An Integrated Approach to Requirements Development and Hazard Analysis

2015-01-0274

Published 04/14/2015

John Thomas, John Sgueglia, Dajiang Suo, and Nancy Leveson
Massachusetts Institute of Technology

Mark Vernacchia and Padma Sundaram
General Motors Company

CITATION: Thomas, J., Sgueglia, J., Suo, D., Leveson, N. et al., "An Integrated Approach to Requirements Development and Hazard Analysis," SAE Technical Paper 2015-01-0274, 2015, doi:10.4271/2015-01-0274.

Abstract

The introduction of new safety critical features using software-intensive systems presents a growing challenge to hazard analysis and requirements development. These systems are rich in feature content and can interact with other vehicle systems in complex ways, making the early development of proper requirements critical. Catching potential problems as early as possible is essential because the cost increases exponentially the longer problems remain undetected. However, in practice these problems are often subtle and can remain undetected until integration, testing, production, or even later, when the cost of fixing them is the highest.

In this paper, a new technique is demonstrated to perform a hazard analysis in parallel with system and requirements development. The proposed model-based technique begins during early development when design uncertainty is highest and is refined iteratively as development progresses to drive the requirements and necessary design features. The technique is evaluated by applying it to a realistic but generic Shift-By-Wire design concept in two iterations with varying levels of detail. In addition, as the requirements and design evolve and change over time, the changes can be immediately analyzed for new hazards without repeating the entire analysis. The approach is also applicable even before requirements are developed, providing feedback when some of the most important decisions are being made instead of waiting for a finished design or model to begin an analysis. In this way, potential issues can be identified immediately and more efficiently, thereby reducing the need for future rework.

Introduction

Modern automobiles are incorporating more and more advanced software-controlled safety features such as lane keeping assist, automatic emergency braking, adaptive cruise control, and a growing number of automated or semi-automated systems. Increasingly complex behaviors are now being incorporated into safety-critical software and there are very few physical constraints that limit the complexity of the software elements. While this has allowed many innovative features that were not possible in the past, the additional complexity has made it much more challenging to develop these systems and perform adequate safety analyses. Software-related issues are frequently being discovered late in development, testing, and even during production when the problems are the most expensive to fix and when the range of practical solutions is the most limited.

The software in modern vehicles is not only complex but also increasing in size exponentially, making the need for efficient analysis techniques increasingly urgent. One manufacturer reports two to three million lines of software code for average 2005 models [1], six million lines of code for 2007-2008 models [1], and sixteen million lines of code in 2013 [2]. Meanwhile, NHTSA data shows that more and more software-related issues are being discovered too late. About 382,000 U.S. vehicles were indicated in software-related recalls in the year 2000, compared to 13 million in 2013 and 48 million a year later [3]. In fact, some have claimed that up to 50% of car warranty costs are related to the electronics and their embedded software [4].

The traditional techniques used for analysis have not kept pace with the increased complexity of modern software-intensive systems. Although most analysis techniques consider individual failures (e.g. Fault Tree Analysis, Failure Modes and Effects Analysis, etc.), many other safety-critical decisions and assumptions are made during the development process and can easily be overlooked in a component failure-based analysis. Judgments about what behavioral requirements are needed, what interactions could potentially be dangerous, what information and feedback the software decisions should be based on, how to coordinate multiple distributed controllers, and other issues must be determined during development. While these issues are typically addressed using engineering best-judgment, they are often resolved in a compartmentalized manner without a complete understanding of the system-level impact of the proposed solution. As a result, unintended interactions are often introduced and may be overlooked by standard failure-based analyses.

In fact, a new class of accidents called "component interaction accidents" is becoming increasingly common. Component interaction accidents arise due to dysfunctional or unintended interactions among several components and often occur without a component failure. For example, the software requirements may be incorrect, incomplete, or ambiguous. Individual software components may operate exactly as required and as designed without failure, but the overall behavior of multiple interacting components may still lead to unanticipated vehicle behavior. In fact, most software-related accidents are caused not by coding errors but flawed software requirements [5, 6, 7,12]. Such flaws can be very difficult to identify or anticipate with traditional failure-based methods or by analyzing individual components in isolation.

Another limitation is that most approaches to safety analysis focus on assessment of an existing design as opposed to driving the design and requirements from the start. Most techniques require a considerable amount of information about the system to perform an effective analysis, but some of the most important decisions affecting safety are made early during development before the design and requirements are known. When a safety analysis is eventually performed, the result is often rework or inefficient patches to correct earlier mistakes. In many cases the best solutions are no longer practical to implement by the time problems are discovered, resulting in difficult compromises.

Although system engineering processes continue to emphasize more up front hazard and risk assessments, integrating hazard analysis techniques into early engineering design processes remains a challenge. Engineering and analysis often remain separate sequential activities, effectively adding new analysis and assessment tasks to an already long list of otherwise unchanged engineering tasks. What is needed is not simply more analysis and more paperwork, but more efficient and effective ways to do system engineering from the start. A truly integrated safety-driven design process should not just periodically monitor the design but continuously drive decisions as they are made so that safety is "baked in" and less analysis is required later. Such a process has the potential to identify problems sooner, reduce rework when problems are corrected, reduce the total integration time needed later, and simplify the final safety assessment by leveraging analysis artifacts already created during development.

