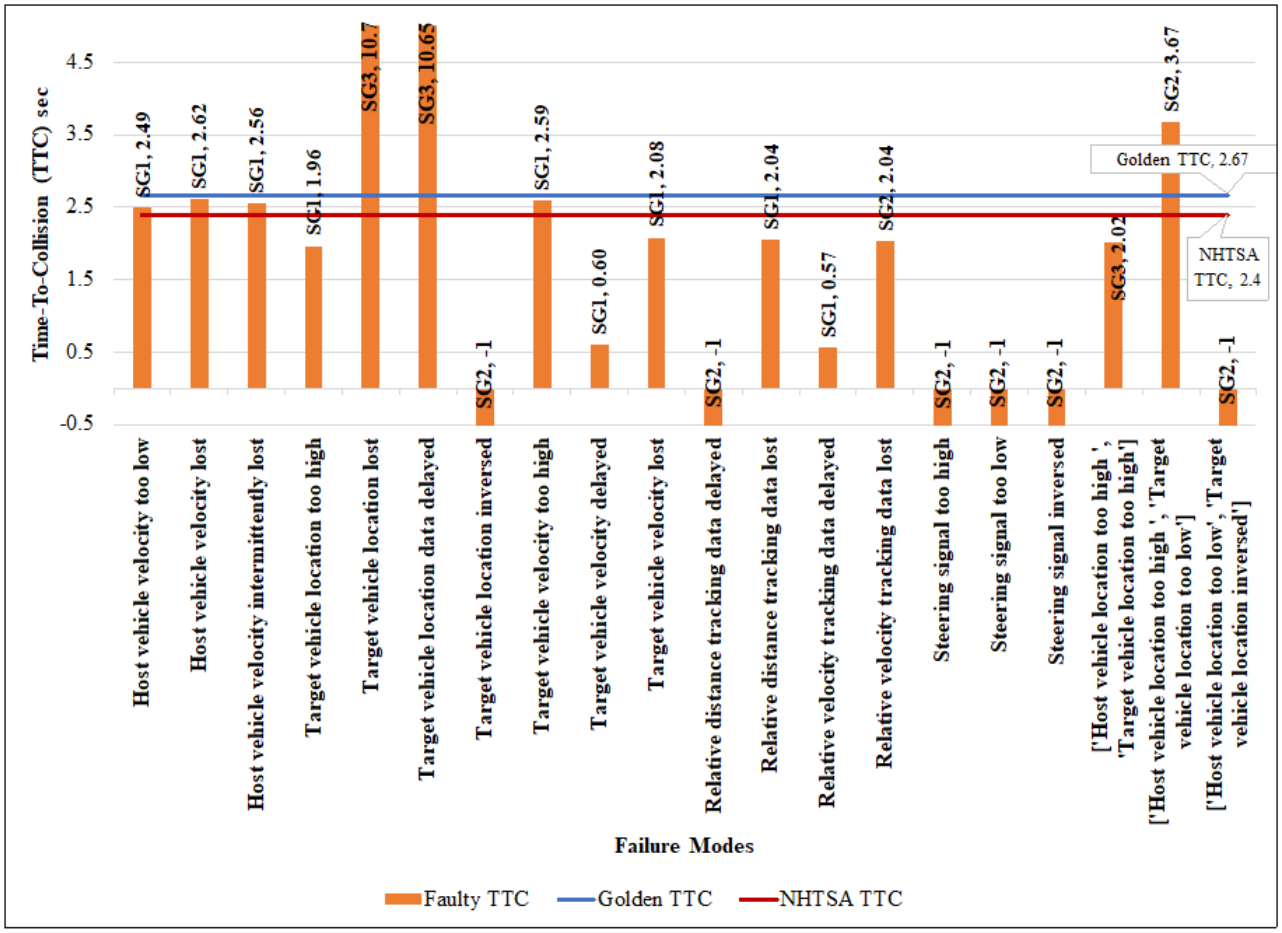


Figure 42: Results from safety verification for Scenario 2: Host vehicle encounters a decelerating target vehicle on a straight road. The results show the 13 behaviors of the FCW system where the faulty TTC, represented by the orange bars, is less than 2.4 seconds and the three behaviors of the FCW system where it would pass NHTSA safety assessment but violate the system's own safety goals (i.e., orange bars where the TTC is greater than the golden run TTC value, i.e., the blue line, by at least 0.05 seconds).

5.2.2.1.3 Results for Scenario 3: Host vehicle encounters a slower target vehicle on a straight road

Figure 43 illustrates the results from the fault injection for failure operational scenario 3. In this scenario, the host vehicle equipped with the FCW system encountered a slower target vehicle in its path on a straight road. The TTC for the golden run, indicated by the horizontal blue line, was 2.74 seconds. The NHTSA assessment requires this value to be $\geq$ 2 seconds. Hence, this system would pass safety assessment based on the current state-of-the-art. Of the 43 faulty runs for this scenario (40 faulty runs with single failure mode injection and three faulty runs with dual failure mode injection), 22 runs resulted in a violation of one or more safety goals. Similar to the previous section, Figure 43 only highlights data from those 22 runs and the negative faulty TTC values shown for eight failure modes represent instances when the FCW system failed to engage and did not register a TTC value. During the injection of these eight failure modes, the FCW system encountered the H2: unexpected loss of FCW system hazard and violated SG2: Prevent unexpected loss of FCW system.

Since the NHTSA assessment criteria only requires the TTC $\geq$ 2 seconds, only ten out of the 22 faulty runs would fail its safety assessment. Since there is no upper limit to the criteria, it is difficult to classify if runs with a large faulty TTC value would pass or fail safety assessment based on the NHTSA assessment. In Figure 43, these runs correspond to the injection of the *Target vehicle location too low* failure mode (violate SG3 with a

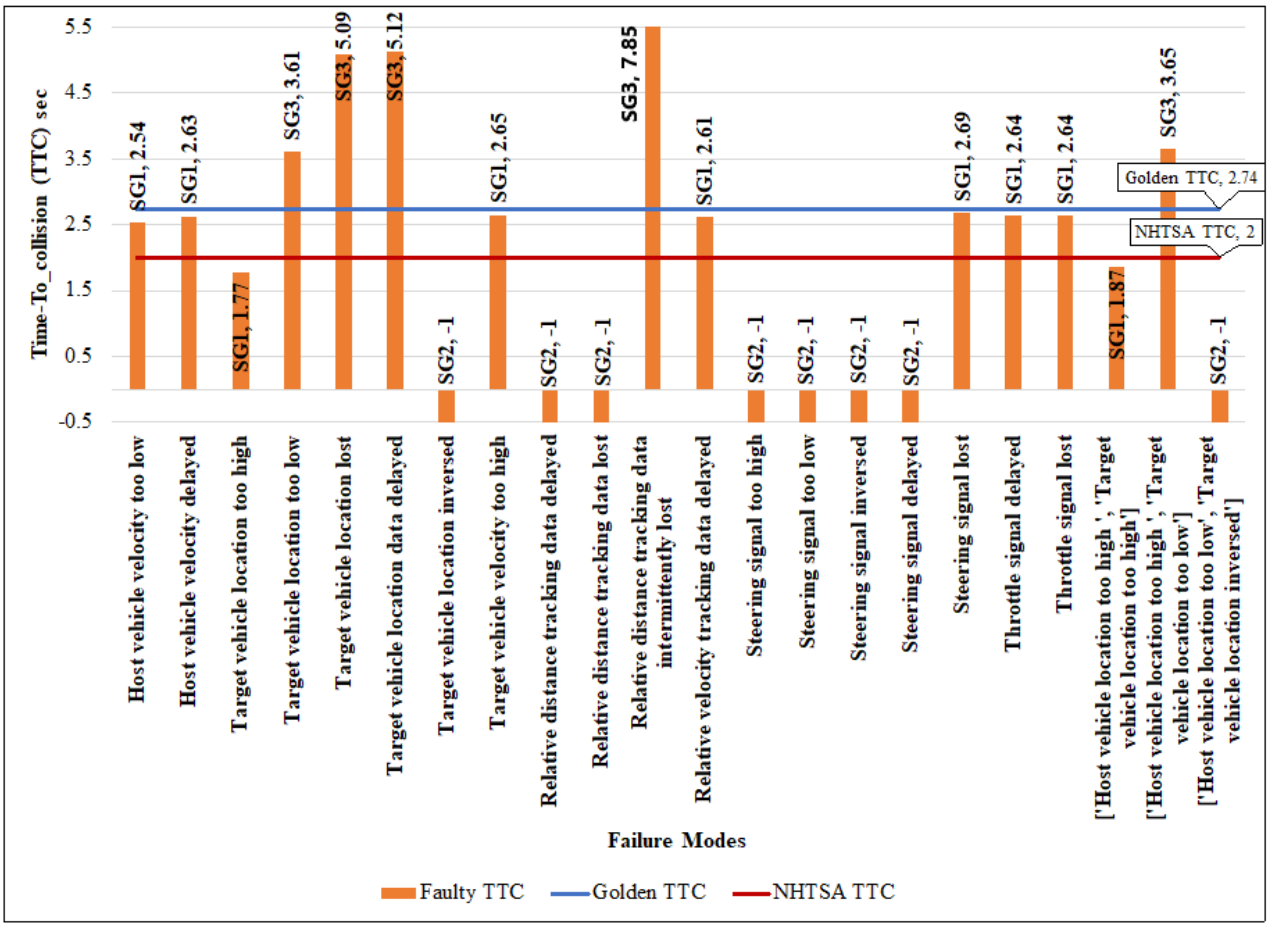


Figure 43: Results from safety verification for Scenario 3: Host vehicle encounters a slower target vehicle on a straight road. The results show the ten behaviors of the FCW system where the faulty TTC, represented by the orange bars, is less than 2 seconds and the five behaviors of the FCW system where it would pass NHTSA safety assessment but violate the system's own safety goals (i.e., orange bars where the TTC is greater than the golden run TTC value, i.e., the blue line, by at least 0.05 seconds).

faulty TTC of 3.61 seconds), *Target vehicle location lost* failure mode (violate SG3 with a faulty TTC of 5.09 seconds), *Target vehicle location delayed* failure mode (violate SG3 with a faulty TTC of 5.12 seconds), *Relative distance tracking data intermittently lost* failure mode (violate SG3 with a faulty TTC of 7.85 seconds), and the combination of *Host vehicle location too high* failure mode with *Target vehicle location too low* failure mode (violates SG3 with a faulty TTC of 3.65 seconds). Hence, for scenario 3, in an FCW system that passes the NHTSA safety assessment (based on the golden run data), the ISDS framework is able to identify at least ten behaviors (characterized by the system's failure modes) where the FCW system would fail the NHTSA safety assessment and five other behaviors where it would pass NHTSA safety assessment but violate the system's safety goals, as defined during system definition.

5.2.2.1.4 Results for evaluating mitigation strategy The model-based safety verification also allows engineers to assess if the mitigation strategy allocated to one or more failure modes is able to achieve the desired effect. For the FCW system, a redundant location sensor was allocated to the failure modes of the *Determine host vehicle location* function of the Vehicle Dynamics Sensor module. The redundant sensor activates if the primary sensor fails. Figure 44, 45, and 46 illustrates the results from the fault injection with and without the

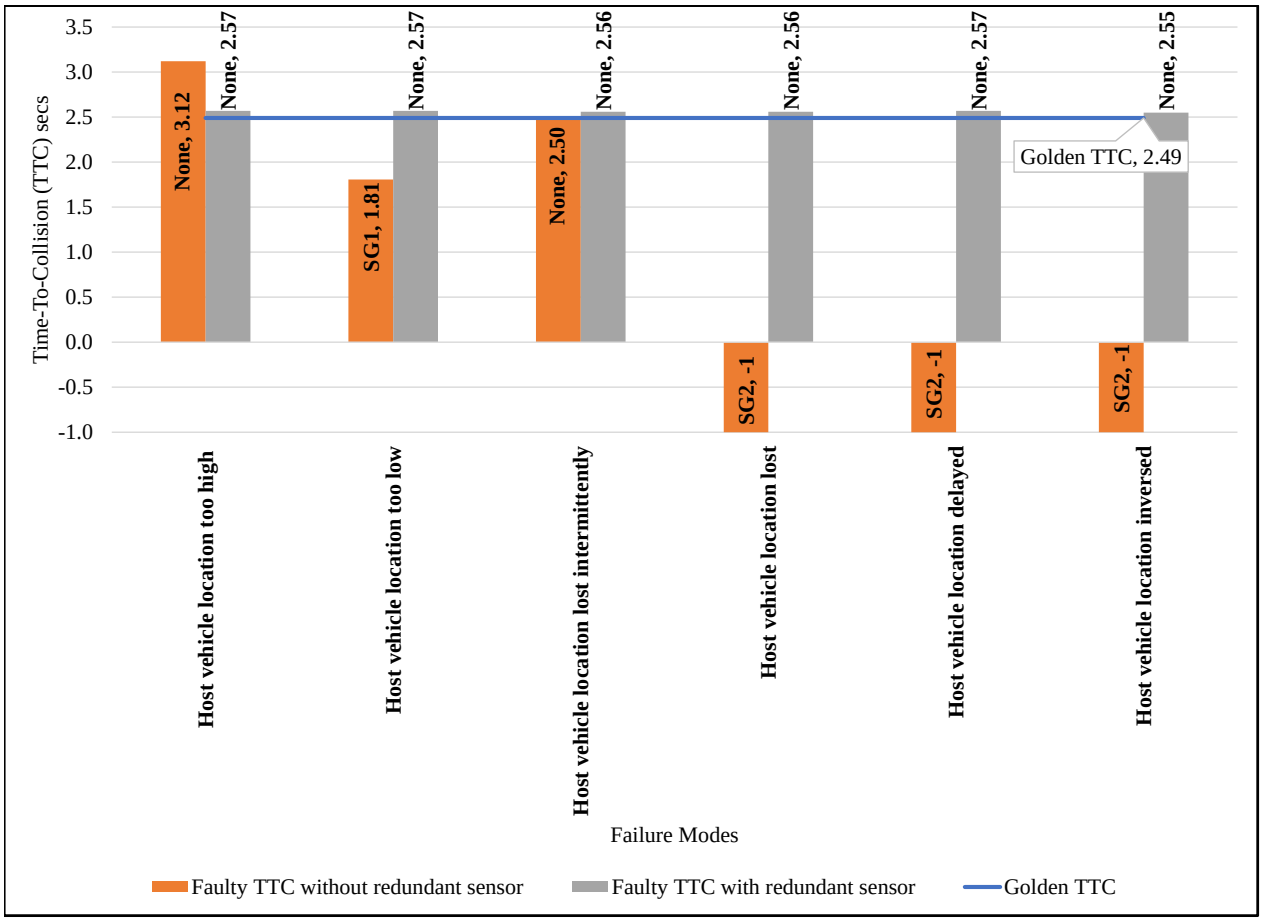


Figure 44: Results from safety verification of the redundant location sensor mitigation strategy for Scenario 1: Host vehicle encounters a stopped target vehicle on a straight road. The results show that the FCW system performs better with the mitigation strategy, shown in the gray columns, compared to the FCW system without the redundant sensor, shown in the red columns. With a redundant sensor, the FCW system engages at a TTC comparable to the TTC during the golden run (shown via the blue line) without violating a safety goal. In contrast, without the redundant sensor, the FCW system engages at a TTC with greater variations and four out the six failure modes violate safety goals.

redundant location sensor for scenario 1, 2, and 3 respectively. The *Determine host vehicle location* function contains six failure modes. The red column in each figure represents the faulty TTC when the failure modes of the function were injected without any mitigation strategy i.e., the system did not have a redundant location sensor. The gray columns represent the faulty TTC when the failure modes of the function were injected with a mitigation strategy present, i.e., the system was equipped with a redundant location sensor that was activated when the primary sensor failed. The blue horizontal line indicates the golden TTC for the failure operational scenario.

Figure 44 illustrates the results from the fault injection for failure operational scenario 1. Without a redundant sensor, (i.e., no mitigation strategy implemented), four out of the six failure modes injected clearly cause a safety goal violation, as indicated in the red columns. With a redundant location sensor, (i.e., the mitigation strategy is implemented), no safety goal violations occur, as shown in the gray columns. This data shows that with a redundant sensor, the FCW system engages 0.06 to 0.08 seconds earlier than it did during the golden run. In contrast, without a redundant sensor the FCW system engages by much wider margins than it did during the golden run. Based on this data, the engineer can conclude that the mitigation strategy is working as desired.

