Safety-Driven Design for Software-Intensive Aerospace and Automotive Systems

INVITED PAPER

Safety-Driven Design for Software-Intensive Aerospace and Automotive Systems

A new hazard-analysis technique, that gives system designers the information they need to make good decisions before their designs are completed, has been successfully applied to many diverse systems.

By Margaret V. Stringfellow, Nancy G. Leveson, *Member IEEE*, and Brandon D. Owens

ABSTRACT | Too often, systems are designed and then an attempt is made to add safety features or to prove that the design is safe after the fact. Safety has to be designed into a system from the start—it cannot be effectively added on to a mature design. In addition, the increasing use of software is changing the nature of accident causation in software-intensive systems and our safety engineering techniques must change accordingly. This article will describe a new hazard analysis technique, called STPA, which is effective on software-intensive systems. An advantage of this technique is that it can be used to drive the earliest design decisions and then proceed in parallel with ensuing design decisions and design refinement. Not only is this approach more effective, but the cost is no more than a more conventional design process and potentially much cheaper.

KEYWORDS | Accident; complexity; control; hazard; process; risk; safety; safety-driven design; software; STAMP; STPA

I. INTRODUCTION

While the traditional approach to design of safety-critical systems has been very successful in the aerospace and automotive industries, we believe that relative simplicity and the ability to effectively isolate and simplify the interfaces between system components has been a critical factor in this success. These factors, however, are changing: 1) aerospace and automotive software is becoming more complex along with the systems in which it is embedded and 2) software is being used to implement complex interactions and interfaces between previously independent components. It is not surprising, then, that the nature of accidents is changing and more are being attributed to software-related problems [9] or to dysfunctional interactions among system components (many of these interactions controlled by software) rather than to random individual component failure. Current hazard[1] analysis techniques were developed for the simpler, loosely coupled systems of the past and are becoming less effective as system complexity and coupling increases.

Because the primary cause of accidents in the older systems was component failure, the hazard analysis techniques and safety design techniques focused on identifying critical components and either preventing their failure (increasing component integrity) or providing redundancy to mitigate the effects of component failure. One standard approach is to identify critical components (fault hazard analysis) at the system level. In the automotive and commercial aerospace communities, once the software component has been assigned a criticality level, the component is implemented using varying levels of assurance techniques depending on the level assigned to it. Often, no special hazard or safety analysis is applied to the software (and sometimes not even on the hardware components). In contrast, in the defense and space world and occasionally in other industries, hazard analysis is commonly applied at the system level and used to identify

Manuscript received August 10, 2009; revised December 7, 2009. Current version published March 31, 2010. This research was partially supported by JPL University Subcontract 1297013 and NSF grant 0527660.

M. V. Stringfellow and **N. G. Leveson** are with the MIT Complex Systems Research Laboratory, Cambridge, MA 02139 USA (email: mstring@alum.mit.edu; leveson@mit.edu).

B. D. Owens was with the MIT Complex Systems Research Laboratory, Cambridge, MA 02139 USA. He is now at the University of California Space Sciences Laboratory, Berkeley, CA 94720 USA (e-mail: owensbd@alum.mit.edu).

Digital Object Identifier: 10.1109/JPROC.2009.2039551

[1]We define a hazard as, "A system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to an accident (loss) [1]."

system hazards, which are used to guide system design to eliminate or control the hazards. But once again, rarely is anything done other than applying good software engineering techniques to the software.

There are several limitations of these approaches. First, current hazard analysis techniques essentially treat software functions as if they were hardware functions, assuming that they will fail randomly and that unreliability of the software will be the major software problem. In fact, however, the vast majority of software-related accidents have involved errors in the requirements, not failures of the software to correctly implement the requirements, i.e., the software was working completely correctly and reliably with respect to the requirements provided to the software developers [7].