The functional safety standard ISO 26262 describes activities and requirements throughout the safety lifecycle of safety-related systems comprised of electrical, electronic, and software elements that provide safety-related functions [8]. The standard is based on the popular system engineering V-Model [9] and has the potential to encourage consideration of safety aspects much earlier than other standards such as IEC 61508 [10,11]. In many areas, ISO 26262 lists broad activities that must be done but does not prescribe a specific method to be used or provide a specific process to follow. For example, Part 3 Clause 7.4.4.2.1 [8] requires that potential sources of harm "shall be determined systematically by using adequate techniques". A note suggests that a variety of techniques such as brainstorming could be used, but no specific process is provided to ensure the requirement is met. Part 3 Clause 7.4.4.2.4 [8] requires that "consequences of hazardous events shall be identified.", but no method or process is indicated and "relevant" is left to the reader to decide. These are not necessarily weaknesses of the standard, and in fact these are advantages in many ways because they offer flexibility and can accommodate innovation and advancement as new hazard analysis methods are developed.[1] However, the standard is not enough on its own as it generally stops short of prescribing a specific method or technique to be used.[2]

Hazard analysis methods such as STPA (System Theoretic Process Analysis) can be used to satisfy some of these goals. STPA was created based on systems theory to address the limitations of traditional safety analysis techniques [12]. STPA is a top-down hazard analysis method designed to go beyond traditional component failures to also identify problems such as dysfunctional interactions, flawed requirements, design errors, external disturbances, human error and human-computer interaction issues, and other problems. Although STPA has been very successful, it is typically applied as a separate analysis-i.e. all steps of the traditional STPA analysis are completed before the system is fixed, refined, or augmented. More detailed comparisons and evaluations of STPA can be found in [13, 14, 15].

This paper proposes and demonstrates a new safety-guided *design* methodology based on STPA that interleaves development and analysis tasks to provide an integrated development process. The process begins with very little information about the system design and proceeds to drive an initial control model and initial behavioral requirements. Design decisions can be incorporated in top-down fashion as they are made, and the safety implications are immediately fed back into the engineering process to reduce future rework. New techniques are also incorporated to help systematically ensure potential unsafe controls are not overlooked and that the resulting behavioral requirements are complete and consistent. This process can also be used to help prevent initial mistakes in requirements, specification, and design rather than waiting to assess and correct mistakes after-the-fact. In addition, all intermediate results of the process can be accumulated and assembled at the end to create a comprehensive overall safety analysis, thereby reducing extra work that is often done at the conclusion of each development stage.

System Theoretic Process Analysis

STPA begins by identifying the system accidents and system hazards to be prevented [12]. In STPA, an accident is an event that results in a loss and a system hazard is a system state that will lead to an accident in a worst-case environment [12]. Although the accidents of interest often involve human injury or loss of life, STPA can also be applied more generally to other losses that must be prevented [12]. Other losses may include quality issues such as loss of customer satisfaction, performance issues such as reduced power or efficiency, economic issues such as damage to the vehicle, environmental pollution, or any other loss that is unacceptable.

STPA uses a model of the control relationships within the system to guide the analysis. These relationships are modeled using a safety control structure, as shown in Figure 1. Control actions or commands that affect lower-level processes are identified as well as feedback

1. In other areas ISO26262 provides a non-exhaustive list of methods that might be used, but stops short of requiring a specific method. For example, Part 9 Clause 8.2 [8] lists Failure Modes and Effects Analysis (FMEA), Fault Tree Analysis (FTA), HAZOP, and Event Tree Analysis (ETA), although with a note that "The qualitative analysis methods listed above can be applied to software where no more appropriate software-specific analysis methods exist." Part 4 Clause 7.4.3.1 [8] provides recommendations for inductive vs. deductive analysis methods, but does not require a particular method to be used.

2. For a more detailed assessment of ISO 26262 and its relationship to system engineering and System Theoretic Process Analysis see [11].

that is used to inform higher-level controllers. The analysis begins by identifying control actions that can be unsafe and may lead to a hazard. There are four ways in which a control action may be hazardous:

1). A control action required for safety is not provided.
2). A control action is provided when it is unsafe to do so.
3). A control action is provided too early or too late.
4). A continuous control action is applied too long or stopped too soon.

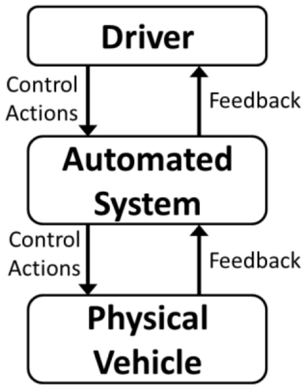

Figure 1. Simplified example control structure

Once Unsafe Control Actions (UCAs) have been identified, the analysis proceeds to identify potential causes of the unsafe control. A generic control loop with causal factors, as shown in Figure 2 from [12], can be used to guide this part of the analysis. One important cause of unsafe control actions involves the controller's process model, also called a mental model when the controller is a human. The process model captures the controller's understanding and beliefs about the outside world, including the state of the controlled process and assumptions about how the controlled process works. Accidents in complex systems, particularly those related to software, often result from inconsistencies between the model of the process used by the controller and the actual process state, which leads to the controller providing unsafe control. Figure 2 also shows other potential causes such as missing or inadequate feedback, component failures (such as a sensor failure), inadequate control inputs, and others.

In addition to identifying causes of unsafe control actions, STPA also identifies how safe control actions may not be followed or executed properly. For example, appropriate control actions may be issued but the actuator may introduce excessive delays or the controlled process may experience a physical failure that prevents the actuator from being effective.