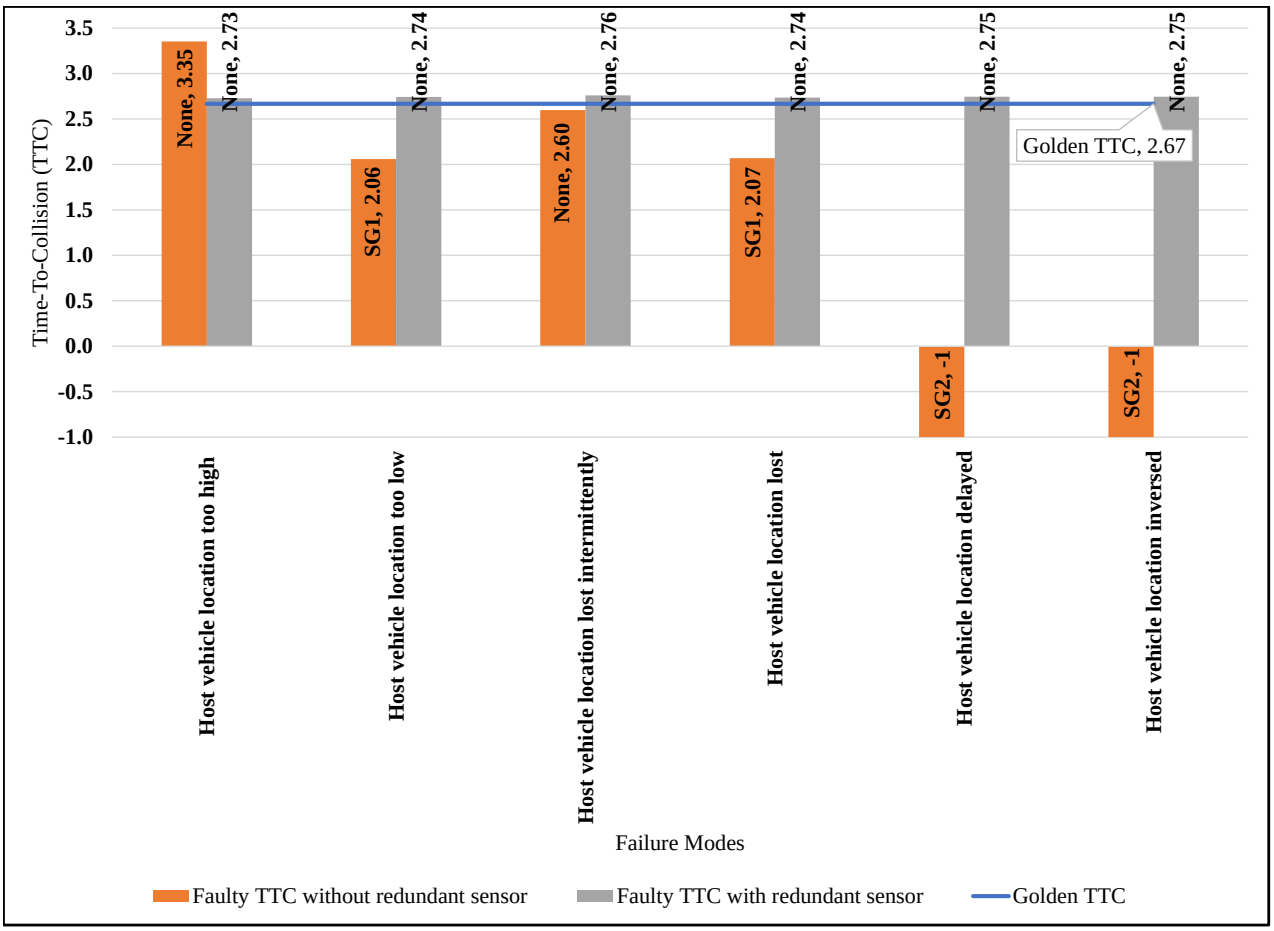


Figure 45: Results from safety verification of the redundant location sensor mitigation strategy for Scenario 2: Host vehicle encounters a decelerating target vehicle on a straight road. The results show that the FCW system performs better with the mitigation strategy, shown in the gray columns, compared to the FCW system without the redundant sensor, shown in the red columns. With a redundant sensor, the FCW system engages at a TTC comparable to the TTC during the golden run (shown via the blue line). In contrast, without the redundant sensor, the FCW system engages at a TTC with greater variations and four out of six failure modes violate safety goals.

Similarly, Figure 45 illustrates the results from the fault injection for failure operational scenario 2. Without a redundant sensor, (i.e., no mitigation strategy implemented), four out of six failure modes injected clearly cause a safety goal violation, as indicated in the red columns. With a redundant location sensor, (i.e., the mitigation strategy is implemented), no safety goal violations occur, as indicated in the gray columns. This data shows that with a redundant sensor, the FCW system engages 0.06 and 0.09 seconds earlier than it did during the golden run. In contrast, without a redundant sensor the FCW system violates the safety goals by wider margins. Based on this data, the engineer can conclude that the mitigation strategy is working as desired.

Finally, Figure 46 illustrates the results from the fault injection for failure operational scenario 3. Without a redundant sensor, (i.e., no mitigation strategy implemented), five out of six failure modes injected clearly cause a safety goal violation, as indicated in the red columns. With a redundant location sensor, (i.e., the mitigation strategy is implemented), no safety goal violations occur, as indicated in the gray columns. This data shows that with a redundant sensor, the FCW system engages 0.11 and 0.14 seconds earlier than it did during the golden run. In contrast, without a redundant sensor the FCW system violates the safety goals by wider margins. Based on this data, the engineer can conclude that the mitigation strategy is working as desired.

Once the fault injection simulation is complete and the results have been compiled by the fault injection

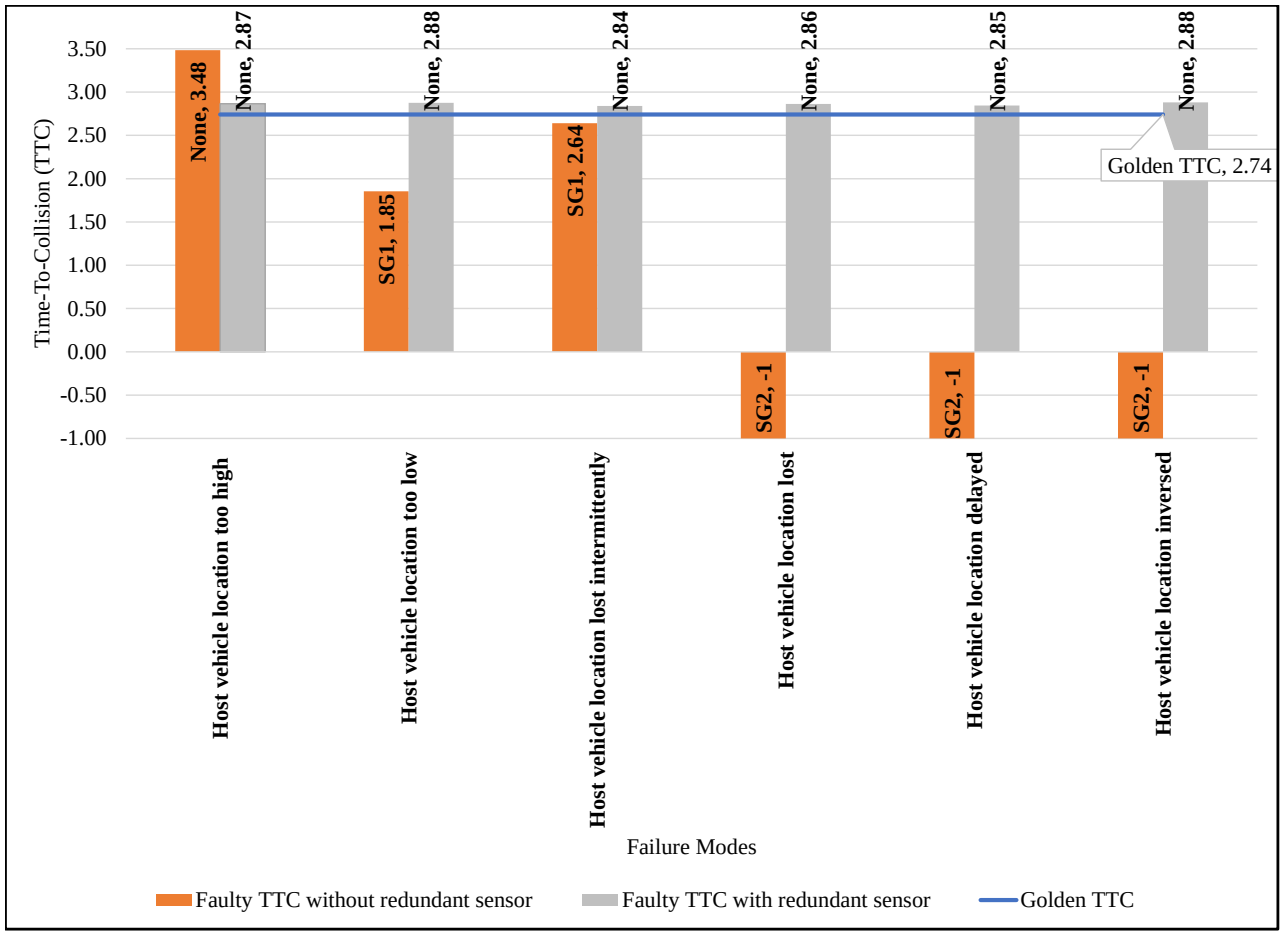


Figure 46: Results from safety verification of the redundant location sensor mitigation strategy for Scenario 3: Host vehicle encounters a slower target vehicle on a straight road. The results show that the FCW system performs better with the mitigation strategy, shown in the gray columns, compared to the FCW system without the redundant sensor, shown in the red columns. With a redundant sensor, the FCW system engages at a TTC comparable to the TTC during the golden run (shown via the blue line). In contrast, without the redundant sensor, the FCW system engages at a TTC with greater variations and five out of six failure modes violate safety goals.

engine, the next step is to update the SysML model with the results using the feedback mechanism of the framework. The next section will highlight the updates made to the SysML model via the feedback mechanism as well as the updates to the safety artifacts.

5.2.2.2 Feedback from safety verification to SysML model

The feedback mechanism in the ISDS framework is one of the key contributions of this dissertation and helps in answering research question 1. The feedback mechanism from safety verification to the SysML model is implemented by the FICM module in the fault injection engine. The module updates the XML file that was previously used to configure the CARLA simulator with the results from the safety verification. The updated XML file can be imported into the SysML tool to be viewed by the engineer.

Two types of SysML model elements are updated with results from the safety verification - the failure modes and the component safety requirements. Figure 47 illustrates the failure modes of the *Determine host vehicle location* function updated with results from the safety verification. For each failure mode, the "Single FM Simulation Results" tagged value is updated with the safety goal violation and failure operational scenario in

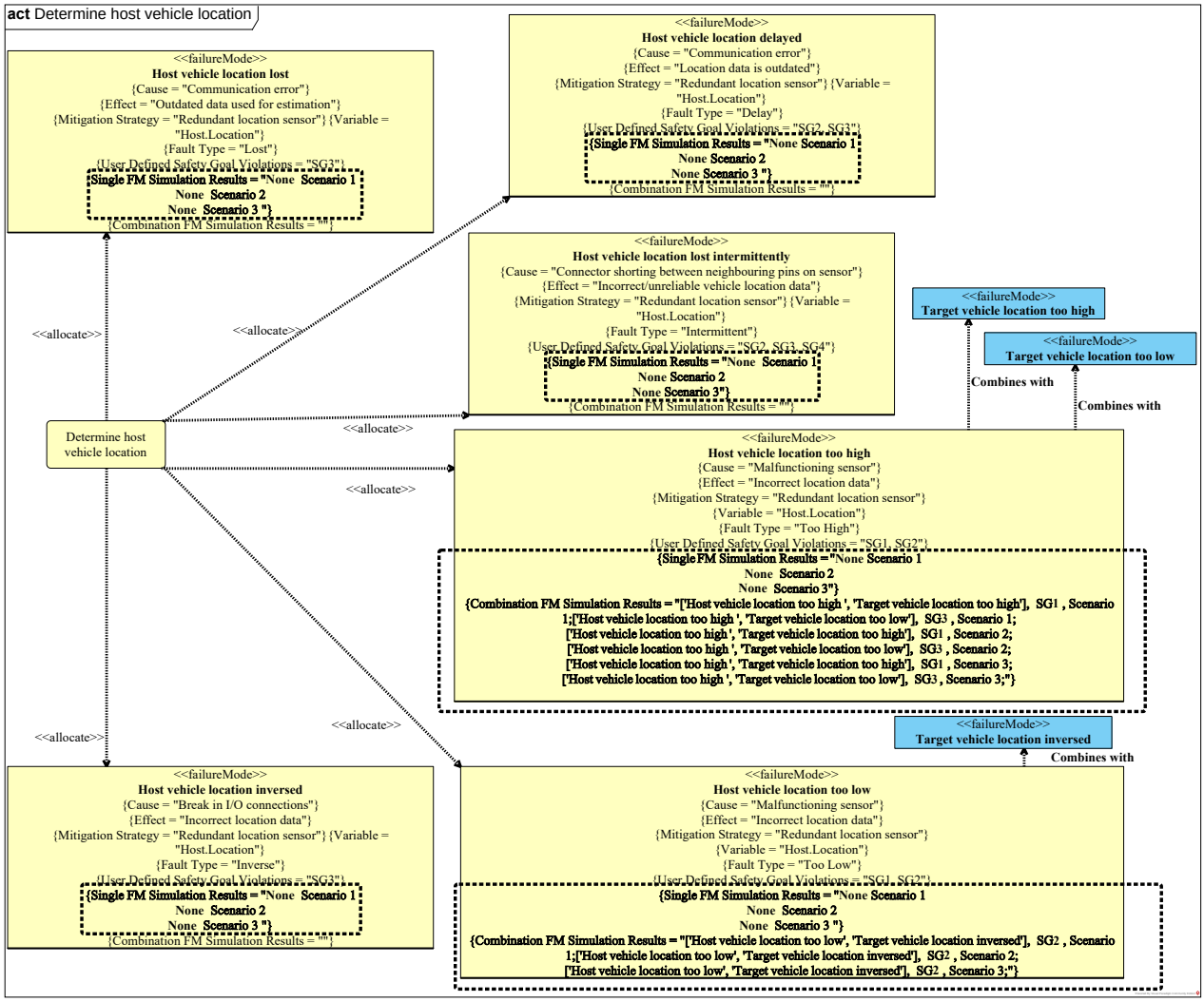


Figure 47: Failure modes of the *Determine host vehicle location* function updated with results from the safety verification. The "Single FM Simulation Results" and Combination FM Simulation Results" tagged values are updated with the safety goal violations and corresponding failure operational scenario during which the violation occurred.

which the violation occurred. For failure modes that combine with other failure modes, the "Combination FM Simulation Results" are updated with the names of the failure modes, the safety goal violations, and the failure operational scenario in which the violations occurred. These results are identical to the safety goal violations shown in Figure 41, 42, and 43.

The second SysML model element that is updated with the results of the safety verification is the component safety requirements. Figure 48 illustrates the component safety requirement *Yaw rate must not exceed yaw authority limits* updated with the results from the safety verification. The requirement's "Violation simulation data" tagged value is updated with the scenario in which the requirement was violated and the failure mode that was injected when the violation occurred. For the FCW system, all three failure operational scenarios require the vehicle to drive on a straight road. Consequently, the failure modes that affect this component safety requirement are limited to those of the steering actuator only as these are the only failure modes that can cause a sudden change in the yaw rate of the vehicle.

Once the feedback from safety verification to the SysML model is complete, the updated SysML model is used to generate a new FMEA table and four new fault trees of the system. The new FMEA table stores the

results from the safety verification in the simulation data column. Not only does this ensure consistency between the SysML model, safety artifact, and safety verification but it also allows the engineer to compare the safety goal violations detected using a formal verification approach against the user-defined safety goal violations, which was an analysis based on an engineer's intuition and experience. Figure 49 illustrates a small section of the updated FMEA table, which highlights the FMEA for the "Vehicle Dynamics Sensor" block.

Finally, the fault tree generation algorithm is re-run to generate fault trees based on the safety goal violations found during the fault injection simulation. A fault tree is generated for each safety goal, and each fault tree only contains failure modes that violate the safety goal. The failure modes included in the updated fault tree are based on the updated safety goal violations from safety verification. Figure 50 illustrates the updated fault tree for the safety goal SG1: Prevent late engagement of FCW system. The updated fault tree is generated using the results from the safety verification, i.e., using the simulation data. Compared to the fault tree generated after model-based safety analysis (or before model-based safety verification), the final fault tree contains 23 changes made to its events (events were either added or removed). The feedback mechanism of the ISDS framework eliminated the need for engineers to perform this step manually by automating the feedback from safety verification to the SysML model as well as by automatically generating the fault tree. This data directly contributes towards answering research question 1. Generating the fault trees for each safety goal based on the results from safety verification marks the end of the model-based safety verification approach and completes the model-based safety assessment of the FCW system.