A second major problem is that most common hazard analysis techniques, e.g., fault tree analysis, FMECA (failure modes and effects criticality analysis) or HAZOP, work on an existing design. But systems and system designs have become so complex that waiting until a design is completed to perform a safety analysis on it is impractical. Even if by dint of sheer will it is possible to perform such an analysis, changing the design after the fact is usually impractical (financially and intellectually). Much of the safety effort, therefore, goes into proving that existing designs are safe rather than building designs that are safe from the beginning. The only hope for practical and cost-effective safe design approaches in these systems is to design safety in from the beginning. In *safety-driven design*, the information needed by the designers to make good decisions is provided to them *before* they create the design and the analyses are performed in parallel to the design process rather than after it. We believe that such design approaches will not only cost a lot less but will result in much safer systems.

In this paper, we describe a new hazard analysis approach, and its use in safety-driven design. Because current hazard analysis techniques start from a completed design and assume that accidents are caused by component failures (which is not true for software), a new approach to hazard analysis is needed. That, in turn, requires an expanded model of causality upon which the hazard analysis technique can be based. In this paper, we briefly describe the expanded causality model (STAMP), which has been described in detail elsewhere [10], and provide a detailed description of a new hazard analysis method called STPA (Systems Theoretic Process Analysis) that can be used to handle software contributions to accidents in existing systems, or can be used to create safety-driven design for new systems. A detailed example, using an aerospace system (a planetary lander) is provided.

II. CURRENT STATE OF THE ART

Current hazard analysis techniques are basically bottom-up or top-down. Bottom-up hazard analysis techniques, such as FMECA, hypothesize component failures and determine whether the failures will result in system hazards. This approach quickly becomes impractical for hardware components when there are thousands and perhaps tens of thousands in the design. And it ignores problems that occur due to multiple component failures (and all their potential combinations) as well as hazards arising from dysfunctional interactions among non-failing components, i.e., interface errors. Attempting to identify all possible incorrect behavior of software and tracing that to the system behavior is clearly impractical as software can potentially do thousands and perhaps millions of things wrong.

Top-down hazard analysis methods, such as Fault Trees, while working better than bottom-up techniques for relatively simple systems, also explode in size for complex systems and quickly become impractical for systems with functionally complex software components. Because of the impracticality of doing anything else, usually a box is assigned in the fault tree to software with the content "software fails" or "software error" and then some arbitrary failure density function is assigned to the box [13]. This numerical information provides no useful input to the designers or implementers of the software (besides being unmeasurable and thus impossible to derive) and appears to be useful (even if it made sense) only in an after-the-fact certification argument.

Something new is clearly needed to maintain the same levels of safety as we increase the complexity of our designs and systems.

III. AN EXPANDED MODEL OF ACCIDENT CAUSALITY BASED ON SYSTEMS THEORY

Current models or assumptions about the cause of accidents are based on reliability theory and assume that accidents arise from failures of system components. However, as we have noted many times before, most software-related losses arise from requirements errors, i.e., the software does not fail in the same sense as hardware, but instead it operates exactly as intended by the designers [7]. These flawed requirements usually involve either a) incomplete or wrong assumptions about the operation of the controlled system or the required operation of the computer or b) unhandled controlled-system states and environmental conditions. Merely trying to get the software "correct" (satisfy its requirements) or to make it reliable will not make it safer under these conditions. "Correct" or "reliable" software operation may, in fact, result in unsafe system states if:

- The software correctly implements its requirements but the specified (required) behavior is unsafe from a system perspective;
- The software requirements do not specify some particular behavior required for system safety (they are incomplete); or

- The software has unintended (and unsafe) behavior beyond what is specified in the requirements.

While these conditions may also be true for hardware, we can usually thoroughly test hardware and remove the requirements and design errors before using the system. Thus the only remaining problems almost always involve some type of component failure. But software can only be tested partially prior to use and only a small part of the potential software behavior can be validated to be safe through testing. In complex systems, formal verification methods to validate the safety of the software under all the conditions listed above are not practical.