The following sections introduce a safety-driven design process based on STPA that begins with very little information and proceeds to drive initial development and design decisions. The process is demonstrated using a realistic but fictional shift-by-wire concept with the potential for safety-critical automated behaviors.

New Integrated Approach

Process overview

The safety-driven design process proposed in this paper interleaves hazard analysis tasks with development tasks to provide quicker feedback to engineers and to provide guidance as the system is being developed. The approach can be summarized as:

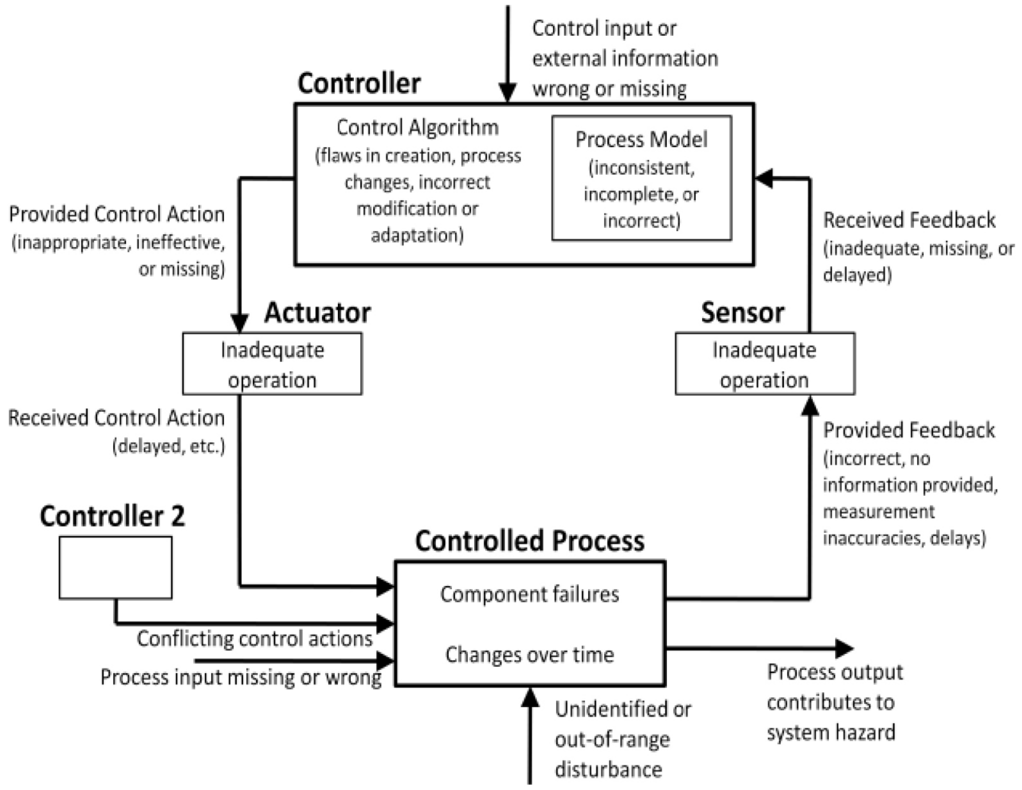
Control input or external information wrong or missing
Controller
Control Algorithm (flaws in creation, process changes, incorrect modification or adaptation)
Process Model (inconsistent, incomplete, or incorrect)
Provided Control Action (inappropriate, ineffective, or missing)
Received Feedback (inadequate, missing, or delayed)
Actuator
Inadequate operation
Sensor
Inadequate operation
Received Control Action (delayed, etc.)
Provided Feedback (incorrect, no information provided, measurement inaccuracies, delays)
Controller 2
Controlled Process
Component failures
Changes over time
Conflicting control actions
Process input missing or wrong
Unidentified or out-of-range disturbance
Process output contributes to system hazard

Figure 2. Causal factors used to create scenarios in STPA [12]

First iteration:

1. Define the system accidents and system hazards
2. Create the initial control structure
3. Identify initial unsafe control actions
4. Derive safety constraints/requirements from unsafe control actions

a. Use the safety constraints to revise the control structure and design

5. Identify high-level causal scenarios

a. Identify controls to eliminate or mitigate the high-level scenarios

Second iteration:

6. Formalize the unsafe control actions to identify any missing or conflicting UCAs and constraints

a. Resolve the identified conflicts and revise the safety constraints

7. For scenarios not already controlled, identify more detailed causes by incorporating additional design detail

a. Provide controls for the new causal factors identified

System Overview

A new vehicle shift-by-wire concept was chosen to demonstrate and evaluate the proposed safety-driven design process. The generic shift-by-wire concept was chosen because it introduces a new software controller with the potential for automated behavior and new safety features. It is also a relatively new technology that is under active development by several manufacturers. In addition, there are a large number of design decisions that clearly impact safety (such as what automated behaviors are appropriate and what information the controller should monitor) but there are very few regulations, industry standards, or processes that provide clear answers.

The shift-by-wire concept replaces traditional mechanical cables between the shifter and the transmission with an electronic lever, a shift control module, and various electronic actuators and sensors. The shift control module senses the shift lever position and commands some actuator to achieve the appropriate transmission range. Although the shift-by-wire concept may reduce manufacturing costs and increase packaging flexibility, there are several important safety implications that require careful consideration. Note that many details-such as the shift control module algorithm and the method of sensing and actuating-are unknown at this early stage and they are not needed to begin the proposed safety driven design process.