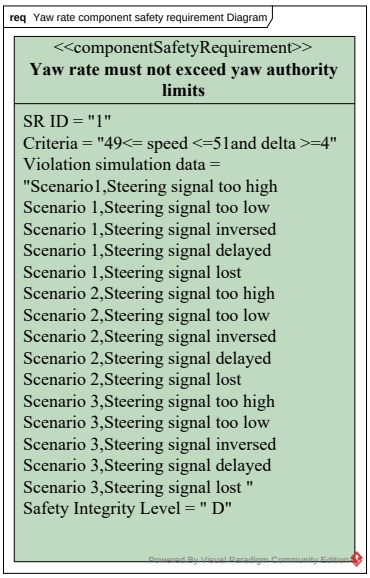
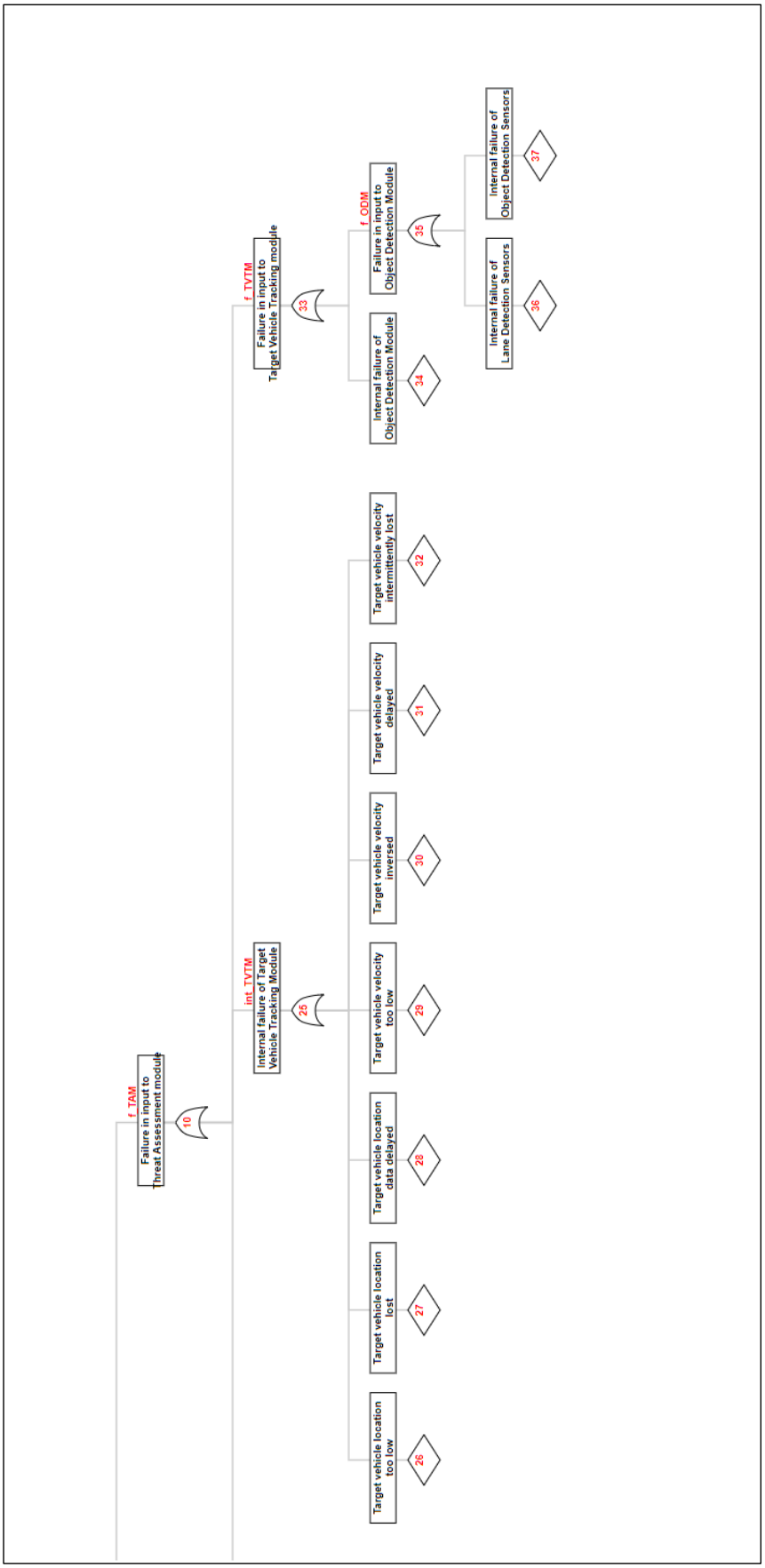


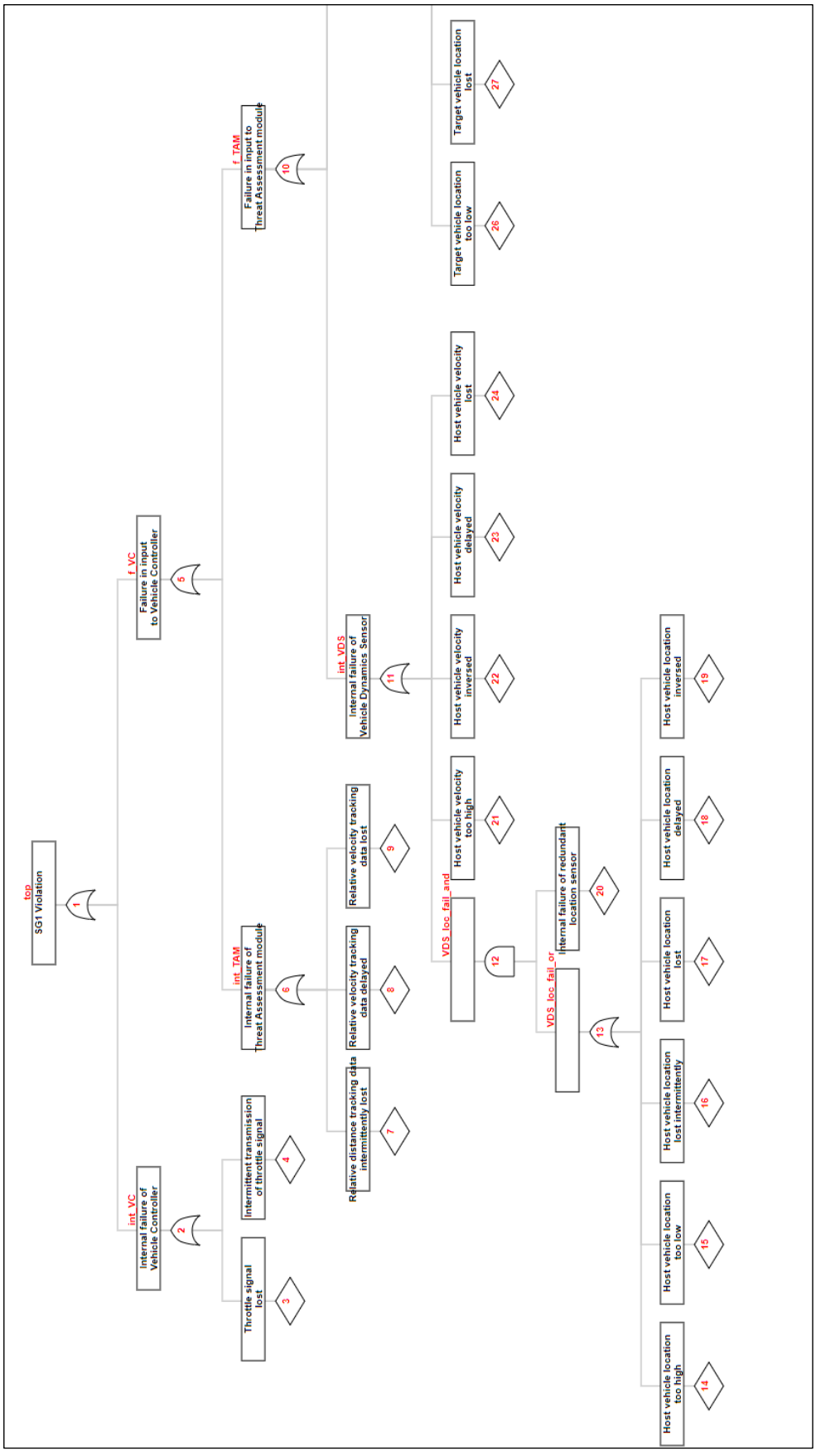
Figure 48: The *Yaw rate must not exceed yaw authority limits component safety requirement* updated with the results from the safety verification. The "Violation Simulation data" tagged value is updated with the failure operational scenario during which the requirement was violated as well as the failure mode that caused the violation.

Block	Function	Failure Mode	Cause	Safety Goal Violation	Risk	Effect	Mitigation Strategy	Simulation Data
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location too high	Malfunctioning sensor	SG1	D	Incorrect location data	Redundant location sensor	None Scenario 1 None Scenario 2 None Scenario 3
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location too low	Malfunctioning sensor	SG1	D	Incorrect location data	Redundant location sensor	None Scenario 1 None Scenario 2 None Scenario 3
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location lost intermittently	Connector shorting between neighboring pins on sensor	SG1,SG2	D	Incorrect/unreliable vehicle location data	Redundant location sensor	None Scenario 1 None Scenario 2 None Scenario 3
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location lost	Communication error	SG2	D	Outdated data used for estimation	Redundant location sensor	None Scenario 1 None Scenario 2 None Scenario 3
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location delayed	Communication error	SG1,SG2	D	Location data is outdated	Redundant location sensor	None Scenario 1 None Scenario 2 None Scenario 3
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location inversed	Break in I/O connections	SG2	D	Incorrect location data	Redundant location sensor	None Scenario 1 None Scenario 2 None Scenario 3

Figure 49: FMEA for the "Vehicle Dynamics Sensor" block that has been updated with the results from safety verification. The Simulation Data column is updated with the safety goal that was violated when the failure corresponding to that row was injected as well as the scenario during which the violation occurred.



(a) Right side of fault tree for violation of SG1



(b) Left side of fault tree for violation of SG1

Figure 50: Fault tree for violation of safety goal SG1 updated with results from safety verification. The fault tree only contains failure modes that were found to violate safety goal 1 during the fault injection simulation.

5.3 Discussion

In the previous section, the ISDS framework has been applied towards the development and safety assessment of a Forward Collision Warning (FCW) system to evaluate its effectiveness as a model-based safety assessment framework and to demonstrate its ability to overcome the limitations of the current state-of-the-art. This section will tie the results of the case study to the research questions formulated in Chapter 3, Section 3.4.

5.3.1 Research Questions

Two research question were formulated in Chapter 3: the first question investigated the ability of the framework to eliminate tasks that introduce inconsistencies in safety assessment, and the second question investigated if the framework could assess the safety of the system for a wider range of behaviors than the current state-of-the-art.

5.3.1.1 Research Question 1

In the FCW system case study, does the ISDS framework eliminate tasks that introduce inconsistencies compared to the current state-of-the-art?

Chapter 2 highlighted the sources of inconsistencies that exist in literature related to model-based safety assessment. The sources of inconsistencies include -

- Inconsistencies that arise when different versions of the same system design model and/or safety model are created for performing safety assessment (related to safety analyses and safety verification). As the system design evolves, each version of the model will have to be manually updated before performing each analysis. This problem was resolved by researchers by using a single SysML model to store the system design data as well as the safety data via SysML profiles or by using model transformation to generate the safety model from the system design model. In either case, the use of a single SysML model to store the system design and/or safety data eliminated the need for different versions of the same model.
- The lack of feedback from safety artifact to the SysML model as well as the lack of feedback from safety verification to the SysML model introduce inconsistencies between the safety artifact, safety verification and the SysML model. This is an open research question and the focus of research question 1.

Due to the lack of feedback from the safety artifact to the SysML model, an engineer has to manually update any changes made to the safety artifact in the SysML model as well. Similarly, the results from safety verification have to manually be updated in the SysML model. The manual method for feedback can introduce inconsistencies in safety assessment and represents the current gap in the literature. The ISDS framework's feedback mechanism was developed to eliminate this source of inconsistency. The framework's feedback mechanism uses the SysML safety profile and algorithms to automate the feedback process and eliminates the task that requires engineers to manually update the SysML model. The safety profile links each element in the FMEA table to an element in the SysML model. The framework's algorithms leverage this link to automatically update the SysML model with the data from the FMEA table. Similarly, the safety profile contains placeholders to store the results from safety verification in the SysML model. The framework's algorithms can automatically store the results from the safety verification in these placeholders.

In the FCW system case study, Section 5.2.1.8 highlights the feedback mechanism from the safety artifact to the SysML model. The manual method for feedback would require the engineer to update seven fields in the SysML model based on the changes made in the FMEA table, six changes related to the added mitigation strategy and one change related to the modified safety goal violation. The ISDS framework's feedback mechanism

automates this process and eliminates the need for an engineer to complete this task. Similarly, Section 5.2.2.2 highlights the feedback mechanism from safety verification to the SysML model. In total, 129 failure mode injections were completed (43 failure modes were injected for three failure operational scenarios), and each failure mode injection contains a safety goal violation that needs to be updated in the SysML model. The manual method for feedback would require the engineer to update 129 fields in the SysML model based on the results from safety verification. The ISDS framework's feedback mechanism automates this process and eliminates the need for an engineer to complete this task. Hence, the FCW system case study is able to demonstrate that the ISDS framework does eliminate the need for engineers to manually feedback changes from the safety artifact to the SysML model or manually feedback the results from safety verification to the SysML model, thereby eliminating tasks that introduce inconsistencies in safety assessment.

5.3.1.2 Research Question 2

In the FCW system case study, does the ISDS framework assess the safety of the system for a wider range of behaviors than the current safety assessment method for FCW systems?

Chapter 2 highlighted the need for a safety assessment approach that does not rely heavily on field-testing and can use simulation testing to observe faulty behavior caused by component degradation early in the life cycle in order to improve the safety of the system. The current safety assessment method developed by the NHTSA evaluates the FCW system in three scenarios and assesses if the TTC at which the FCW system engages is greater than the minimum TTC that has been assigned for each scenario. The ISDS framework uses the same scenarios and TTC criteria to perform fault injection simulation and evaluates the behavior of the system under fault-free conditions as well as faulty conditions.

In the FCW system case study, Section 5.2.2.1 highlights the results of the safety verification for each of the three scenarios. The current safety assessment method for FCW systems involves performing a fault-free run of the system in each of the three scenarios. The FCW system developed in the case study passes safety assessment in all three scenarios. However, for the same FCW system design evaluated during scenario 1, the ISDS framework is able to identify at least nine behaviors (characterized by the system's failure modes) where the FCW system would fail the NHTSA safety assessment and two other hazardous behaviors where it would pass NHTSA safety assessment but violate the system's safety goals, as defined during system definition. Similarly, for the same FCW system design evaluated during scenario 2, the ISDS framework is able to identify at least 13 behaviors (characterized by the system's failure modes) where the FCW system would fail the NHTSA safety assessment and three hazardous behaviors where it would pass NHTSA safety assessment but violate the system's safety goals, as defined during system definition. Finally, for the same FCW system design evaluated during scenario 3, the ISDS framework is able to identify at least ten behaviors (characterized by the system's failure modes) where the FCW system would fail the NHTSA safety assessment and five hazardous behaviors where it would pass NHTSA safety assessment but violate the system's safety goals, as defined during system definition. Overall, for the FCW system case study, the ISDS framework is able to assess the safety of the system for a wider range of behavior than the current safety assessment method for FCW systems by using simulation testing.

5.4 Summary

This chapter applied the ISDS framework, described in Chapter 4, to a case study to evaluate its effectiveness as a model-based safety assessment framework. The case study involves the development and safety assessment of an FCW system in an autonomous vehicle. The chapter begins by introducing the reader to FCW systems and the current safety assessment method used by the NHTSA to evaluate the safety of such systems. Next, the ISDS framework is applied to the FCW system case study by first implementing the framework's model-based safety analysis approach followed by implementing the model-based safety verification approach. The chapter concludes by discussing the results of the case study and answering the research questions formulated for this dissertation.

Chapter 6

Conclusion

As system complexity and sophistication increases, so does the challenge in assessing the safety of such systems. The safety assessment of a system judges the safety conformance and safety integrity achieved by every safety instrumented function within the system. This assessment is based on either assessing the safety of the system development process or the safety of the system design. This dissertation seeks to address the challenges in safety assessment based on the safety of the system design.

The first key challenge lies in the inconsistencies that are introduced by tasks that safety engineers perform manually during safety analysis and safety verification. Experts recommend assessing the safety of the system design using a combination of safety analyses, safety verification, and testing at various stages in the life cycle [12, 13, 14, 15, 16]. However, to identify safety-related design issues early in the life cycle, researchers have focused more on safety analyses and safety verification of early design models instead of testing, which is usually performed later in the life cycle. A key issue for safety analysis and safety verification of early system design models lies in the inconsistencies that arise between the current system design model and the results from the analyses. Safety analysis and safety verification are often performed on independent tools that require a re-description or transformation of the original system design model to the language/model used by the tool. When this process is manually performed by an engineer, it is time-consuming and error-prone. If it is automated using model-transformation, it is accompanied by loss of data between the different versions of the system design model. Additionally, feedback of results from the analyses to the system design model is either non-existent or at best a manual process. Such feedback is time-consuming and error-prone and may introduce inconsistencies between the system design model and results of safety analysis or safety verification. The inconsistencies that arise between the current system design, the safety analysis, and the safety verification subsequently can lead to an incorrect safety assessment. The first objective of the ISDS framework proposed in this dissertation was to eliminate tasks in the safety assessment process that are sources of inconsistency.

Another key challenge in safety assessment lies in the need to characterize faulty system behavior that is caused by component degradation or unforeseen environment disturbances early in the life cycle without relying on field-testing. Performing a safety assessment that relies on field-testing to observe such behavior is not feasible as it is an inefficient way to observe system behaviors arising due to these factors. This is because it is unlikely that the system will exhibit a range of exceptional behavior during normal testing conditions.

Consequently, there is a need for a safety assessment approach that does not rely on field-testing and allows engineers to characterize faulty system behavior under component degradation or environmental disturbances early in the life cycle. With the increased availability of high-fidelity simulators and increased computing power, simulation testing offered a potential solution to this problem. The second objective of the ISDS framework was to address the lack of simulation testing in current safety assessment approaches and to use simulation to observe faulty system behavior caused by component degradation.