The safety of the operating software in a particular system is also a function of the algorithm implemented, not simply whether there are no coding or implementation errors in that algorithm. As an example, consider a simple altitude switch that issues a signal when a particular altitude is reached. In this example, safety involves more than simply getting the software correct: a different algorithm is required for safety depending on how the switch is used by the encompassing system. For example, assume that there are three altimeters reporting the current altitude to the switch (redundancy is needed because of the possibility of a hardware altimeter failure). If the signal sent by the altitude switch is *safety increasing*, i.e., the signal is used to maintain safety such as lowering the landing gear in an aircraft, then the algorithm used in the altitude switch should require that any of the three altimeters report reaching the threshold before issuing the signal. The safety of the system will be compromised if the signal is not sent when the altitude has been reached. On the other hand, if the signal is *safety decreasing*, e.g., the signal is used to fire a missile, the algorithm should probably require all three altimeters to report the altitude as reaching the threshold. Safety in this second case is compromised by inappropriately sending the signal when the altitude has not been reached. Safety, in this example, is not just dependent on the software executing correctly but depends on the appropriateness of the particular algorithm used with respect to the safety of the entire system.

To handle software, the simple accident causality model based on component failure that was developed for relatively simple electro-mechanical systems needs to be changed. STAMP (Systems-Theoretic Accident Model and Process) is a new causality model that expands the potential causes of accidents considered.

Rather than having reliability theory as its foundation, STAMP is based on systems theory. In systems theory, focus is on systems as a whole, not on the parts taken separately. There are some properties of systems that can only be treated adequately in their entirety, taking into account all the social and technical aspects of these *system* or *emergent* properties. The properties derive from the relationships among the parts of the systems, i.e., how the system components interact and fit together.

Complex systems can be modeled as a hierarchy of organizational levels, where each level is more complex than the one below and the levels are characterized by *emergent* properties, which are irreducible and represent constraints on the degree of freedom of the components at a lower level of the hierarchy. Safety is such an emergent property—it is not a component property. In the altitude switch example above, a particular implementation (algorithm) may be safe in one system and unsafe in another, depending on how the component is used and the design of the system as a whole. Thus safety can only be analyzed in the context of the whole.

Every controller must contain a model of the system, plant or process it is controlling [1]. For example, a thermostat contains a model of the controlled process (room or vehicle temperature) that includes the current room temperature, the desired temperature, and some simple control laws about how temperature can change. In a human, we often call this model a mental model. Accidents often occur when the process model used does not match the true state of the process and incorrect control commands are given, correct ones are not given, correct commands are given at the wrong time (too early or too late), or control stops too soon. Most software-related accidents result from these types of mismatches between the model of the process being used by the software and the real process state. For example, the software thinks the spacecraft has reached the planet surface (when it has not) and turns off the descent engines prematurely, which is what is thought to have happened with the Mars Polar Lander [5], [22].

How do the process model and the process become inconsistent? The model may be wrong from the beginning, there may be missing or incorrect feedback to update the model as the process changes, the updating algorithm may be incorrect, or the control algorithm may not properly account for time lags in the control loop. The result may be uncontrolled disturbances, unhandled process states, inadvertent commanding of the system into a hazardous state, or unhandled or incorrectly handled system component failures. Note that this theory includes component failure as a causal factor in accidents, but it is only one aspect of causality.

Thus, in STAMP, safety is viewed as a dynamic control problem rather than simply a component failure or component reliability problem. For example, the O-ring did not control the propellant gas release in the Challenger spacecraft loss by sealing a gap in a field joint. The software did not adequately control the descent speed of the Mars Polar Lander. The events that occur prior to and during an accident, while clearly important in understanding the causality, are the result of the inadequate control. They result from lack of enforcement of the safety constraints (constraints on the system state or behavior required to ensure safety) by the system design or by operations. Accidents occur when the control structure or control

actions do not enforce the safety constraints, resulting in unhandled environmental disturbances or conditions, unhandled or uncontrolled component failures, or unsafe interactions among system components.