Building a foundation

The first step is to identify the system accidents and system hazards. These form a foundation for the safety-driven design process and define the losses or system states are unacceptable and must be prevented. At this early phase, the system is the overall vehicle. Tables 1 and 2 list the system accidents and system hazards defined for this case study. Notice that these are defined at a very high level and in fact they will be similar for many automotive systems. The system accidents and system hazards help define the scope of the effort, and all results from the safety-driven design process will be traceable to these system accidents and system hazards.

Note that the term "system hazard" as defined in STPA is different from the term "hazard" in ISO 26262, which is defined as any potential source of physical injury or damage to the health of people [12]. Because STPA is a top-down process, it does not start with a list of all potential sources of injury or damage. Instead, it begins with system-level states or conditions that must be prevented (i.e. system hazards) and proceeds to systematically identify the potential causes.[3]

Table 1. System Accidents

Number	System Accident Description
A-1	Two or more vehicles collide
A-2	Vehicle collides with other obstacle or terrain
A-3	Vehicle occupants injured without vehicle collision

Table 2. System Hazards

Number	System Hazard Description
H-1	Vehicle does not maintain safe distance from nearby vehicles
H-2	Vehicle does not maintain safe distance from terrain and other obstacles
H-3	Vehicle enters uncontrollable/unrecoverable state
H-4	Vehicle occupants exposed to harmful effects and/or health hazards

The first system hazard describes vehicles that become too close to each other. Because the system hazards are system states and conditions by definition, they do not specify potential causes and the analysis is not restricted to certain types of causes. The system hazards simply specify the system behavior that must be prevented, and all relevant causes will be systematically identified later. Potential causes of H-1 could involve a vehicle accelerating into another vehicle, an unsecured vehicle rolling backwards towards another vehicle, a vehicle moving in an unintended direction, physical malfunctions, driver errors, confusing automation, etc.

System hazard H-2 is similar to H-1 but refers to vehicles that are too close to non-vehicular objects. Examples of other objects include pedestrians, animals, bikers, and guardrails. H-3 captures situations in which the driver may be unable to use the vehicle's systems to gain control. This includes cases in which the vehicle may be travelling too fast, the driver's commands are ignored, or a system such as braking or power steering is ineffective. H-4 captures additional problems that may occur even without nearby objects such as a vehicle rollover, excessive deceleration or acceleration, or a vehicle fire.

Note that the System Hazards can lead to the System Accidents. For example, H-1 can cause A-1, H-2 can cause A-2, H-3 can lead to any of the accidents, and H-4 can lead to A-3.

3. In fact, STPA system accidents and system hazards can be defined much broader than ISO 26262 and may include anything that is unacceptable to the user and must be prevented. For example, loss of customer satisfaction or damage to the vehicle (without human injury) could be included if desired [12].

Figure 3 shows the high-level system control structure model of the traditional mechanical shifter system, while Figure 4 shows the high-level control structure model for the proposed shift-by-wire system. These control structures are defined directly from the initial shift-by-wire description above and do not yet contain much detail. This is intentional; by starting at a very high level, the process can begin even before the design and requirements are available and provide actionable results before rework is needed.

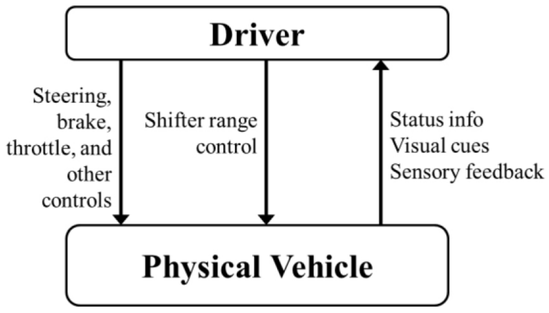

Figure 3. High-level control structure for traditional mechanical shifting

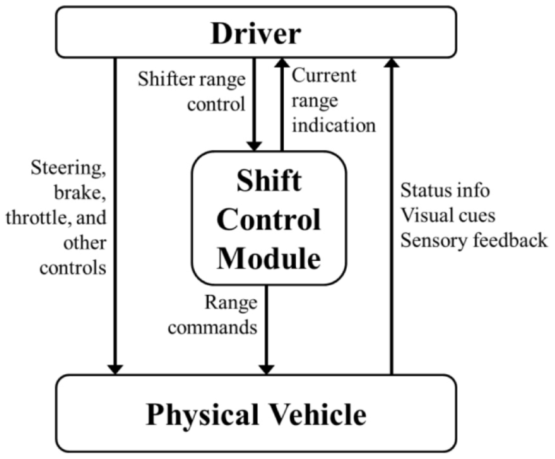

Figure 4. Initial high-level control structure for shift-by-wire system

Initial unsafe control actions and safety constraints

Once the high-level control structure is defined, an initial set of unsafe control actions can be identified by considering the four ways a control action can be unsafe, as discussed in the introduction. Table 3 shows the results that can be obtained at this stage of the process for the Range Command. The Range Command, also shown in Figure 4, is a command from the Shift Control Module to select a new range (e.g. Park, Reverse, Neutral, Drive, etc.).

Each unsafe control action is then translated into a safety constraint / safety requirement to be enforced, as shown in Table 4. As the design progresses, these unsafe control actions and safety constraints will be refined and examined in more detail.