Chapter 2 reviewed the current state-of-the-art approaches aimed at addressing these challenges in safety analysis and safety verification. Approaches that used MBSE, particularity SysML-based approaches, showed promising results in addressing the challenges of safety analysis within safety assessment. For safety analysis, the approaches performed one or more safety analyses using a single SysML model via SysML profiles or model transformation methods. Safety artifacts such as FMEA tables and fault trees were automatically generated from the SysML model. However, the lack of feedback from the safety analyses to the SysML model is yet to be resolved and highlights the current gap in the literature. Similarly, SysML-based MBSE approaches that use simulation-based fault injection for verifying the safety of the system design have shown promising results in addressing the challenges of safety verification within safety assessment. However, the lack feedback of the results from safety verification to the SysML model still exists and highlights another gap in the literature. The ISDS framework proposed in this dissertation overcomes the limitations of the current-state-of-the-art and addresses the gaps in the literature.

Chapter 3 discussed the research methodology and research questions that guided this dissertation. The framework proposed in this dissertation was developed using the design research methodology. The framework was evaluated via a case study, where it was used to develop and assess the safety of a Forward collision Warning (FCW) system in an autonomous vehicle. Two research questions were investigated in the FCW system case study, i.e., if the ISDS framework eliminated tasks that introduced inconsistencies in the safety assessment process and if the ISDS framework improved the safety of the system by assessing a wider range of behaviors as compared to the current state-of-the-art.

Chapter 4 described the ISDS framework and the approach adopted for model-based safety analysis as well as model-based safety verification. To perform model-based safety analysis, the framework used SysML profiles to store system design and safety data in the same SysML model. The framework's algorithms, written in python, extract data from the SysML model to automatically generate FMEA tables and fault trees. The feedback mechanism of the framework updates the SysML model with any changes made to the FMEA table. This feedback ensures that consistency is maintained between the SysML model and safety analysis. To perform model-based safety verification, the framework uses simulation-based fault injection to observe how the system design behaves in the absence and presence of faults and evaluate if any resulting behavior of the design violates the safety requirements of the system. The framework uses SysML profiles to store system design data, safety data, and simulation-specific properties in the SysML model. This data is used to configure the system in the CARLA simulator. The framework injects the system's failure modes (extracted from the SysML model) and evaluates if any safety goals or component safety requirements were violated. The feedback mechanism of the framework updates the safety goal and component safety requirements violations in the SysML model, based on which the FMEA table and fault trees are updated. This feedback ensures consistency between the SysML model, the safety artifacts (i.e., safety analysis) and safety verification.

Finally, Chapter 5 evaluated the ISDS framework by applying it towards the development and safety assessment of an FCW system. The data from the case study was used to answer the research question

formulated in Chapter 3. In response of the first research question, the results show that the automated feedback mechanism of the ISDS framework eliminates the need for engineers to manually update the SysML model, thereby eliminating a source of inconsistency to the safety assessment process. The update includes the feedback from the safety artifact (i.e., safety analysis) to the SysML model as well as from safety verification to the SysML model. In response to the second research question, the results highlight that in a system that passes safety assessment using the current state-of-the-art method and metric, the ISDS framework is able to identify additional system behaviors that would result in the system failing safety assessment using the same metric. This result shows that the ISDS framework improves the safety of the system by assessing a wider range of behaviors than the current state-of-the-art.

The ISDS framework in its current form is ideally suited for safety assessment of complex systems in the automotive industry, particularly highly automated or autonomous vehicles. As the framework is closely coupled to the CARLA simulator, it is limited to systems that can be modeled and simulated in it. The CARLA simulator is championed by several leaders in the autonomous vehicle industry, like Intel, Toyota Research Institute, and the Computer Vision Center, which further supports the applicability of the ISDS framework in this domain. It can be used by safety engineers and system architects to evaluate their system designs early in the life cycle, evaluate system safety mechanisms and mitigation strategies for different scenarios, sub-systems or components and improve the confidence in the safety of the system design earlier in the life cycle. Advanced driving technologies in an automobile, like lane departure warning, blind spot monitoring, adaptive cruise control etc., can be developed and assessed using the ISDS framework, as done in the case study. It can also serve as a blueprint for similar safety assessments in other domains. High-fidelity simulators from other industries like aviation, robotics, etc. can be used in place of the CARLA simulator. The fault injection engine will need to be modified to configure and run the specific simulator; however, the framework can provide guiding principles to (in theory) perform similar safety assessments in other domains.

The results of the case study highlight the utility of the ISDS framework as a model-based safety assessment framework. The implications of this research include minor aspects such as reinforcing the case for model-based development as a useful tool for complex system development as well as major aspects such as the ability to reduce the time and money required to identify safety related design issues in an automotive system, identify safety strategies and system designs for dangerous scenarios, and evaluate potential hazards early in the life cycle instead of during production and deployment. This reduced reliance on field-testing has the potential to catch hazardous system behavior early and potentially save human lives!

6.1 Research Contributions

This dissertation contributed to the existing body of knowledge by extending the current state-of-the-art in 2 key areas -

1. The feedback mechanism of the ISDS framework eliminated the need for engineers to a) manually update changes made to the safety artifact in the SysML model and, b) manually update the results from safety verification to the SysML model. Consequently, the ISDS framework eliminated a key source of inconsistency in the safety assessment process. The automated feedback mechanism combined with the ability to automatically generate FMEA tables and fault trees allows the ISDS framework to capture the non-linear and iterative nature of safety assessment while maintaining consistency between the SysML model, safety analyses (i.e., safety artifacts), and safety verification.

2. The fault injection engine of the ISDS framework enables the use of simulation testing to observe faulty behavior caused by component degradation that may have been missed by approaches that rely on field-testing alone. The fault injection engine allows engineers to observe system behavior under fault-free and faulty conditions. By configuring the CARLA client based on the SysML model, the ISDS framework assesses the safety of an automotive system by determining if any safety requirements, safety goals or component safety requirements, were violated after injecting the failure modes defined for the system in the SysML model.

6.2 Limitations

This section highlights the limitations of the research proposed in this dissertation that must be acknowledged and can be overcome by future research -

1. The results from the FCW system case study demonstrates that the ISDS framework improves on the current state-of-the-art by eliminating sources of inconsistency and assessing safety of the system for a wider range of behaviors (answers to research question 1 and 2, respectively). However, the framework is limited to evaluating automotive systems that can be characterized in CARLA. The FCW system represented the type of complex system that introduces novel challenges in safety assessment and the case study demonstrated the type of issues this dissertation aimed to address. Hence, while it does show promise, the framework must be evaluated for a broad range of conditions and applications before the results from the case study can be generalized across other domains or applications. Future research can look at the effort required to apply the fault injection engine of the ISDS framework in other high-fidelity simulators across different domains.
2. A key assumption of this research was that the CARLA client, which is the code running in the CARLA simulator representing the system under development, was consistent with the model developed in the SysML model. It is the responsibility of the system design and safety engineer to ensure that the SysML model is an accurate representation of the developed CARLA client. Only the configurable parameters in the CARLA client are defined by data from the SysML model. However, this could potentially be a source of inconsistency. Future research can look at automated generation of executable code from SysML models, which would circumvent the need to develop the CARLA client independent from the SysML model.
3. While the automatic generation of FMEA tables and fault trees coupled with the automatic feedback mechanism does eliminate manual tasks that introduce inconsistencies, the framework does require engineers to manually update the SysML model with the mitigation strategy identified for each failure mode. Modeling is still largely a manual effort and requires the engineer to ensure consistency with the code running in CARLA.
4. Fault coverage is limited to the failure modes identified in the FMEA and the fault models that are supported by the framework for the CARLA simulator. Currently, the failure modes are restricted in the type of faults it can inject - velocity, location, and actuator faults, or transmission faults. Consequently, framework cannot ensure that all possible faults have been covered. While 100% fault coverage is not realistically possible, the constraints on fault coverage must be acknowledged.

6.3 Future Work

There are several avenues for future work that can extend the ISDS framework. The results from the case study shows promise to extend the framework across different domains like aviation, marine systems or robotics. The need for early identification of safety-related design issues is consistent across different domains and highlights the need to further extend this framework. The fault injection engine will have to be modified based on the domain and a corresponding simulator will have to be used. It will be worthwhile to study and document the effort required to modify the current framework for each domain.

Another exciting possibility for the framework is to support automated generation of executable code from SysML models. Making use of automatic code generation will allows engineers to generate the CARLA client completely from the SysML model and remove the responsibility of ensuring that the client is consistent with the SysML model off the shoulder of the design and safety engineers. This effort would require a comprehensive SysML profile that would allow for translating complex python code supported by the CARLA simulator's libraries into SysML model elements and vice-versa.

To eliminate the need for engineers to manually insert potential mitigation strategies in the FMEA table and update the SysML model, a potential solution could be to exploit the reuse-ability in model-based development and evaluate the ideal mitigation strategy from a list of solutions that have been pre-defined or maintained in a database based on previous use. Engineers will no longer need to manually intervene as the fault injection engine can run safety verification on each model and pick the one that perform best based on the defined safety requirements.

Another direction is to extend the types of faults that the fault injection engine that can inject in the CARLA simulator. This can help improve the fault coverage of the framework and support more fault models for automotive systems. For example, by supporting perception-related faults, the ISDS framework will be able to inject failure modes corresponding to image or lidar systems to determine how those faults impact the system's safety requirements. This work will help extend the range of behaviors that the ISDS framework can assess.

Another direction would be to update the algorithms of the ISDS framework to support parsing XMI files of the SysML model instead of XML file. Due to limited tool support, the ISDS framework currently requires the SysML model to be exported in the XML file format and is constrained to using the Visual Paradigm tool for SysML modelling. As tool support for the XMI file format improves, the algorithms can be updated to handle XMI files, which would allow the framework to parse a SysML model from any SysML modelling tool.

Appendix A
Risk assessment for hazard H2 and H3

Assessment Variable	Description
Hazard	H2: Unexpected loss of FCW system
Failure operational scenario	Scenario 1: Host vehicle encounters a stopped target vehicle on a straight road
Crash Situation	Host vehicle is traveling at 45 mph or 72 km/h on a straight road in an urban zone with no pedestrians present on a clear day, encounters a stationery target vehicle in its path. The vehicle's FCW system fails to engage and crashes into the target vehicle.
Severity	S3: Life threatening injuries (survival uncertain), fatal injuries
Exposure	E4: High Probability
Controllability	C3: Difficult to control or uncontrollable
Safety Integrity Level	D

Table 11: Risk assessment for system-level hazard H2: Unexpected loss of FCW system for failure operational scenario 1.

Assessment Variable	Description
Hazard	H2: Unexpected loss of FCW system
Failure operational scenario	Scenario 2: Host vehicle encounters a decelerating target vehicle on a straight road.
Crash Situation	Host vehicle is traveling at 45 mph or 72 km/h on a straight road in an urban zone with no pedestrians present on a clear day and encounters a target vehicle travelling at 45 mph or 72 km/h which begins to decelerate. The vehicle's FCW system fails to engage and crashes into the target vehicle.
Severity	S3: Life threatening injuries (survival uncertain), fatal injuries
Exposure	E4: High Probability
Controllability	C3: Difficult to control or uncontrollable
Safety Integrity Level	D

Table 12: Risk assessment for system-level hazard H2: Unexpected loss of FCW system for failure operational scenario 2.

Assessment Variable	Description
Hazard	H2: Unexpected loss of FCW system
Failure operational scenario	Scenario 3: Host vehicle encounters a slower target vehicle on a straight road.
Crash Situation	Host vehicle is traveling at 45 mph or 72 km/h on a straight road in an urban zone with no pedestrians present on a clear day and encounters a target vehicle travelling at 20 mph or 32 km/h. The vehicle's FCW system fails to engage and crashes into the target vehicle.
Severity	S3: Life threatening injuries (survival uncertain), fatal injuries
Exposure	E4: High Probability
Controllability	C3: Difficult to control or uncontrollable
Safety Integrity Level	D

Table 13: Risk assessment for system-level hazard H2: Unexpected loss of FCW system for failure operational scenario 3.

Assessment Variable	Description
Hazard	H3: Unexpected engagement of FCW system
Failure operational scenario	Scenario 1: Host vehicle encounters a stopped target vehicle on a straight road
Crash Situation	Host vehicle is traveling at 45 mph or 72 km/h on a straight road in an urban zone with no pedestrians present on a clear day, encounters a stationery target vehicle in its path. The vehicle's FCW system engages when it should not but avoids crashing into the target vehicle.
Severity	S1: Light and moderate injuries such as whiplash
Exposure	E4: High Probability
Controllability	C2: Normally controllable
Safety Integrity Level	A

Table 14: Risk assessment for system-level hazard H3: Unexpected engagement of FCW system for failure operational scenario 1.

Assessment Variable	Description
Hazard	H3: Unexpected engagement of FCW system
Failure operational scenario	Scenario 2: Host vehicle encounters a decelerating target vehicle on a straight road.
Crash Situation	Host vehicle is traveling at 45 mph or 72 km/h on a straight road in an urban zone with no pedestrians present on a clear day and encounters a target vehicle travelling at 45 mph or 72 km/h which begins to decelerate. The vehicle's FCW system engages when it should not but avoids crashing into the target vehicle.
Severity	S1: Light and moderate injuries such as whiplash
Exposure	E4: High Probability
Controllability	C2: Normally controllable
Safety Integrity Level	A

Table 15: Risk assessment for system-level hazard H3: Unexpected engagement of FCW system for failure operational scenario 2.

Assessment Variable	Description
Hazard	H3: Unexpected engagement of FCW system
Failure operational scenario	Scenario 3: Host vehicle encounters a slower target vehicle on a straight road.
Crash Situation	Host vehicle is traveling at 45 mph or 72 km/h on a straight road in an urban zone with no pedestrians present on a clear day and encounters a target vehicle travelling at 20 mph or 32 km/h. The vehicle's FCW system engages when it should not but avoids crashing into the target vehicle.
Severity	S1: Light and moderate injuries such as whiplash
Exposure	E4: High Probability
Controllability	C2: Normally controllable
Safety Integrity Level	A

Table 16: Risk assessment for system-level hazard H3: Unexpected engagement of FCW system for failure operational scenario 3.