The control structure is a graphical representation of the control flow between the system elements. Enforcement of the safety constraints is accomplished by decomposing the system into a hierarchy of abstract controllers. Each controller levies constraints to lower-level controllers in order to control their behavior. At each level of functional decomposition, each functional element is assigned responsibility for the control of the functional interactions within the element while one hierarchically superior element is assigned responsibility for control of the interactions across elements. Note that these assignments are not necessarily a description of the software and hardware architecture, but a representation of the functions the system must perform and how the functions are related to each other. Using the results of the functional analysis, a high-level system control structure is designed. A more in-depth discussion of control structures can be found elsewhere [13].

An example system control structure for a spacecraft is shown in Fig. 1. In the example (and in current standard spacecraft architectures), interactions between spacecraft functional elements are controlled by the spacecraft command and data handling functional element. The control structure can be evolved iteratively to capture lower-level interactions and inform the lower-level design as will be discussed later in the article.

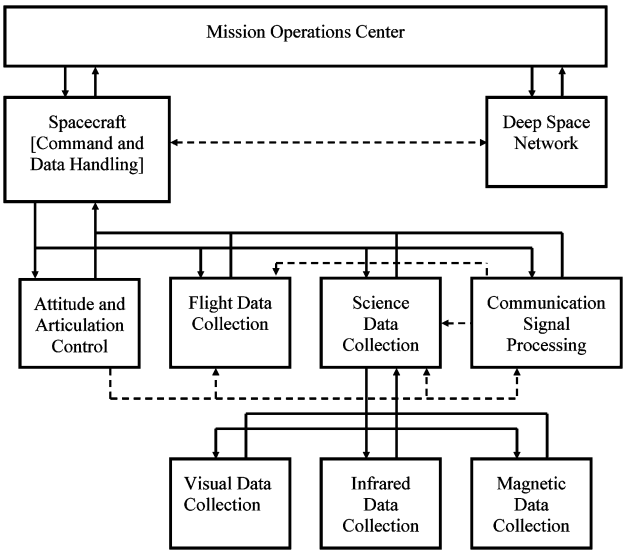

Fig. 1. ***Control Structure. Solid arrows down represent control in the form of directive(s) or command(s). Solid arrows up represent feedback in the form of sensor measurement(s). Dashed lines represent physical or informational interaction other than control or feedback interactions.***

A second source of risk occurs when the safety control structure degrades or changes over time. Systems tend to migrate toward states of higher risk under competitive pressures and economic constraints [19] or because of other factors such as complacency when no losses have occurred for a while. These unsafe changes must either be handled in the design or detected and corrected during system operation.

A third cause of accidents is related to the occurrence of multiple controllers in a system where the control actions are inadequately coordinated among the controllers. For example, two aircraft collided over southern Germany in 2002 when one pilot followed the instructions given by the ground air traffic controller while the other pilot followed the instructions provided by TCAS, an airborne collision avoidance system. The two sets of instructions were not coordinated, resulting in the deaths of all aboard the two aircraft.

Using this causality model for accidents, a more powerful hazard analysis technique can be created that is effective not only for hardware components of systems but for software components as well, as described in Section V.

IV. STPA AND SAFETY DRIVEN DESIGN OVERVIEW

An important advantage of the new causality model and new hazard analysis technique on which it is based, is that hazard analysis does not require a completed design. The technique can be applied as soon as the high-level system accidents and hazards are known. The first design decisions are made on the basis of high-level system safety constraints derived from the system hazards, at which point the design and the hazard analysis are refined in parallel, each driving the other in an iterative process.