Table 3. Initial unsafe control actions for shifter control module[4]

Control Action	Not providing causes hazard	Providing causes hazard	Too early, too late, wrong order	Stopped too soon, applied too long
Range command (from shifter control module)	UCA-1: Shifter Control Module does not provide range command when driver selects new range [H-1, H-2, H-3, H-4] UCA-2: Shifter Control Module does not provide new range command once current range becomes unavailable [H-1, H-2, H-3, H-4]	UCA-3: Shifter Control Module provides range command without driver new range selection [H-1, H-2, H-3, H-4] UCA-4: Shift Control Module provides range command when that range is unavailable [H-3] UCA-5: Shift Control Module provides inconsistent range command [H-1, H-2, H-3, H-4]	UCA-6: Shifter Control Module provides range command too late after driver range selection [H-1, H-2, H-3, H-4] UCA-7: Shift Control Module provides range commands consistent with driver selection but in different order [H-1, H-2, H-3,H-4]	N/A

4. Inconsistent means the requested range would cause physical damage, an unsafe change in motion, or violate motor vehicle regulations. Unavailable means a physical fault exists that would prevent the vehicle from shifting to the selected range. Also note that each UCA includes a link to the system hazards to provide traceability; every STPA result is traceable to the system hazards and accidents defined at the start of the analysis.

As soon as the initial safety constraints are defined, potential design implications can be assessed. Although the process is not finished and causal factors have not yet been identified, these preliminary results can immediately be used to drive important design decisions and identify necessary safety features. The following questions can be used to guide this part of the process:

- Does the initial control structure allow the controller to monitor the conditions in the constraints?
- Do additional control actions need to be added to achieve or enforce the constraints?
- Are there other controllers that may interfere with or violate the constraints?

Table 4. Safety constraints derived from unsafe control actions

Safety Constraint	Description
SC-1	Shifter Control Module must provide range command when driver selects new range [H-1, H-2, H-3, H-4]
SC-2	Shifter Control Module must provide new range command once current range becomes unavailable [H-1, H-2, H-3, H-4]
SC-3	Shifter Control Module must not provide range command without driver new range selection [H-1, H-2, H-3, H-4]
SC-4	Shift Control Module must not provide range command when that range is unavailable [H-3]
SC-5	Shift Control Module must not provide range commands that are inconsistent [H-1, H-2, H-3, H-4]
SC-6	Shifter Control Module must provide range command within X milliseconds of driver range selection [H-1, H-2, H-3, H-4]
SC-7	Shift Control Module must provide range commands in the same order as received by the driver [H-1, H-2, H-3,H-4]

For example, SC-4 states that the shift control module must not issue a range command when that range is unavailable. How would the controller know which ranges are available or unavailable? The initial control structure in Figure 4 does not provide the Shift Control Module with that information. This new feedback can be immediately identified and added to the control structure without requiring additional design details or a detailed analysis. By identifying these dependencies early, solutions can be incorporated during early development without causing additional rework.

Figure 5 shows a revised control structure with the available range feedback. Note that it is not necessary yet to know how the available ranges are detected, just that they must be detected. The constraints produced at this stage are intentionally flexible to allow the design team to determine the most effective solutions. Once specific solutions are proposed, the control structure can be refined to include additional detail and identify whether any new safety concerns are introduced.

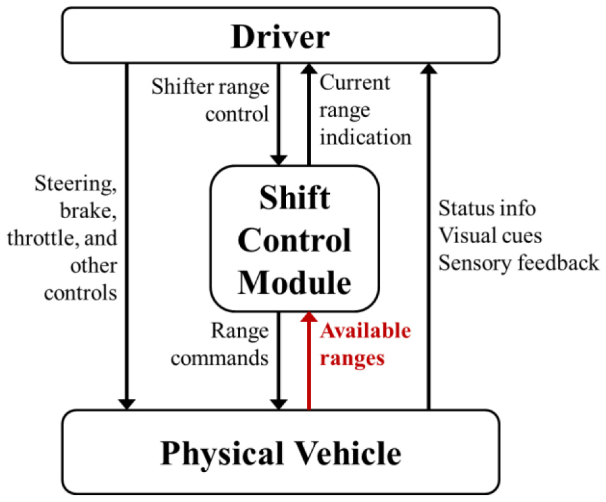

Figure 5. Revised control structure

Identifying causal scenarios

Once the control structure has been revised, causal scenarios can be identified for each of the unsafe control actions. The causal factors in Figure 2 can be used to guide the generation of causal scenarios. Notice that more design information may be incorporated at this stage, such as information about the controller process model and other control inputs. Table 5 shows two causal scenarios that can be identified for the first unsafe control action and corresponding safety constraint:

Table 5. Causal scenarios that violate SC-1

Safety Constraint	**Causal scenario that violates the safety constraint**
SC-1: Shifter Control Module must provide range command when driver selects new range	**S-1**: Shifter Control Module does not provide range command because it *incorrectly believes no new range was selected* **S-2**: Shift Control Module does not send range command because it *incorrectly believes the range was already achieved* *Etc.*

Once the causal scenarios are identified, potential design implications can be assessed. The following questions can be used to help guide this part of the process:

- How does the controller determine the information referenced in the scenarios?
- Are additional controls needed to prevent identified flaws?
- Are new controllers or new functionalities needed?
- Do new constraints need to be defined?

For example, S-2 describes a case where the shift control module incorrectly believes a commanded range was achieved. How could this incorrect belief arise? In the control structure in Figure 5, the controller would only be aware of the last range command sent; there are no means for the controller to detect the current range. This reveals a potential hidden assumption that the current range matches the previous range command. However, if the command is not executed for any reason (whether a failure or otherwise), it will lead to S-2. One solution is to incorporate an additional feedback for the Shift Control Module to directly detect the current range, as shown in Figure 6.