Appendix B

Failure modes of FCW system

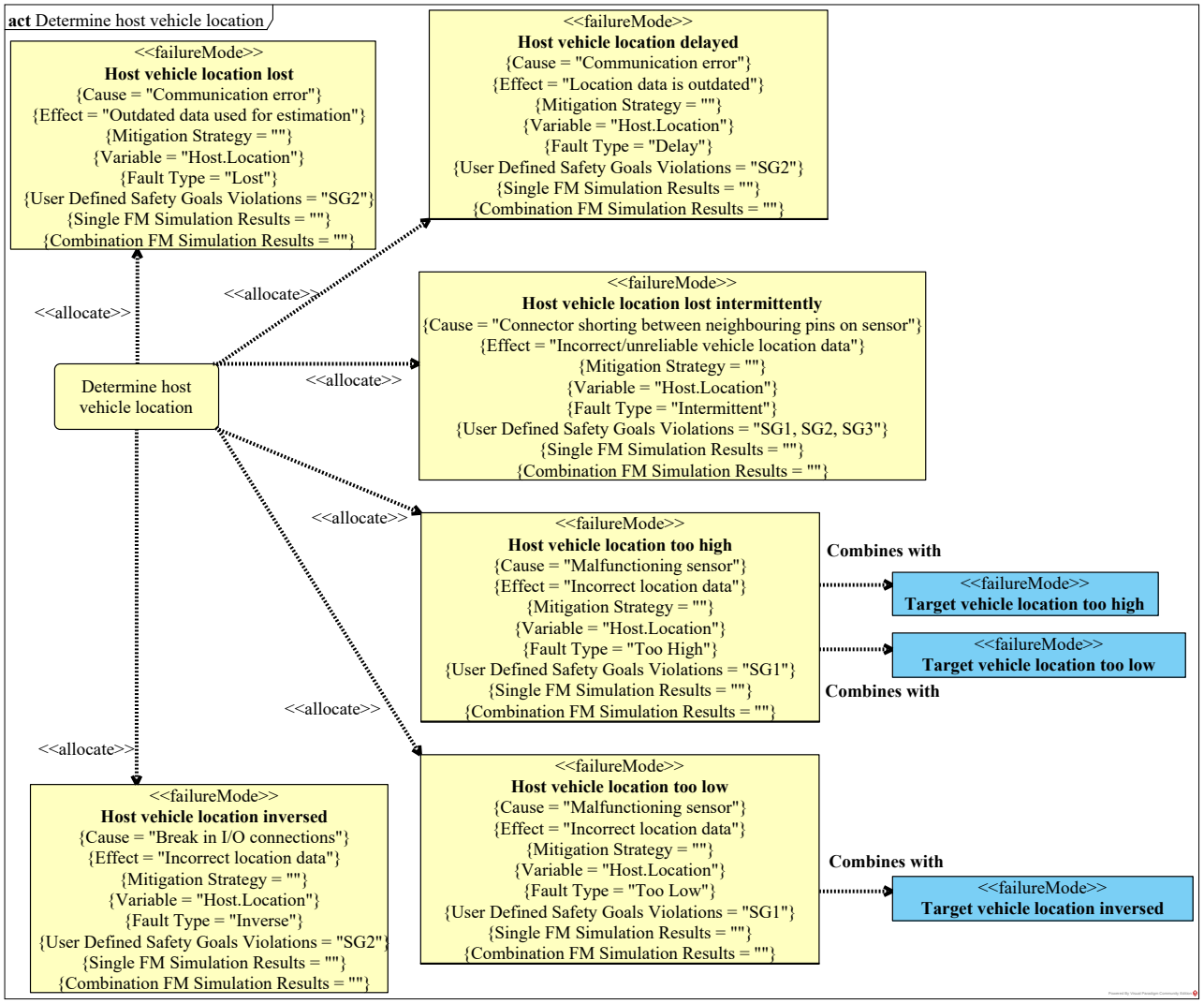


Figure 51: Failure modes for the *Determine host vehicle location* function

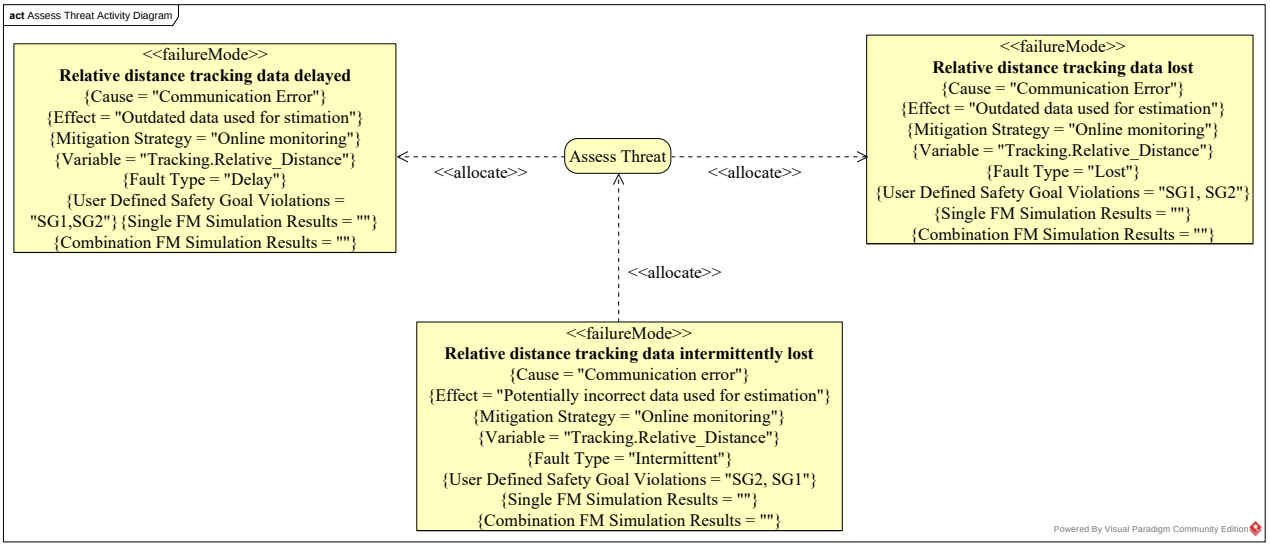


Figure 52: Failure modes for the *Assess threat* function

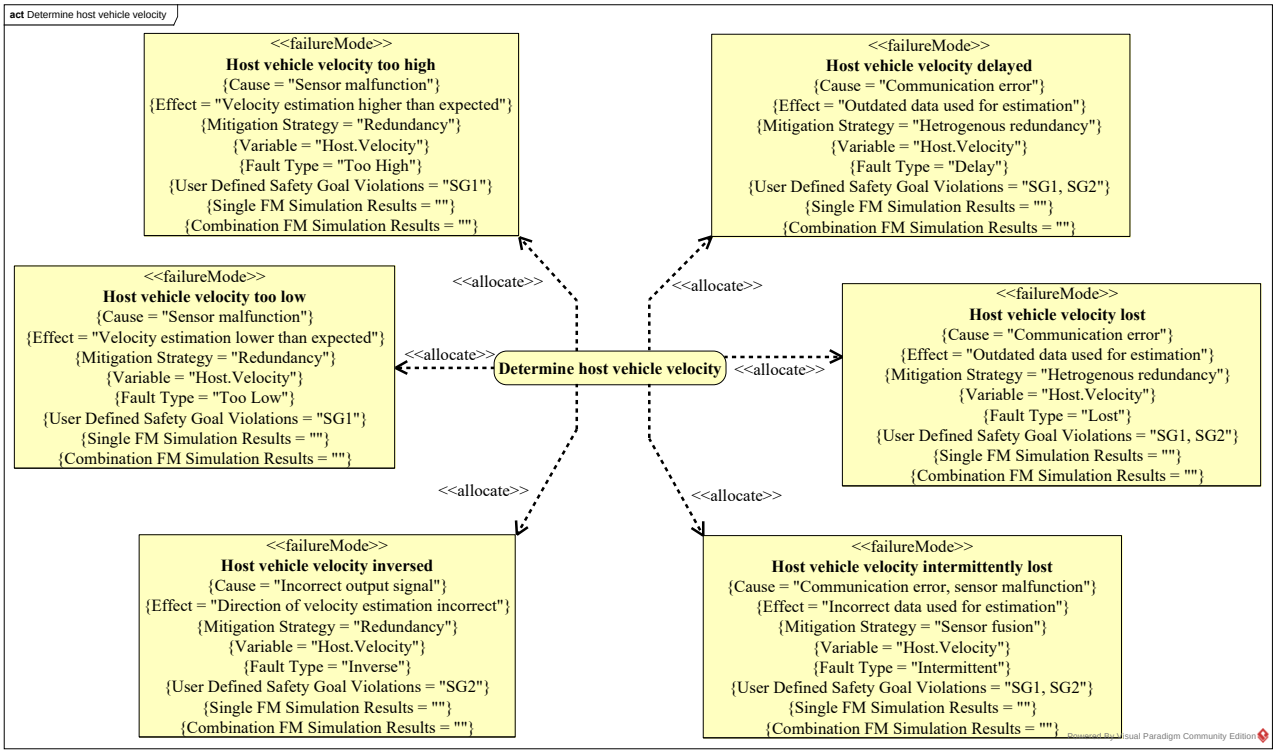


Figure 53: Failure modes for the *Determine host vehicle velocity* function

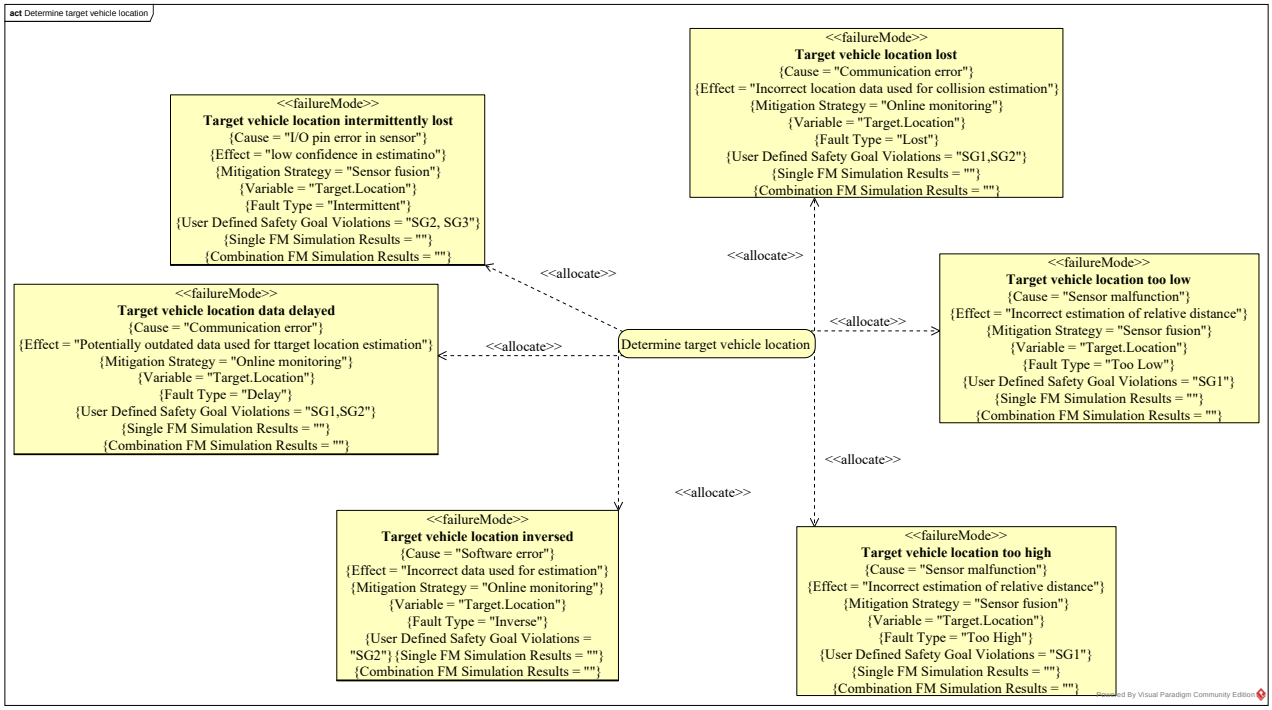


Figure 54: Failure modes for the *Determine target vehicle location* function

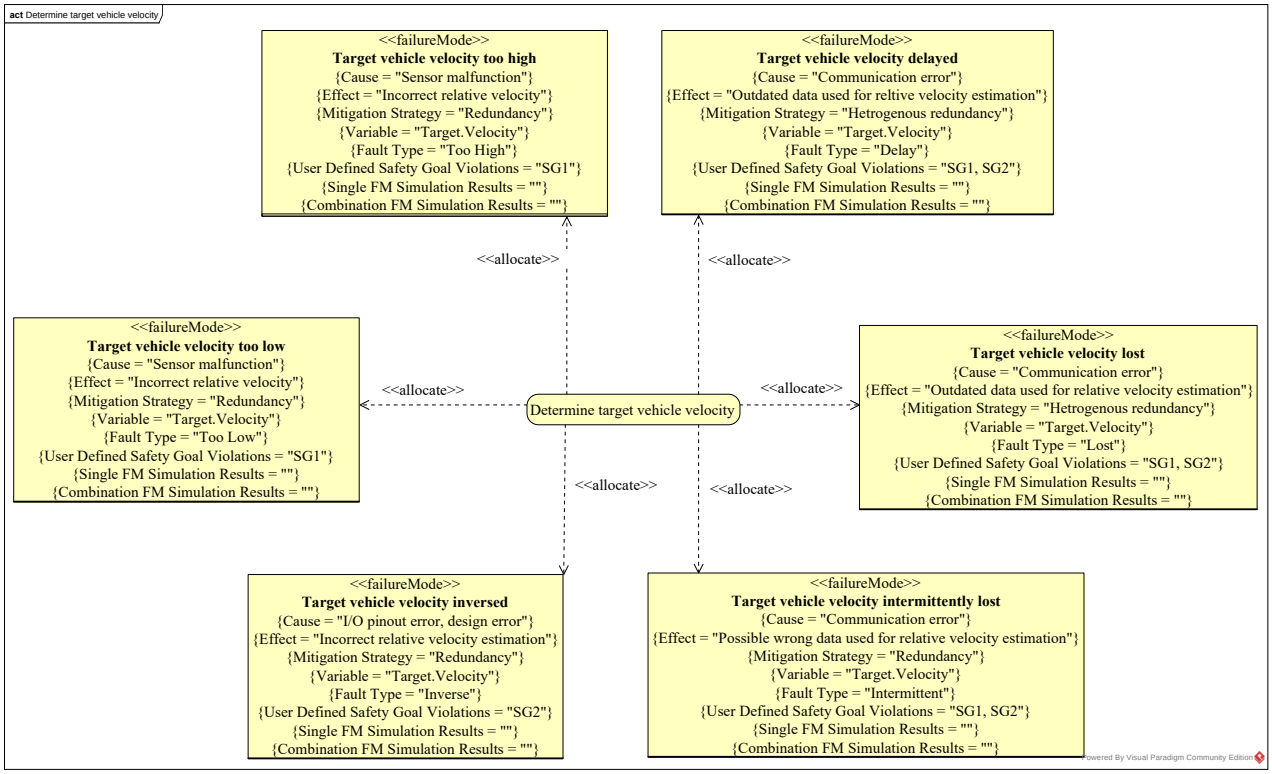


Figure 55: Failure modes for the *Determine target vehicle velocity* function

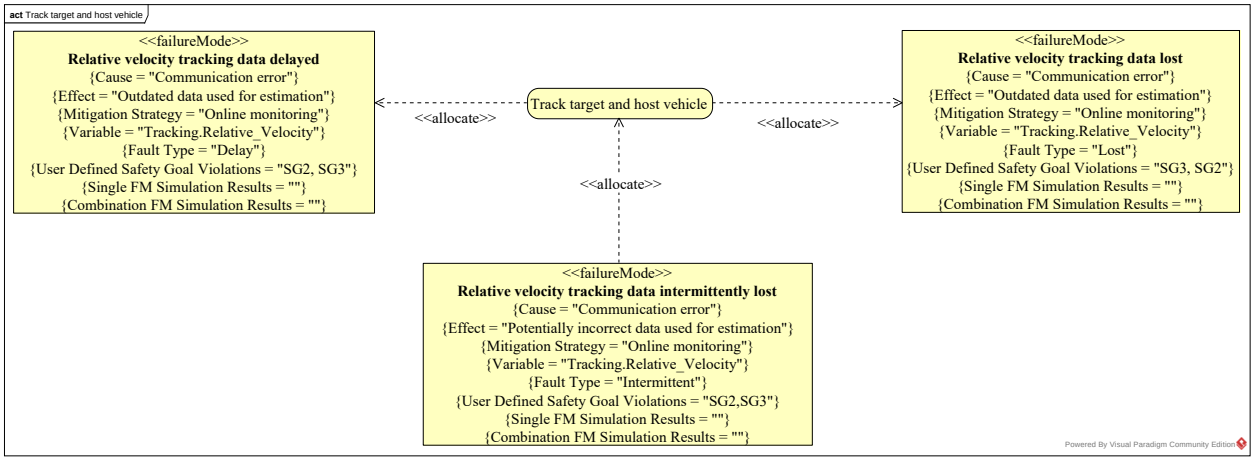


Figure 56: Failure modes for the *Track target and host vehicle* function

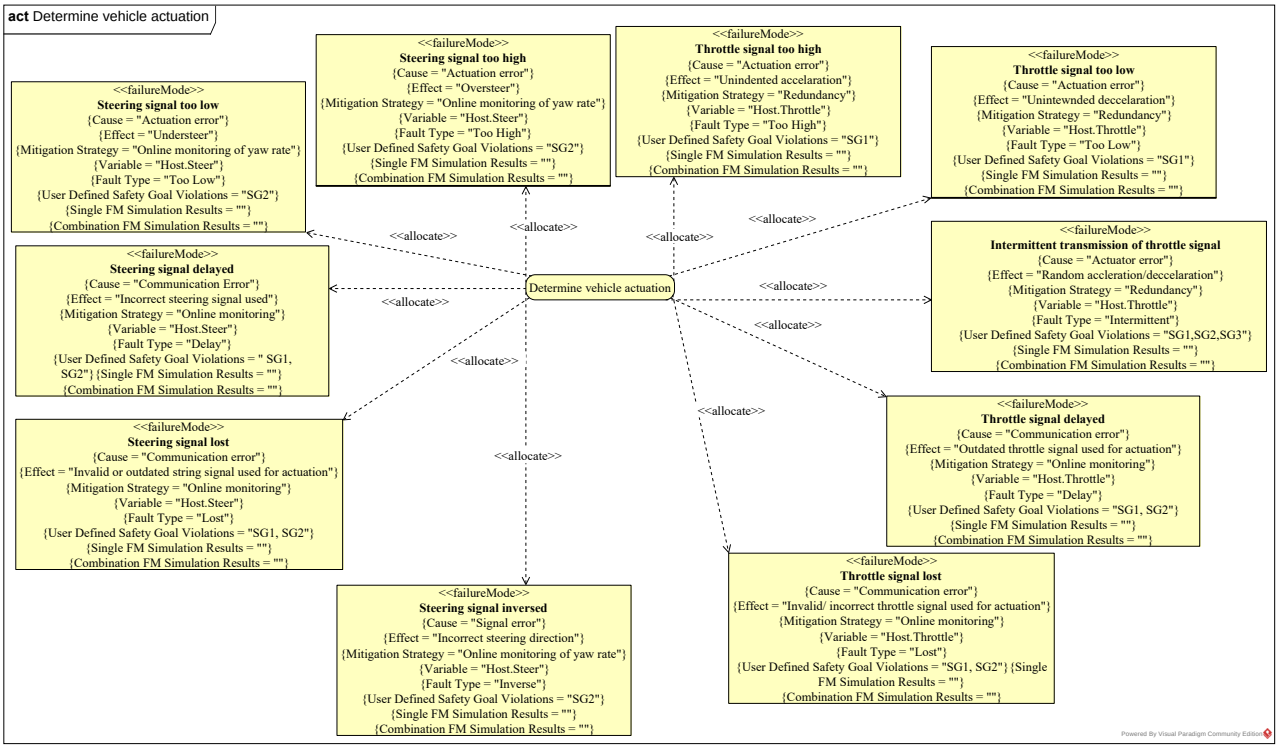


Figure 57: Failure modes for the *Determine host vehicle actuation* function

Appendix C
FMEA for FCW system

The FMEA tables shown in this appendix represent image exports of sections of the FMEA table. For ease of readability, the FMEA table is shown in an image format, with the FMEA for each function illustrated in a separate figure.