The name of the new hazard analysis technique is STPA (Systems Theoretic Process Analysis). The objectives, as described in [10], are to identify instances of inadequate control that could lead to the presence of hazards and the safety-related constraints necessary to ensure acceptable risk. Furthermore, performing STPA produces information about how the safety constraints may be violated. This information can be used to control, eliminate, and mitigate hazards in the system design and operation. Although the first steps of STPA are similar to those performed in other hazard analysis techniques, the later steps either deviate from traditional practice or provide a rigorous framework for doing what is traditionally done in an ad hoc manner.

The first step in safety-driven design is to identify accidents or unacceptable loss events, such a loss of vehicle, loss of life, or a great deal of money. The next step is to define the hazardous states in the system that would allow accidents to occur. These hazards are then translated into safety constraints on system state and behavior so that

the hazardous states cannot occur. For example, the translation of hazard to safety constraint in an automated train door controller is simple:

Hazard 1. A person is present in the doorway when the doors are closing.

The related system-level safety constraint is:

Safety Constraint 1: Train doors must not close while anyone is in the doorway.

Various types of design features could be used to enforce this constraint, such as motion detectors, mechanical interlocks, etc. Choosing amongst possible safety-related design decisions to enforce Safety Constraint 1 will clearly affect the cost and performance of the overall system. By considering the safety impact of design decisions early, engineers can make trades between safety, performance, and cost in an informed fashion, rather than suffering unexpected costs when trying to add on safety (often in the form of additional fault protection) to an already complete design.

After the high-level hazard analysis is created, the overall control structure of the system is designed. For a spacecraft, this structure may contain logical components like attitude and articulation control, science data collection, communications signal processing, propulsion, etc. Once the high-level design is completed, the safety design process proceeds iteratively using STPA throughout the layers of abstraction in the control structure. The required safety constraints for system components can be generated, and then design decisions made to enforce those constraints or the safety constraints may be refined to a lower level of abstraction. As design decisions are made and safety constraints identified, they are analyzed with STPA. The process continues to iteratively refine and generate ever more detailed constraints and design decisions in parallel. Just as in any system engineering process, requirements, or shall statements, are levied on the system in order to accomplish the system-level goals. Constraints, on the other hand, are requirements levied on the system to constrain system behavior as it seeks to satisfy system goals and fulfill requirements. In safety-driven design, requirements are decomposed from system level goals and are created in parallel with safety constraints. More details about this safety-driven process can be found in [21] and [17]. An example is provided in the next section.

V. DETAILED DESCRIPTION OF STPA AND ITS USE IN SAFETY DRIVEN DESIGN

Underlying the STPA process is the notion that hazards are eliminated or controlled through system design. Fig. 2 presents a generic, low-level process control loop in STPA. As seen in the figure, the control input is a reference signal. The controller uses the control input, in conjunction with received measurements, to generate commands. Continuing along the loop, the command is sent to the actuator, which implements the command through the arrow labeled U. The U vector refers to actions of the actuator that influence the controlled process. The control algorithm used by the controller is based on an internal process model of the controlled process. The controlled process, or plant, is subject to process inputs and disturbances. The process output may become an input into another linked process control loop. The sensors measure the output resulting from the actuator's actions and disturbances, and generate measurements that are then fed into the estimator.