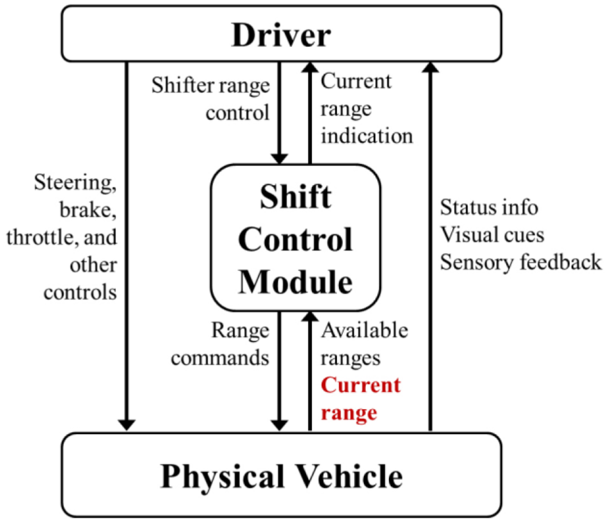

Figure 6. Revised control structure

By identifying these potential issues early, it is possible to drive the initial design decisions without causing rework.

So far, the process has focused on what control and feedback information will be required to be safe and what basic behaviors must be controlled. Specific design details have not yet been included so that the process can be performed quickly and extremely early during development. However, the design details eventually need to be incorporated into the process because they can introduce new safety concerns or cause unexpected side-effects. The next section explains how the process may be iterated to refine the requirements and provide more specific results based on the design details.

Refining and formalizing unsafe control actions

Although the initial set of unsafe control actions in Table 3 can be identified quickly to provide immediate feedback to the design team, a second iteration can be performed to help identify any additional UCAs and to identify and resolve any conflicts between the existing safety constraints. In addition, the results from the first iteration may have caused new interactions among existing controllers or new controllers to be introduced. By applying a formal framework to the initial set of unsafe control actions, a revised and more precise set of safety constraints and executable requirements can be defined.

A rigorous process for systematically identifying unsafe control using a formal framework has been developed in [16]. Missing UCAs and potential conflicts can be logically identified using a context table as shown in Table 6. Each row in a context table corresponds to an unsafe control action, and the columns represent different elements of the unsafe control action. For example, the first row of the context table in Table 6 corresponds to UCA-3 (Shifter Control Module provides range command without driver new range selection). The first column specifies that the control action is the range command. The next four columns describe the conditions that can make a range command unsafe, and asterisks are used to denote which conditions do not matter for a given row.

Table 6. Context table for Shifter Control Module UCAs

Control Action	**Driver Selected Range**	**SCM Selected Range Available**	**SCM Selected Range Consistent**	**Current range available**	**Not Providing Causes Hazards**	**Providing Causes Hazards**	
Shifter Control Module provides range command when…	None	*	*	*	No	Yes	UCA-3
	*	*	*	No	Yes	No	UCA-2
	Doesn't match SCM cmd	*	*	*	No	Yes	
	Matches SCM cmd	*	*	*	Yes	No	UCA-1
	Matches SCM cmd	No	*	*	No	Yes	UCA-4
	Matches SCM cmd	*	No	*	No	Yes	UCA-5

* denotes conditions that do not matter for a given row

The final two columns indicate whether providing or not providing the control action in the given context could cause a hazard. Note that the affirmative answers (indicating an unsafe control action) are just as important to document as the negative answers (indicating when control actions are believed or assumed to be safe). These assumptions are important because they influence the UCAs and requirements that are (or are not) defined. Although documenting assumptions is widely recognized as an important part of system engineering and many accidents have been linked to undocumented assumptions, very few techniques provide ways to systematically identify assumptions as they are made. Using a context table such as Table 6 can help make it clear what assumptions are being made and ensure that they are recorded so they can be reviewed or revisited if changes are later made to the system.

Comparing Table 6 to the initial set of unsafe control actions reveals a new unsafe control action. The third row describes a case where the range command does not match the driver selected range:

- **New UCA-8**: Shifter Control Module provides a range command that does not match the new range selection provided by the driver

The context table can also help identify conflicts between the UCAs and identify multiple safety constraints that cannot all be satisfied. For example, rows 4 and 5 have conditions that overlap (i.e. the driver selected range matches SCM command and the selected range is not available) but they give conflicting answers: row 5 indicates that it is hazardous to provide the command while row 4 indicates it is hazardous not to provide the command. Table 7 shows the two UCAs and safety constraints side-by-side. This is an example of an unresolved conflict, and the Shifter Control Module as currently conceived may have no choice but to cause UCA-1 or UCA-4 through either action or inaction. In other words, in the current design it is not possible to satisfy both of the corresponding safety constraints if the driver selects an unavailable range.

Table 7. Unresolved conflict between two safety constraints

	<= Conflict =>	
UCA	UCA-1: Shifter Control Module does not provide range command when driver selects new range	UCA-4: Shift Control Module provides range command when that range is unavailable
Safety Constraint	SC-1: Shifter Control Module must provide range command when driver selects new range	SC-4: Shift Control Module must not provide range command when that range is unavailable

Once the conflicts and ambiguities are identified, there are often several potential ways to resolve them. In this case, one solution might be to design the Shift Control Module to take a special action when the driver selects an unavailable range (such as warning the driver or re-trying the range). The UCAs and safety constraints can then be updated accordingly. However, the most significant challenge is often not in ensuring that known problems are solved but in ensuring that unknown problems are identified early and will not go unnoticed until late in development or even during production. By formalizing the unsafe control actions early, the results can be used to drive the design and requirements in real time. In this case, SC-1 could be revised as follows:

- **Revised SC-1**: Shifter Control Module must provide range command when driver selects new range that is available

Because the context table is based on a formal definition of an unsafe control action, conflicts may be automatically identified from the table using software tools. The completed context table also represents a model of the controller that issues the control actions and can be used to automatically generate executable safety requirements. For more information about automatic conflict detection and requirements generation, see [16]. For more information about tools being developed for this purpose, see [17].