Block	Function	Failure Mode	Cause	User-Defined SG violation	Risk	Effect	Mitigation Strategy	Simulation SG violation
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location too high	Malfunctioning sensor	SG1	D	Incorrect location data		
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location too low	Malfunctioning sensor	SG1	D	Incorrect location data		
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location lost intermittently	Connector shorting between neighbouring pins on sensor	SG1,SG2,SG3	D	Incorrect/unreliable vehicle location data		
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location lost	Communication error	SG2	D	Outdated data used for estimation		
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location delayed	Communication error	SG1,SG2	D	Location data is outdated		
Vehicle Dynamics Sensors	Determine host vehicle location	Host vehicle location inversed	Break in I/O connections	SG2	D	Incorrect location data		

Figure 58: FMEA for the *Determine host vehicle location* function

Block	Function	Failure Mode	Cause	User-Defined SG violation	Risk	Effect	Mitigation Strategy	Simulation SG violation
Vehicle Dynamics Sensors	Determine host vehicle velocity	Host vehicle velocity too high	Sensor malfunction	SG1	D	Velocity estimation higher than expected	Redundancy	
Vehicle Dynamics Sensors	Determine host vehicle velocity	Host vehicle velocity too low	Sensor malfunction	SG1	D	Velocity estimation lower than expected	Redundancy	
Vehicle Dynamics Sensors	Determine host vehicle velocity	Host vehicle velocity inversed	Incorrect output signal	SG2	D	Direction of velocity estimation incorrect	Redundancy	
Vehicle Dynamics Sensors	Determine host vehicle velocity	Host vehicle velocity delayed	Communication error	SG1,SG2	D	Outdated data used for estimation	Hetrogenous redundancy	
Vehicle Dynamics Sensors	Determine host vehicle velocity	Host vehicle velocity lost	Communication error	SG1,SG2	D	Outdated data used for estimation	Hetrogenous redundancy	
Vehicle Dynamics Sensors	Determine host vehicle velocity	Host vehicle velocity intermittently lost	Communication error, sensor malfunction	SG1,SG2	D	Incorrect data used for estimation	Sensor fusion	

Figure 59: FMEA for the *Determine host vehicle velocity* function

Block	Function	Failure Mode	Cause	User-Defined SG violation	Risk	Effect	Mitigation Strategy	Simulation SG violation
Target Vehicle Tracking module	Determine target vehicle location	Target vehicle location too high	Sensor malfunction	SG1	D	Incorrect estimation of relative distance	Sensor fusion	
Target Vehicle Tracking module	Determine target vehicle location	Target vehicle location too low	Sensor malfunction	SG1	D	Incorrect estimation of relative distance	Sensor fusion	
Target Vehicle Tracking module	Determine target vehicle location	Target vehicle location lost	Communication error	SG1,SG2	D	Incorrect location data used for collision estimation	Online monitoring	
Target Vehicle Tracking module	Determine target vehicle location	Target vehicle location data delayed	Communication error	SG1,SG2	D	Potentially outdated data used for ttarget location estimation	Online monitoring	
Target Vehicle Tracking module	Determine target vehicle location	Target vehicle location intermittently lost	I/O pin error in sensor	SG2,SG3	D	low confidence in estimatino	Sensor fusion	
Target Vehicle Tracking module	Determine target vehicle location	Target vehicle location inversed	Software error	SG2	D	Incorrect data used for estimation	Online monitoring	

Figure 60: FMEA for the *Determine target vehicle location* function

Block	Function	Failure Mode	Cause	User-Defined SG violation	Risk	Effect	Mitigation Strategy	Simulation SG violation
Target Vehicle Tracking module	Determine target vehicle velocity	Target vehicle velocity too high	Sensor malfunction	SG1	D	Incorrect relative velocity	Redundancy	
Target Vehicle Tracking module	Determine target vehicle velocity	Target vehicle velocity too low	Sensor malfunction	SG1	D	Incorrect relative velocity	Redundancy	
Target Vehicle Tracking module	Determine target vehicle velocity	Target vehicle velocity inversed	I/O pinout error, design error	SG2	D	Incorrect relative velocity estimation	Redundancy	
Target Vehicle Tracking module	Determine target vehicle velocity	Target vehicle velocity delayed	Communication error	SG1,SG2	D	Outdated data used for reltive velocity estimation	Hetrogenous redundancy	
Target Vehicle Tracking module	Determine target vehicle velocity	Target vehicle velocity lost	Communication error	SG1,SG2	D	Outdated data used for relative velocity estimation	Hetrogenous redundancy	
Target Vehicle Tracking module	Determine target vehicle velocity	Target vehicle velocity intermittently lost	Communication error	SG1,SG2	D	Possible wrong data used for relative velocity estimation	Redundancy	

Figure 61: FMEA for the *Determine target vehicle velocity* function

Block	Function	Failure Mode	Cause	User-Defined SG violation	Risk	Effect	Mitigation Strategy	Simulation SG violation
Threat Assessment module	Assess Threat	Relative distance tracking data delayed	Communication Error	SG1,SG2	D	Outdated data used for stimation	Online monitoring	
Threat Assessment module	Assess Threat	Relative distance tracking data lost	Communication Error	SG1,SG2	D	Outdated data used for estimation	Online monitoring	
Threat Assessment module	Assess Threat	Relative distance tracking data intermittently lost	Communication error	SG1,SG2	D	Potentially incorrect data used for estimation	Online monitoring	
Threat Assessment module	Track target and host vehicle	Relative velocity tracking data delayed	Communication error	SG1,SG2	D	Outdated data used for estimation	Online monitoring	
Threat Assessment module	Track target and host vehicle	Relative velocity tracking data lost	Communication error	SG1,SG2	D	Outdated data used for estimation	Online monitoring	
Threat Assessment module	Track target and host vehicle	Relative velocity tracking data intermittently lost	Communication error	SG1,SG2	D	Potentially incorrect data used for estimation	Online monitoring	

Figure 62: FMEA for the *Assess Threat* and*Track target and host vehicle* function

Block	Function	Failure Mode	Cause	User-Defined SG violation	Risk	Effect	Mitigation Strategy	Simulation SG violation
Vehicle Controller	Determine vehicle actuation	Steering signal too high	Actuation error	SG2	D	Oversteer	Online monitoring of yaw rate	
Vehicle Controller	Determine vehicle actuation	Steering signal too low	Actuation error	SG2	D	Understeer	Online monitoring of yaw rate	
Vehicle Controller	Determine vehicle actuation	Steering signal inversed	Signal error	SG2	D	Incorrect steering direction	Online monitoring of yaw rate	
Vehicle Controller	Determine vehicle actuation	Throttle signal too high	Actuation error	SG1	D	Unindented accelaration	Redundancy	
Vehicle Controller	Determine vehicle actuation	Throttle signal too low	Actuation error	SG1	D	Unintewnded deccelaration	Redundancy	
Vehicle Controller	Determine vehicle actuation	Intermittent transmission of throttle signal	Actuator error	SG1,SG2,SG3	D	Random accleration/deccelaration	Redundancy	
Vehicle Controller	Determine vehicle actuation	Steering signal delayed	Communication Error	SG1,SG2	D	Incorrect steering signal used	Online monitoring	
Vehicle Controller	Determine vehicle actuation	Steering signal lost	Communication error	SG1,SG2	D	Invalid or outdated string signal used for actuation	Online monitoring	
Vehicle Controller	Determine vehicle actuation	Throttle signal delayed	Communication error	SG1,SG2	D	Outdated throttle signal used for actuation	Online monitoring	
Vehicle Controller	Determine vehicle actuation	Throttle signal lost	Communication error	SG1,SG2	D	Invalid/ incorrect throttle signal used for actuation	Online monitoring	

Figure 63: FMEA for the *Determine vehicle actuation* function

Appendix D

Fault trees for safety goal SG2, SG3

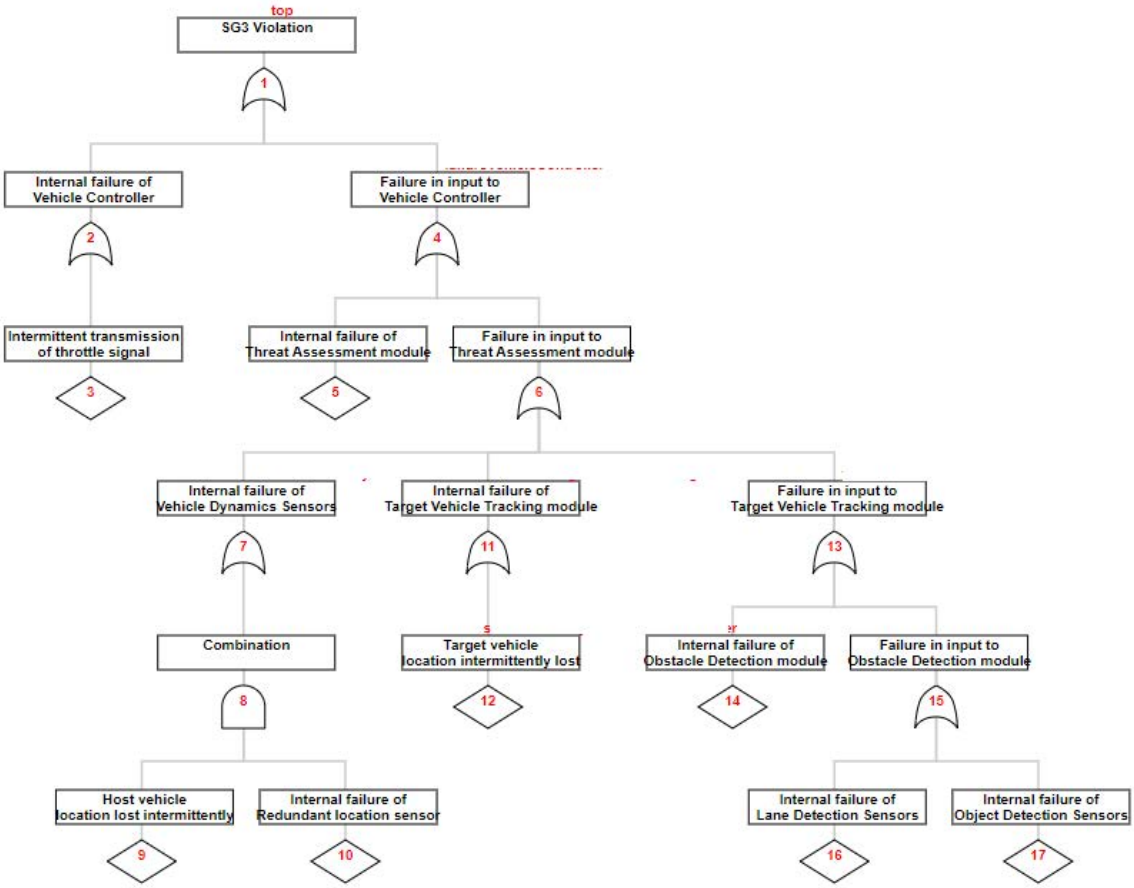
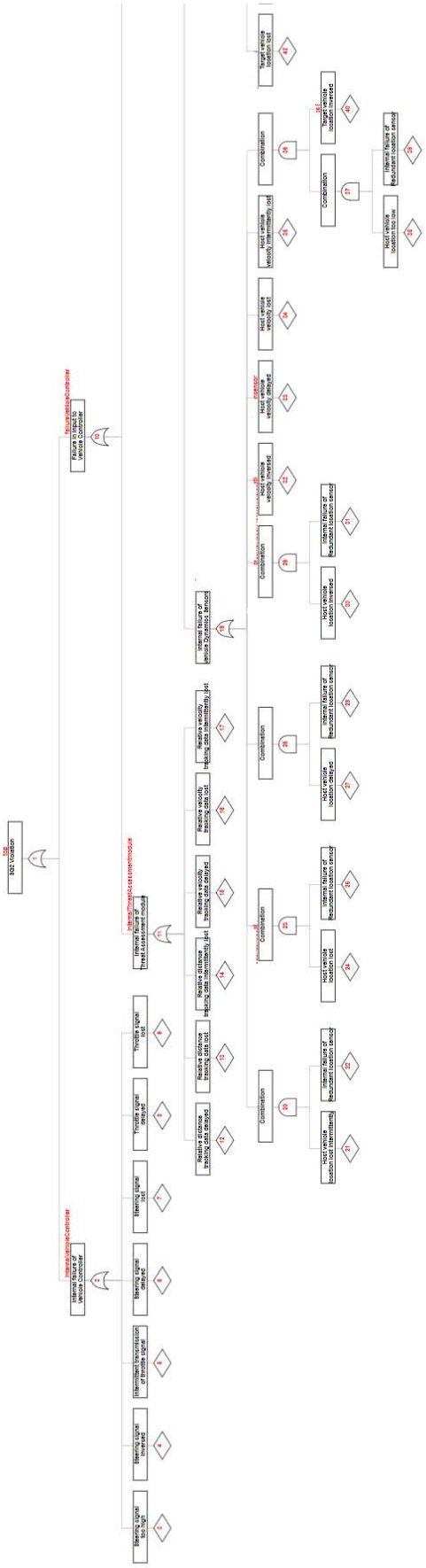
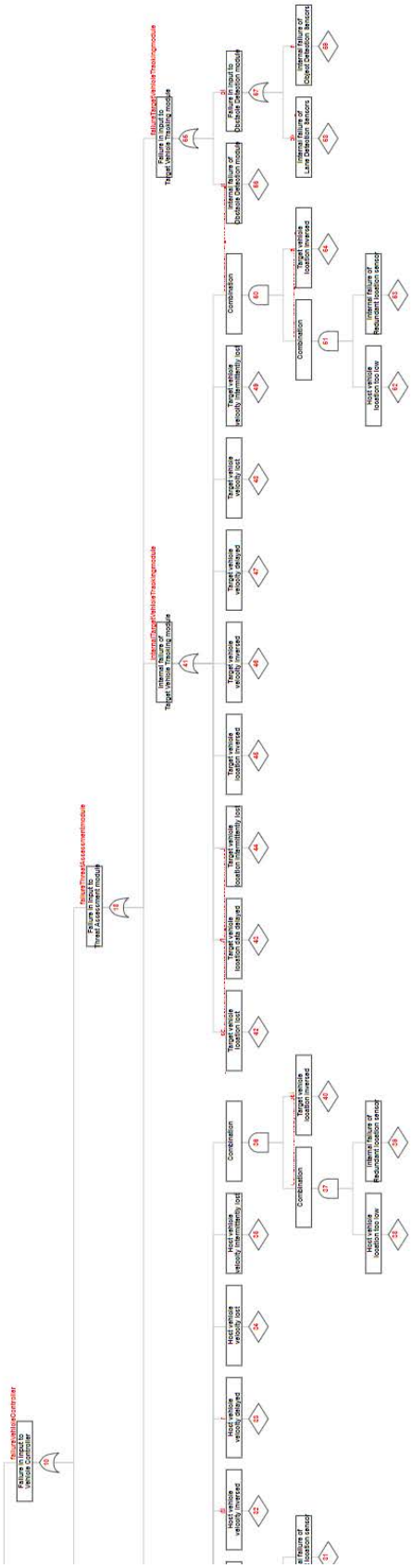


Figure 64: Fault tree for violation of safety goal SG3: Prevent unexpected engagement of FCW system



(a) Right side of the fault tree for violation of SG1



(b) Left side of the fault tree for violation of SG1

Figure 65: Fault tree for violation of safety goal SG2: Prevent unexpected loss of FCW system

Appendix E

Updated system architecture of FCW system

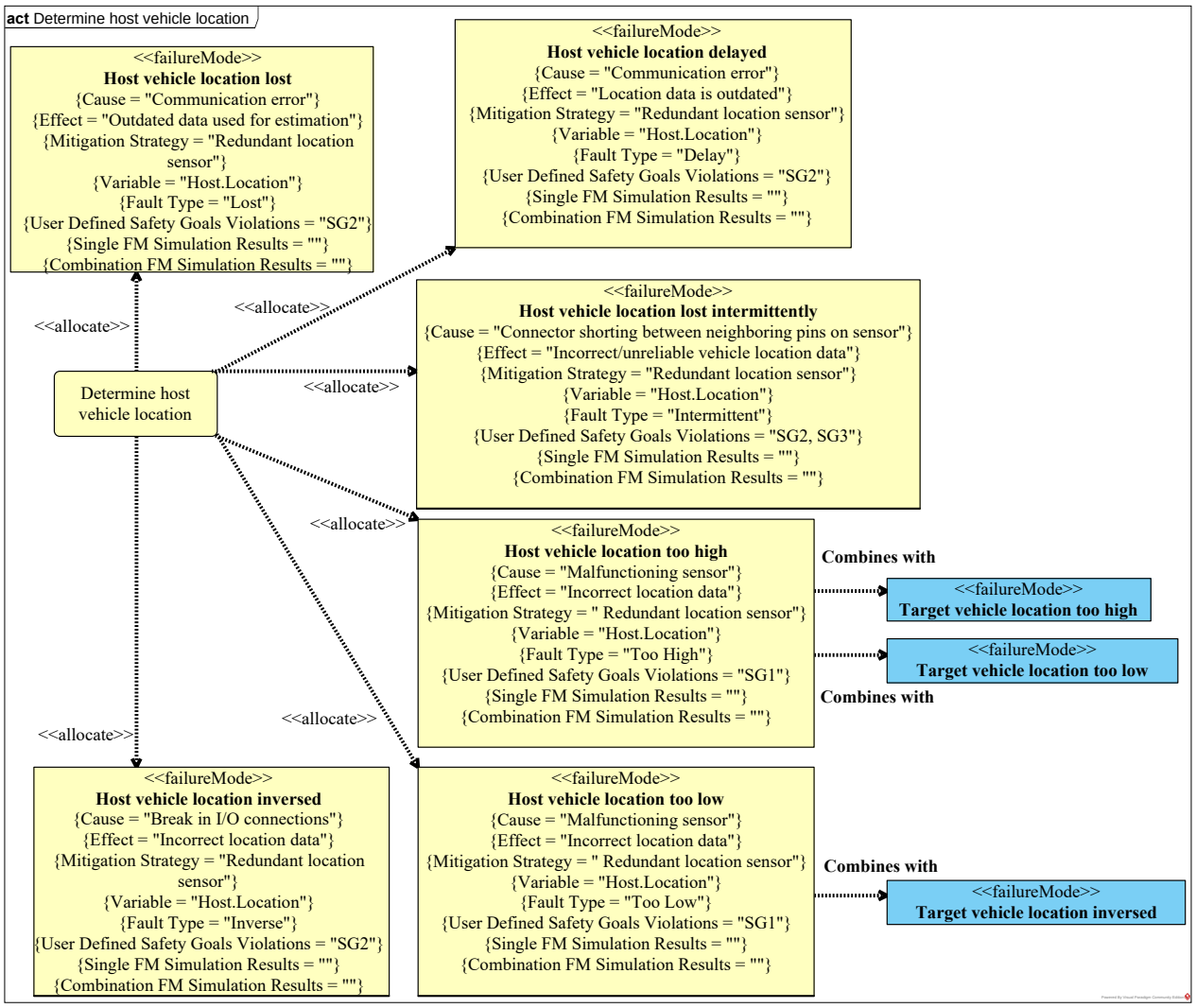


Figure 66: Updates made to the mitigation strategy and safety goal violation of the "Determine host vehicle location" function's failure modes