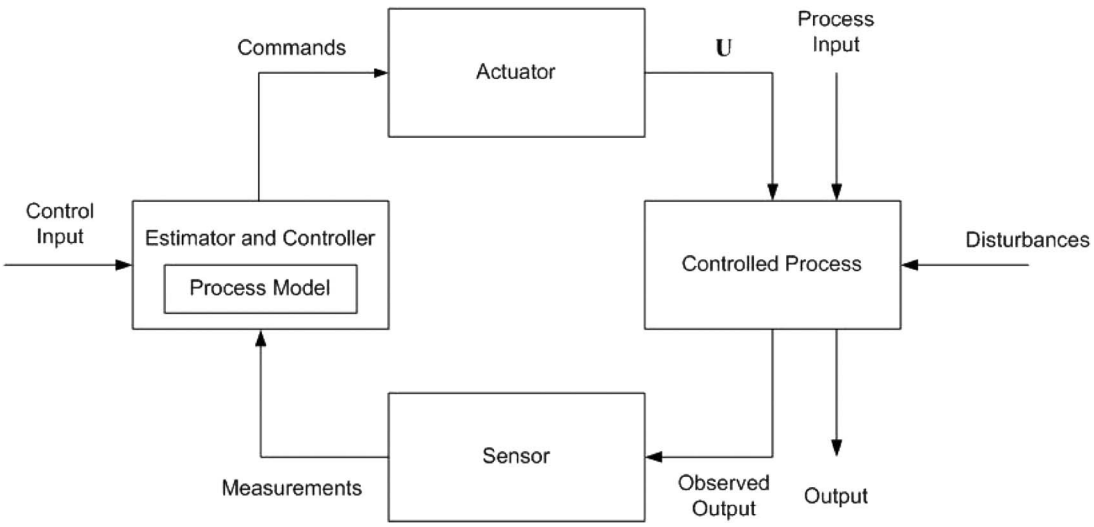

Fig. 2. *Generic STPA low-level process control loop.*

Depending on the particular system, the control input may be referred to as a goal, plan, sequence, directive or set point in spacecraft or automotive engineering parlance. The controller may send directives to a lower-level controller rather than an actuator in order to affect control on that process. Similarly, the lower-level control loop, rather than a sensor, may pass measurements or status information (such as its health and other components of its current state) to the higher-level control loop.

STPA, as described above, is based on the concept of controlling hazards rather than eliminating component failures (which are only one cause of hazards). When a safety constraint is violated, hazardous states arise and accidents can occur. For example, if the physical process being controlled is the landing of a spacecraft, one potential *accident* is that the spacecraft crashes into the surface of the planet and is consequently destroyed. The related *hazard* (H1) is that the spacecraft comes into contact with the planetary surface in an uncontrolled descent. The spacecraft could be inadequately controlled and the hazardous state could occur if, for example, the landing controller directs the thrusters to turn off early. A control flaw, such as an incorrectly calibrated velocity sensor, could contribute to inadequate control of the landing process. For this example, the system-level safety constraints (SC 1 and SC 2) and some related design decisions (DD 1, DD 2, and DD 3) to control them are:[2]

SC 1: The spacecraft must control its terminal descent to the surface of Mars. (↓ DD 1, DD 2, DD 3)

SC 2: The spacecraft must be protected from impact with the surface. (↓DD 2)

Rationale: The spacecraft structure is susceptible to damage from rocks even at low impact velocity.

DD 1: Thrusters on the spacecraft will be used to provide reverse thrust and slow the spacecraft descent. (↑ SC 1)

DD 2: When the spacecraft is within TBD meters of the planet surface, pressurized, gas-filled balloons will inflate around the spacecraft to protect the spacecraft structure during the impact of landing. (↑SC 1, SC 2)

DD 3: A vertical velocity sensor will measure spacecraft velocity during descent and ensure that reverse thrust is not stopped prematurely. (↑SC 1)

[2]The intent specification structure [8] we use for specifying systems includes both the rationale for design decisions (denoted in the italics) and links (denoted by arrows in the text) to ensure complete traceability between requirements and design. Often, the rationale for a design choice or requirement goes unrecorded. As the project evolves or time passes, the perhaps once-obvious rationale for a design decision is lost. In addition, it may be difficult to identify the parts of the design related to a particular safety constraint when changes are required. The discovery of new environment parameter values could require changes in all the design affected by an assumption made using outdated values, and engineers must be able to find all the affected parts of the design to safely make the necessary changes. It is critical that rationale be made obvious and that safety-related constraints and decisions are traceable to the parts of the implementation they affect.