Scenario refinement and adding design detail

At this point in the process, the causal accident scenarios that have been identified do not include much detail about the design. Some of the causal scenarios were already resolved without needing to dive into the design details. For example, S-2 in Table 5 was addressed by adding new feedback to the control structure. However, other scenarios may depend on more specific design details. For example, consider the following scenario:

- **S-3**: Shifter Control Module does not provide range command because it receives incorrect feedback that the range is already selected

Without additional detail, it is not clear what could cause the range feedback to be incorrect or what controls are needed to prevent this scenario. To refine the scenario, it is necessary to “zoom in” on the Current Range feedback in the control structure and incorporate decisions such as:

- How are the range commands implemented?
- How is the current range sensed?
- What values can be reported to the Shift Control Module?

Notice that none of this information was needed until now, which is desirable because the answers may not have been available previously if development is occurring in parallel. However, once potential solutions are eventually proposed they can be immediately incorporated into the model by expanding the appropriate part of the control structure. Figure 7 shows a revised control structure with new detail shown in the dotted box. A new controller called the Range Motor Controller is added to monitor changes in the motor position, calculate the current range, and report the result to the Shift Control Module. In this example, it was also decided that the available range data could be obtained from an existing transmission controller, creating a new feedback loop. By refining the control structure and adding detail as it becomes available, the existing scenarios can be refined to identify more specific causes of unsafe control and determine whether the safety constraints are adequately enforced.

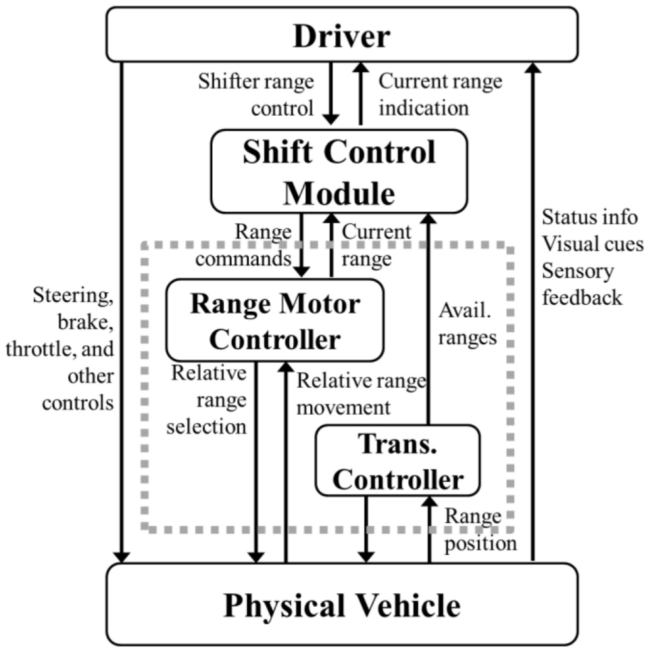

Figure 7. Revised control structure with additional detail in the dotted box

Given the additional detail in Figure 7, scenario S-3 can now be refined to include more detailed causes such as measurement errors or glitches that could result in a fixed offset when the relative movements are integrated. Notice that the current design does not yet have any way to detect this problem or to recover from it. As shown in Table 8, multiple solutions are possible-use a different type of measurement, use the transmission controller to independently verify the current range, provide a way to manually reset the transmission range, etc.

Table 8. Potential design solutions to prevent causal scenarios that violate safety constraints

Refined Safety Constraint	**Detailed causal scenario that violates the constraint**	**Potential controls or design decisions**
SC-1: Shifter Control Module must provide range command when driver selects new range that is available	**S-3**: Shifter Control Module does not provide range command because it receives incorrect feedback that the range is already selected. The incorrect feedback could be caused by a fixed offset in the range motor controller due to a measurement error or a momentary glitch in the relative range movement signal.	- Provide a way to verify the current range (e.g. from transmission controller) - Provide a different type of feedback other than detecting relative movement - Provide a way to reset the transmission range (E.g. reset any time range is changed, provide automatic or manual reset capability, etc.)

This process can be repeated for any remaining scenarios that are not already controlled while scenarios and causes that are already prevented at a high level of abstraction do not need to be studied in more detail. In this way, the complexity of the analysis and the development process is managed while efficiency is improved by providing immediate feedback as the development progresses.

Discussion and Conclusions

The model-based safety-driven design process described in this paper integrates hazard analysis with the design process to build safety into a system as opposed to relying on after-the-fact or periodic analysis. The process was demonstrated on an initial shift-by-wire concept that replaces mechanical shift controls with electronic and computer controls. Intermediate results were used to define the necessary safety constraints/requirements, and corresponding design features were introduced as needed throughout the process. The process was also found to be more efficient because there was no need to wait for a completed analysis before making changes to the design, and the full analysis did not have to be repeated as design changes were made.