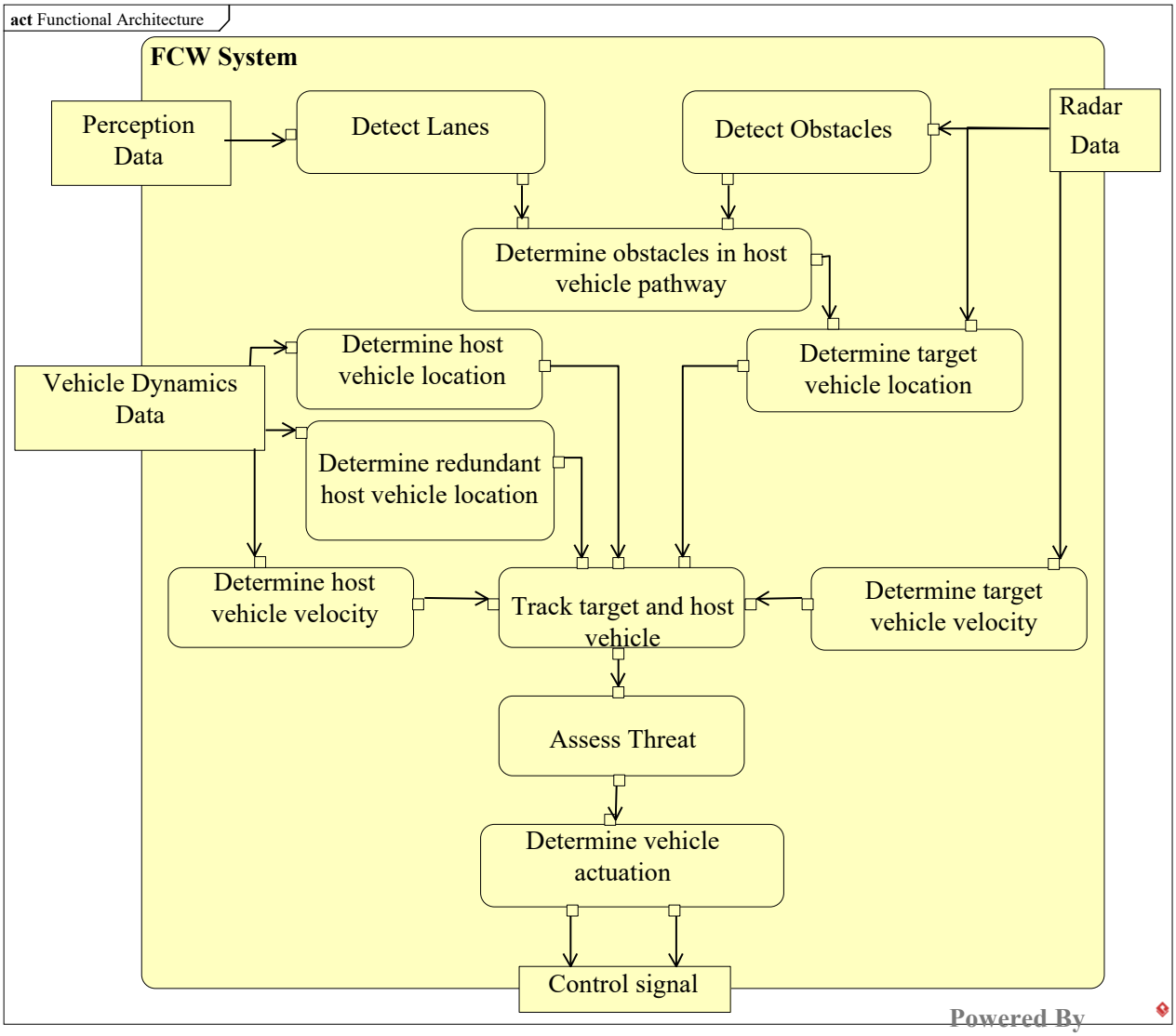


Figure 67: Updates made to the functional architecture of the FCW system to include the *Determine host vehicle location* function

Figure 68: Updates made to the logical architecture (BDD) of the FCW system to include the redundant location sensor

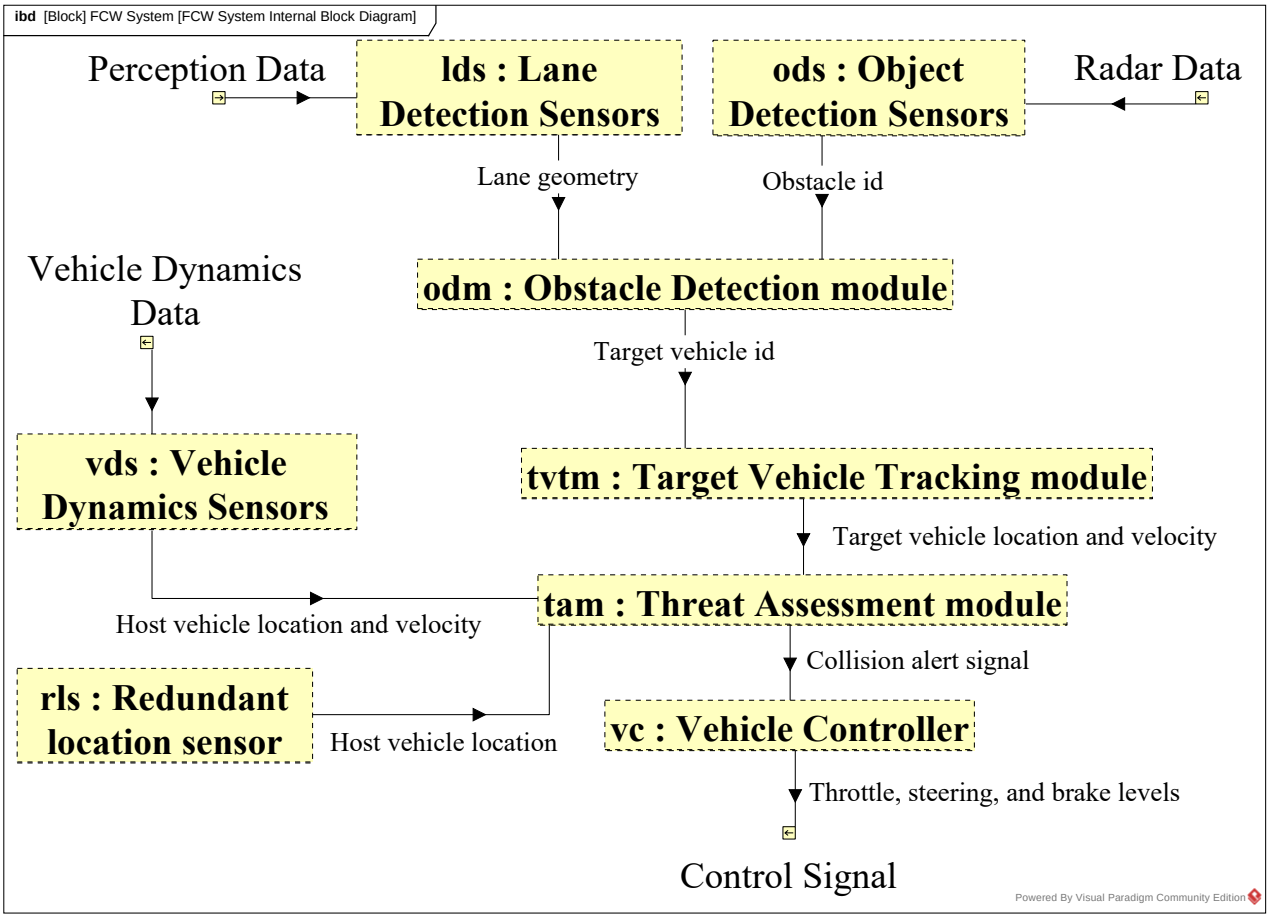


Figure 69: Updates made to the logical architecture (IBD) of the FCW system to include the redundant location sensor

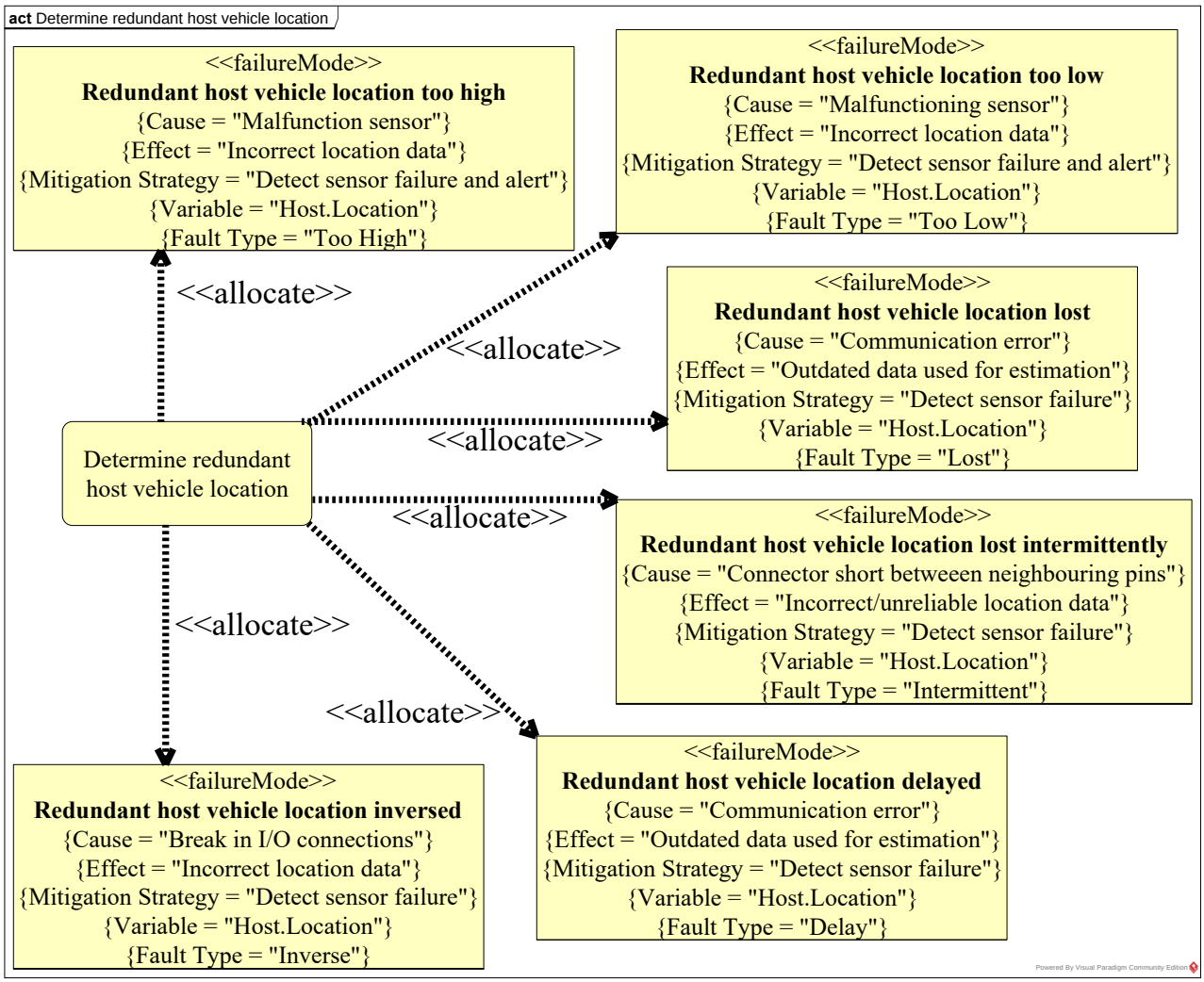


Figure 70: Failure modes of the *Determine redundant host vehicle location* function

Bibliography

[1] ISO 26262: Road vehicles-Functional safety-Part 3: Concept Phase, 2018.

[2] National ITS Architecture Team. Systems engineering for intelligent transportation systems. Technical report, Department of Transportation, Office of Operations, Jan 2007.

[3] Matthew Hause et al. The sysml modelling language. In *Fifteenth European Systems Engineering Conference*, volume 9, pages 1–12, 2006.

[4] Nigel Cross. *Engineering design methods: strategies for product design*. John Wiley & Sons, 2021.

[5] M.L. Cummings and David Britton. Regulating safety-critical autonomous systems: past, present, and future perspectives. In *Living with Robots*, pages 119–140. Elsevier, jan 2020.

[6] Irem Tumer and Carol Smidts. Integrated design-stage failure analysis of software-driven hardware systems. *IEEE Transactions on Computers*, 60(8):1072–1084, aug 2011.

[7] Victor Bolbot, Gerasimos Theotokatos, Luminita Manuela Bujorianu, Evangelos Boulougouris, and Dracos Vassalos. Vulnerabilities and safety assurance methods in cyber-physical systems: A comprehensive review. *Reliability Engineering & System Safety*, 182:179–193, 2019.

[8] John A Mcdermid. Complexity: Concept, Causes and Control. In *Proceedings Sixth IEEE International Conference on Engineering of Complex Computer Systems. ICECCS 2000*, pages 2—-9, 2000.

[9] Kaushik Sinha. *Structural complexity and its implications for design of cyber-physical systems*. PhD thesis, Massachusetts Institute of Technology, 2014.

[10] Zaid Tahir and Rob Alexander. Coverage based testing for VV and Safety Assurance of Self-driving Autonomous Vehicles: A Systematic Literature Review. In *Proceedings - 2020 IEEE International Conference on Artificial Intelligence Testing, AITest 2020*, pages 23–30. Keble College. , GBR. (In Press, 2020.

[11] Jérémie Guiochet, Mathilde Machin, and Hélène Waeselynck. Safety-critical advanced robots: A survey. *Robotics and Autonomous Systems*, 94:43–52, aug 2017.

[12] ISO 26262:Road vehicles-Functional safety, 2018.

[13] Clifton A. Ericson. *Hazard Analysis Techniques for System Safety*. John Wiley & Sons, 2005.

[14] Nicholas J Bahr. *System safety engineering and risk assessment: a practical approach*. CRC press, 2018.

[15] Marco Bozzano and Adolfo Villafiorita. *Design and Safety Assessment of Critical Systems*. Auerbach Publications, Boston, MA, USA, 1st edition, 2010.

[16] Mohammad Modarres. *Risk analysis in engineering: techniques, tools, and trends*. CRC press, 2016.

[17] Margaret V. Stringfellow, Nancy G. Leveson, and Brandon D. Owens. Safety-driven design for software-intensive aerospace and automotive systems. *Proceedings of the IEEE*, 98(4):515–525, 2010.

[18] Barry Boehm, Ricardo Valerdi, and Eric Honour. The ROI of systems engineering: Some quantitative results for software-intensive systems. *Systems Engineering*, 11(3):221–234, jun 2008.

[19] Derek K Hitchins. Systems engineering: in search of the elusive optimum. *Engineering Management Journal*, 8(4):195–207, 1998.

[20] Faïda Mhenni, Nga Nguyen, and Jean Yves Choley. SafeSysE: A Safety Analysis Integration in Systems Engineering Approach. *IEEE Systems Journal*, 12(1):161–172, mar 2018.

[21] Oleg Lisagor, Tim Kelly, and Ru Niu. Model-based safety assessment: Review of the discipline and its challenges. In *ICRMS'2011 - Safety First, Reliability Primary: Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*, pages 625–632. IEEE, jun 2011.

[22] Anthony Legendre, Agnes Lanusse, and Antoine Rauzy. Toward model synchronization between safety analysis and system architecture design in industrial contexts. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10437 LNCS, pages 35–49. Springer, 2017.

[23] Tatiana Prosvirnova, Michel Batteux, Pierre-Antoine Brameret, Abraham Cherfi, Thomas Friedlhuber, Jean-Marc Roussel, and Antoine Rauzy. The AltaRica 3.0 Project for Model-Based Safety Assessment. *IFAC Proceedings Volumes*, 46(22):127–132, 2013.

[24] Yan Li, Qi Gong, and Duo Su. Model-based system safety assessment of aircraft power plant. In *Procedia Engineering*, volume 80, pages 85–92. Elsevier, jan 2014.

[25] Oliver Sträter. *Cognition and Safety*. Routledge, dec 2016.

[26] Geoffrey Biggs, Tomas Juknevicius, Andrius Armonas, and Kyle Post. Integrating Safety and Reliability Analysis into MBSE: overview of the new proposed OMG standard. *INCOSE International Symposium*, 28(1):1322–1336, 2018.

[27] Anjali Joshi, Steven P. Miller, Michael Whalen, and Mats P.E. Heimdahl. A proposal for model-based safety analysis. In *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, volume 2, 2005.

[28] Hongli Wang, Deming Zhong, Tingdi Zhao, and Fuchun Ren. Integrating Model Checking with SysML in Complex System Safety Analysis. *IEEE Access*, 7:16561–16571, 2019.

[29] Nancy G Leveson. *Engineering a safer world: Systems thinking applied to safety*. The MIT Press, 2016.

[30] Charles Perrow. *Normal accidents*. Princeton university press, 2011.

[31] Andrew Rollings and Ernest Adams. *Andrew Rollings and Ernest Adams on game design*. New Riders, 2003.

[32] Jamie P Monat and Thomas F Gannon. What is systems thinking? a review of selected literature plus recommendations. *American Journal of Systems Science*, 4(1):11–26, 2015.

[33] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[34] Alberto Broggi, Michele Buzzoni, Stefano Debattisti, Paolo Grisleri, Maria Chiara Laghi, Paolo Medici, and Pietro Versari. Extensive tests of autonomous driving technologies. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1403–1415, 2013.

[35] WuLing Huang, Kunfeng Wang, Yisheng Lv, and FengHua Zhu. Autonomous vehicles testing methods review. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 163–168. IEEE, 2016.

[36] Philip Koopman and Michael Wagner. Toward a Framework for Highly Automated Vehicle Safety Validation. In *SAE Technical Papers*, volume 2018-April, 2018.

[37] National Highway Traffic Safety Administration. Forward collision warning system confirmation test. Technical report, Office of Vehicle Safety, Office of Crash Avoidance Standards, National Highway Traffic Safety Administration, Washington, DC, 2013.

[38] Philip Koopman and Michael Wagner. Challenges in Autonomous Vehicle Testing and Validation. *SAE International Journal of Transportation Safety*, 4(1):15–24, 2016.

[39] Philip Koopman and Michael Wagner. Autonomous Vehicle Safety: An Interdisciplinary Challenge. *IEEE Intelligent Transportation Systems Magazine*, 9(1):90–96, mar 2017.

[40] Nidhi Kalra and Susan M Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 2016.

[41] N. Yakymets, M. Sango, S. Dhouib, and R. Gelin. Model-Based Engineering, Safety Analysis and Risk Assessment for Personal Care Robots. In *IEEE International Conference on Intelligent Robots and Systems*, pages 6136–6141. IEEE, oct 2018.

[42] Garazi Juez Uriagereka, Estibaliz Amparan, Cristina Martinez Martinez, Jabier Martinez, Aurelien Ibanez, Matteo Morelli, Ansgar Radermacher, and Huascar Espinoza. Design-time safety assessment of robotic systems using fault injection simulation in a model-driven approach. In *Proceedings - 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C 2019*, pages 577–586. Institute of Electrical and Electronics Engineers Inc., sep 2019.

[43] Azad M. Madni and Michael Sievers. Model-based systems engineering: Motivation, current status, and research opportunities. *Systems Engineering*, 21(3):172–190, 2018.

[44] Joseph D'Ambrosio and Grant Soremekun. Systems engineering challenges and MBSE opportunities for automotive system design. In *2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017*, volume 2017-Janua, pages 2075–2080. Institute of Electrical and Electronics Engineers Inc., nov 2017.

[45] Edward R Carroll and Robert J Malins. Systematic literature review: How is model-based systems engineering justified? *Sandia National Laboratories*, 2016.

[46] Mark Chodas. Improving the Design Process of the REgolith X-Ray Imaging Spectrometer with Model Based Systems Engineering, 2014.

[47] Anjali Joshi and Mats P.E. Heimdahl. Model-based safety analysis of simulink models using SCADE design verifier. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3688 LNCS, pages 122–135, 2005.

[48] Marco Bozzano and Adolfo Villafiorita. The FSAP/NuSMV-SA safety analysis platform. *International Journal on Software Tools for Technology Transfer*, 9(1):5–24, feb 2007.

[49] Haifeng Wang, Shuo Liu, and Chunhai Gao. Study on model-based safety verification of automatic train protection system. In *PACIIA 2009 - 2009 2nd Asia-Pacific Conference on Computational Intelligence and Industrial Applications*, volume 1, pages 467–470, 2009.

[50] Jeff A Estefan. Survey of Model-Based Systems Engineering (MBSE) Methodologies. *Jet Propulsion*, 25:1–70, 2008.

[51] INCOSE. MBSE Wiki: Methodology and Metrics. https://www.omgwiki.org/MBSE/doku.php?id=mbse:methodology, 2020.

[52] Aurelijus Morkevicius, Aiste Aleksandraviciene, Andrius Armonas, and Gauthier Fanmuy. Towards a common systems engineering methodology to cover a complete system development process. In *INCOSE International Symposium*, volume 30, Issue 1, pages 138–152. Wiley Online Library, 2020.

[53] Howard Lykins, Sanford Friedenthal, and Abraham Meilich. 4.4. 4 adapting uml for an object oriented systems engineering method (oosem). In *INCOSE International Symposium*, volume 10, Issue 1, pages 490–497. Wiley Online Library, 2000.

[54] Daniel Krob. Cesam: Cesames systems architecting method-a pocket guide, 2017.

[55] Hans-Peter Hoffmann. Sysml-based systems engineering using a model-driven development approach. *White Paper, Telelogic*, 2008.

[56] Donatas Mažeika. *Model-based systems engineering method for creating secure systems*. PhD thesis, Kauno technologijos universitetas, 2021.

[57] Tim Weilkiens. *Systems engineering with SysML/UML: modeling, analysis, design*. Elsevier, 2011.

[58] David A Wagner, Matthew B Bennett, Robert Karban, Nicolas Rouquette, Steven Jenkins, and Michel Ingham. An ontology for state analysis: Formalizing the mapping to sysml. In *2012 IEEE Aerospace Conference*, pages 1–16. IEEE, 2012.

[59] Susan Rose Childers and James E Long. A concurrent methodology for the system engineering design process. In *INCOSE International Symposium*, volume 4, Issue 1, pages 226–231. Wiley Online Library, 1994.

[60] James E Long. Mbse in practice: Developing systems with core. *Vitech briefing slides, Vitech Corporation, Vienna, VA*, 2007.

[61] SEBoK. Modeling Standards - SEBoK, 2018.

[62] Robin Cressent, Pierre David, and Vincent Idasiak. Increasing Reliability of Embedded Systems in a SysML Centered MBSE Process: Application to LEA Project. *1st Workshop on Model Based Engineering for Embedded Systems Design*, pages 1–6, mar 2010.

[63] Ana Luísa Ramos, José Vasconcelos Ferreira, and Jaume Barceló. Model-based systems engineering: An emerging approach for modern systems. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 42(1):101–111, jan 2012.

[64] Razieh Behjati, Tao Yue, Shiva Nejati, Lionel Briand, and Bran Selic. Extending SysML with AADL concepts for comprehensive system architecture modeling. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6698 LNCS, pages 236–252. Springer, Berlin, Heidelberg, 2011.

[65] OMG. OMG Systems Modeling Language (OMG SysML ™) v.1.2. http://www.omgsysml.org/ http://www.omg.org/spec/SysML/1.2/PDF/, June 2010.

[66] Jon Holt and Simon Perry. *SysML for systems engineering*, volume 7. IET, 2008.

[67] IEC. IEC 61508 Functional Safety, 2010.

[68] Takuto Ishimatsu, Nancy Leveson, John Thomas, Masa Katahira, Yuko Miyamoto, and Haruka Nakao. Modeling and hazard analysis using STPA. *European Space Agency, (Special Publication) ESA SP*, 680 SP(1), 2010.

[69] Cody Harrison Fleming, Melissa Spencer, John Thomas, Nancy Leveson, and Chris Wilkinson. Safety assurance in NextGen and complex transportation systems. *Safety Science*, 55:173–187, jun 2013.

[70] Sardar Muhammad Sulaman, Armin Beer, Michael Felderer, and Martin Höst. Comparison of the FMEA and STPA safety analysis methods–a case study. *Software Quality Journal*, 27(1):349–387, mar 2019.

[71] Septavera Sharvia and Yiannis Papadopoulos. Integrated application of compositional and behavioural safety analysis. In *Dependable Computer Systems*, pages 179–192. Springer, 2011.

[72] Sohag Kabir. An overview of fault tree analysis and its application in model based dependability analysis. *Expert Systems with Applications*, 77:114–135, jul 2017.

[73] Jianwen Xiang, Kazuo Yanoo, Yoshiharu Maeno, and Kumiko Tadano. Automatic synthesis of static fault trees from system models. In *Proceedings - 2011 5th International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2011*, pages 127–136. IEEE, jun 2011.

[74] Myron Hecht, Emily Dimpfl, and Julia Pinchak. Using SysML to Automatically Generate of Failure Modes and Effects Analyses. *INCOSE International Symposium*, 25(1):1357–1372, oct 2015.

[75] Pierre David, Vincent Idasiak, and Frédéric Kratz. Reliability study of complex physical systems using SysML. *Reliability Engineering and System Safety*, 95(4):431–450, apr 2010.

[76] Morayo Adedjouma and Nataliya Yakymets. A framework for model-based dependability analysis of cyber-physical systems. In *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, volume 2019-Janua, pages 82–89. IEEE Computer Society, mar 2019.

[77] Saadia Dhouib, Selma Kchir, Serge Stinckwich, Tewfik Ziadi, and Mikal Ziane. RobotML, a Domain-Specific Language to Design, Simulate and Deploy Robotic Applications. In Itsuki Noda, Noriaki Ando, Davide Brugali, and James J Kuffner, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, pages 149–160, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[78] Philipp Helle. Automatic SysML-based safety analysis. In *MODELS 2012 Innsbruck - Proceedings of the 5th International Workshop on Model Based Architecting and Construction of Embedded Systems, ACES-MB 2012*, pages 19–24, New York, New York, USA, 2012. ACM Press.

[79] Kleanthis Thramboulidis and Sven Scholz. Integrating the 3+1 SysML view model with safety engineering. In *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010*, pages 1–8. IEEE, sep 2010.

[80] Geoffrey Biggs, Takeshi Sakamoto, and Tetsuo Kotoku. A profile and tool for modelling safety information with design information in SysML. *Software and Systems Modeling*, 15(1):147–178, feb 2016.

[81] Martina Muller, Michael Roth, and Udo Lindemann. The hazard analysis profile: Linking safety analysis and SysML. In *10th Annual International Systems Conference, SysCon 2016 - Proceedings*, pages 1–7. IEEE, apr 2016.

[82] Anis Baklouti, Nga Nguyen, Faïda Mhenni, Jean Yves Choley, and Abdelfattah Mlika. Improved safety analysis integration in a systems engineering approach. *Applied Sciences (Switzerland)*, 9(6):1246, mar 2019.

[83] SEBoK. System verification — sebok,, 2020. [Online; accessed 19-May-2021].

[84] Tomás Grimm, Djones Lettnin, and Michael Hübner. A survey on formal verification techniques for safety-critical systems-on-chip. *Electronics*, 7(6):81, 2018.

[85] Jacques Carette and John Harrison. Handbook of practical logic and automated reasoning. *Journal of Functional Programming*, 21(6):663, 2011.

[86] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. Model checking and the state explosion problem. In Bertrand Meyer and Martin Nordio, editors, *Tools for Practical Software Verification: LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*, pages 1–30, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[87] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell. Fault injection for dependability validation: a methodology and some applications. *IEEE Transactions on Software Engineering*, 16(2):166–182, 1990.

[88] Domenico Cotroneo, Luigi De Simone, Pietro Liguori, and Roberto Natella. Fault Injection Analytics: A Novel Approach to Discover Failure Modes in Cloud-Computing Systems. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, sep 2020.

[89] Mei Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer. Fault injection techniques and tools, apr 1997.

[90] Roberto Natella and Domenico Cotroneo. Assessing dependability with software fault injection: A survey. *ACM Comput. Surv*, 48, 2016.

[91] Maha Kooli and Giorgio Di Natale. A survey on simulation-based fault injection tools for complex systems. In *Proceedings - 2014 9th IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era, DTIS 2014*. IEEE Computer Society, 2014.

[92] Benjamin Vedder. Testing Safety-Critical Systems using Fault Injection and Property-Based Testing, 2015.

[93] Saurabh Jha, Subho S Banerjee, James Cyriac, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. Avfi: Fault injection for autonomous vehicles. In *2018 48th annual IEEE/IFIP international conference on dependable systems and networks workshops (DSN-W)*, pages 55–56. IEEE, 2018.

[94] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[95] Saurabh Jha, Timothy Tsai, Siva Hari, Michael Sullivan, Zbigniew Kalbarczyk, Stephen W Keckler, and Ravishankar K Iyer. Kayotee: A fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors. *arXiv preprint arXiv:1907.01024*, 2019.

[96] NVIDIA. NVIDIA DRIVE Sim. Accessed: 2021-4-27.

[97] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar, and Sastry Hari. AV-FUZZER : Finding Safety Violations in Autonomous Driving Systems. *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 1–12, oct 2020.

[98] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. Lgsvl simulator: A high fidelity simulator for autonomous driving. *arXiv preprint arXiv:2005.03778*, 2020.

[99] Garazi Juez, Estíbaliz Amparan, Ray Lattarulo, Alejandra Ruíz, Joshué Pérez, and Huáscar Espinoza. Early safety assessment of automotive systems using sabotage simulation-based fault injection framework. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10488 LNCS, pages 255–269, 2017.

[100] A Pena, I Iglesias, J Valera, and A Martin. Development and validation of Dynacar RT software, a new integrated solution for design of electric and hybrid vehicles. *EVS26 Los Angeles*, 26:1–7, 2012.

[101] Nathan Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154, 2004.

[102] Richard Van Der Horst and Jeroen Hogema. Time-to-collision and collision avoidance systems. In *Proceedings of the 6th ICTCT workshop: Safety evaluation of traffic systems: Traffic conflicts and other measures*, pages 109–121, 1993.

[103] SEBoK. System Life Cycle Process Models: Vee, 2019.

[104] Unreal Engine 4. https://www.unrealengine.com/en-US/.

[105] George-Dimitrios Kapos, Vassilis Dalakas, Mara Nikolaidou, and Dimosthenis Anagnostopoulos. An integrated framework for automated simulation of SysML models using DEVS. *Modeling and Simulation International*, 90(6):717–744, 2014.

[106] Wladimir Schamai, Philipp Helle, Peter Fritzson, and Christiaan J.J. Paredis. Virtual verification of system designs against system requirements. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6627 LNCS, pages 75–89, 2011.

[107] Visual Paradigm. Ideal Modeling & Diagramming Tool for Agile Team Collaboration, 2020.

[108] Sparx Systems. UML modeling tools for Business, Software, Systems and Architecture, 2006.

[109] Open Reliability. http://www.openreliability.org/fault-tree-analysis-on-r/.

[110] Matthew Leeke and Arshad Jhumka. Evaluating the use of reference run models in fault injection analysis. In *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 121–124. IEEE, 2009.

[111] P. Barber N. Clarke. Advanced collision warning systems. *IET Conference Proceedings*, pages 2–2(1), January 1998.

[112] K Lee and H Peng. Evaluation of automotive forward collision warning and collision avoidance algorithms. *Vehicle System Dynamics*, 43(10):735–751, oct 2005.

[113] Peter A. DeFazio. H.r.2 - 116th congress (2019-2020): Moving forward act. https://www.congress.gov/bill/116th-congress/house-bill/2, 2020.

[114] Garrick J Forkenbrock and Bryan C O'Harra. A forward collision warning (fcw) performance evaluation. In *Proc. 21st Int. Technical Conf. Enhanced Safety of Vehicles (ESV)*, pages 1–12, 2009.

[115] National Highway Traffic Safety Administration's Traffic Safety Facts Annual Report, TSF Table 42. https://cdan.dot.gov/tsftables/tsfar.htm#, 2021.

[116] Jessica B Cicchino. Effectiveness of forward collision warning and autonomous emergency braking systems in reducing front-to-rear crash rates. *Accident Analysis & Prevention*, 99:142–152, 2017.

[117] NHTSA Announces Update to Historic AEB Commitment by 20 Automakers | NHTSA. https://www.nhtsa.gov/press-releases/nhtsa-announces-update-historic-aeb-commitment-20-automakers, 2019.

[118] Yasuhiko Fujita, Kenji Akuzawa, and Makoto Sato. Radar brake system. *Jsae Review*, 1(16):113, 1995.

[119] R Ervin, J Sayer, D LeBlanc, S Bogard, M Mefford, M Hagan, Z Bareket, and C Winkler. Automotive collision avoidance system field operational test report: methodology and results. Technical report, NHTSA, 2005.

[120] Alexandra Neukum, Eric Ufer, Jörn Paulig, and Hans-Peter Krüger. Controllability of superposition steering system failures. In *Steering tech 2008*, Munich, 2008.