The first iteration can be performed very quickly and requires very little information to begin. The issues identified at this stage tend to be broad in nature, such as feedback or measurements that are altogether missing from the initial system. However, these issues can be easily addressed without requiring major rework when the process is started early. Most of the accident scenarios identified in the first iteration can be immediately addressed, but a few require more detailed study.

The second iteration is more rigorous and identifies deeper issues such as inconsistent safety constraints or feedback that exists but may not be trustworthy. A little more detail is required and the second iteration may take longer than the first, but the process is efficient and leverages all the results from the first iteration to avoid any wasted effort. The detailed scenarios that are identified at this stage do not require major design changes but instead tend to be governed by decisions naturally made during detailed design (such as the type of sensor used to detect the current range).

Some of the issues identified with this approach involve traditional component failures such as a failed sensor. However, many of the issues involved potential requirements and interaction problems that can be much easier to overlook. For example, Table 8 suggests that an additional control action and perhaps a new actuator should be added to the design concept. If the actuator had been included in the initial design then most traditional approaches could easily analyze the physical component, apply failure modes, etc. However few techniques can systematically identify components that are altogether *missing* from the design and had never even been conceived of. Similarly, the proposed approach has the ability to identify potentially hidden design assumptions unlike traditional analysis techniques.

Another important result of this approach is that behavioral software requirements were easily defined in Table 4 based on the unsafe control actions. Although it is fairly straightforward to derive hardware reliability requirements such as such as failure rates based on overall reliability objectives, other requirements can be much more challenging. In particular, there are very few methods that systematically produce the necessary software behavioral requirements such as "Software must provide X output whenever Y occurs". It is also very encouraging that missing requirements could be systematically identified using the proposed process (e.g. from UCA-8). Given the growing complexity of software requirements and the features they provide, this result could play an important role in reducing the number of problems discovered late and the amount of rework needed during later phases of development.

Although much of the focus has been on software, the process also considered important interactions between the vehicle and the human driver such as the driver selecting unavailable ranges. The driver was modeled just as easily as the software was-both are controllers that provide control actions based on beliefs about the system, and both use feedback to update an internal process model of the system. The causes of unsafe behavior, such as inadequate feedback and incorrect beliefs, are also very similar between the two and can be analyzed using the same process.

When the process is finished, executable software requirements can be generated and used to guide detailed software development, produce a controller model, perform model-based simulations, or intelligently generate test cases that are relevant for safety. Tools that can help produce executable requirements are being developed [17] and tools that can simulate the existing requirements are available [18, 19, 20].

References

1. Siemens, "Ford Motor Company Case Study," Siemens PLM Software, 2014, Retrieved from http://www.plm.automation.siemens.com/pub/case-studies/14303?resourceId=14303
2. McKendrick, J. "Cars become 'datacenters on wheels', carmakers become software companies," ZDJNet, 2013
3. NHTSA Office of Defect Investigation, Recalls [Data file], 2014 Retrieved from http://www-odi.nhtsa.dot.gov/downloads/flatfiles.cfm/FLAT_RCL.zip
4. Charette, R. "This car runs on code," *IEEE Spectrum*, 2009.
5. Leveson, N. "Safeware: system safety and computers," Addison-Wesley, Reading, MA, 1995.
6. Lutz, R.R. "Analyzing software requirements errors in safety-critical, embedded systems," *IEEE International Conference on Software Requirements*, 1992.
7. Leveson, N. "Role of Software in Spacecraft Accidents," *Journal of Spacecraft and Rockets*, 41.4 (2004): 564-75.
8. ISO 26262:2011, "Road Vehicles - Functional Safety" International Standardization Organization, Nov 2011
9. Handbook of Systems Engineering, V3.2.1, International Council of Systems Engineering, 2011
10. IEC 61508, "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems," International Electrotechnical Commission. Edition 2.0, 2010-04.
11. Van Eikema Hommes, Q., "Review and Assessment of the ISO 26262 Draft Road Vehicle - Functional Safety," SAE Technical Paper 2012-01-0025, 2012, doi:10.4271/2012-01-0025.
12. Leveson, N. "Engineering a Safer World," MIT Press, Cambridge, MA, 2012.
13. Balgos V. H., "A systems theoretic application to design for the safety of medical diagnostic devices," Master's thesis, MIT, 2012
14. Torok, R., Geddes, B., Systems Theoretic Process Analysis(STPA) Applied to a Nuclear Power Plant Control System, *MITSTAMP Workshop*, March 2013
15. Leveson, N., Wilkinson, C., Fleming, C., Thomas, J., Tracy, I., A Comparison of STPA and the ARP 4761 Safety Assessment Process, *MIT PSAS Technical Report*, 2014
16. Thomas, J. "Extending and Automating a Systems-Theoretic Hazard Analysis for Requirements Generation and Analysis," Ph.D. Dissertation, Engineering Systems Division, MIT, 2013.
17. Thomas, J. and Suo, D. "An STPA Tool," *3rd STAMP/STPA Conference*, Cambridge, MA, 2014.
18. Leveson, N., Heimdahl, M., and Reese, J. "Designing specification languages for process control systems: lessons learned and steps to the future," *Proceedings of the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 1999, Springer-Verlag: Toulouse, France. p. 127-145.
19. Leveson, N. "Completeness in formal specification language design for process-control systems", *Proceedings of the Third Workshop on Formal Methods in Software Practice*, ACM p75-87, 2000
20. Bellagamba, L. "Systems Engineering and Architecting: Creating Formal Requirements", CRC Press, 2012

ISSN 0148-7191

http://papers.sae.org/2015-01-0274