Each safety constraint is analyzed using STPA. Starting from the safety constraint to control the hazard, *inadequate control actions* that could violate the safety constraint are identified. An inadequate control action is an action or inaction by the controller that leads to the violation of a safety constraint. The four possible types of inadequate control are:

1) A required action is not provided or is inadequately executed.
2) An incorrect or unsafe action is provided.
3) A potentially correct or adequate control action is provided at the wrong time (too late or too early).
4) A correct control action is stopped too soon or continued too long.

The analyst takes each of the safety constraints and, using these four general types of inadequate control, identifies the specific types of inadequate control that could be related to each safety constraint. For example, for SC 1:

SC 1: The spacecraft must control its terminal descent to the surface of Mars. (↓DD 1, DD 2, DD 3)

ICA 1: Spacecraft descent control is not engaged.

ICA 2: Spacecraft disengages descent control.

ICA 3: Spacecraft descent control is activated too late.

ICA 4: Spacecraft descent control is de-activated too soon.

The next step in the STPA process is the identification of *control flaws* and *inadequate control executions* in the current system design. Control flaws are the mechanisms that could lead to inadequate control actions due to errors in the control algorithm, poor understanding of the process, or poor coordination between multiple controllers. Control flaws are identified through inspecting the process control loop to determine how the system can produce an inadequate control action. For example, the inadequate control action "Spacecraft descent control is not engaged" may result if a control input to initiate descent is not received by the Mars Lander Estimator and Controller. The general types of control flaws are:

1) Design of the control algorithm does not enforce constraints
   - Flaw(s) in creation process
   - Process changes without appropriate change in control algorithm (asynchronous evolution)
   - Incorrect modification or adaptation
2) Process models inconsistent, incomplete, or incorrect
   - Flaw(s) in creation process
   - Flaw(s) in updating process
   - Inadequate or missing feedback
     - Not provided in system design
     - Communication flaw

- ○ Time lag
- ○ Inadequate sensor operation
- Time lags and measurement inaccuracies not accounted for
- Expected process inputs are wrong or missing
- Expected control inputs are wrong or missing
- Disturbance model is wrong
  - ○ Amplitude, frequency, or period is out of range
  - ○ Unidentified disturbance

3) Inadequate coordination among controllers and decision makers

Inadequate execution of a control action may occur because of a communication flaw (such as a communication line failure), a failure of the mechanisms that actuate control (such as a motor failure), or time lags (such as the sluggish response of the motor; perhaps an indication that the motor is soon to fail). In other words, inadequate control execution can occur when the process model is correct and the correct control action is selected, but the control action is not successfully applied due to inadequate actuator or sensor operation, time lag, or a communication flaw. For example, if the descent sensors fail, the proper measurements will not be received by the Mars Lander landing controller, which may lead to the spacecraft not limiting and controlling the descent velocity. The inadequate control execution types are:

1) Communication flaw
2) Inadequate actuator operation
3) Time lag

Fig. 3 shows some of these flaws superimposed on a control loop. STPA involves drawing the control loops and applying the types of control flaws and inadequate control execution to a particular safety constraint and thus identifying the ways they could occur.

Continuing the Mars Lander example, inadequate control actions, control flaws, and inadequate control executions can be identified for SC 1:

SC 1: The spacecraft must control its terminal descent to the surface of Mars. (↓DD 1, DD 2, DD 3)

ICA 1: Spacecraft descent control is not engaged.

CF 1.1: The spacecraft controller does not receive an initiate descent control input.

CF 1.2: The spacecraft controller does not command the descent actuators to activate.

CF 1.3: The descent actuators do not receive a command to activate.

ICE 1.1: The controller sends the engage command to the descent controller at the right time, but the descent actuators fail.

ICA 2: Spacecraft disengages descent control.

CF 2.1: The descent controller receives incorrect feedback from a velocity sensor.

ICA 3: Spacecraft descent control is activated too late.

ICA 4: Spacecraft descent control is de-activated too soon.

Once the sources of inadequate control have been identified, the associated hazard can be eliminated or

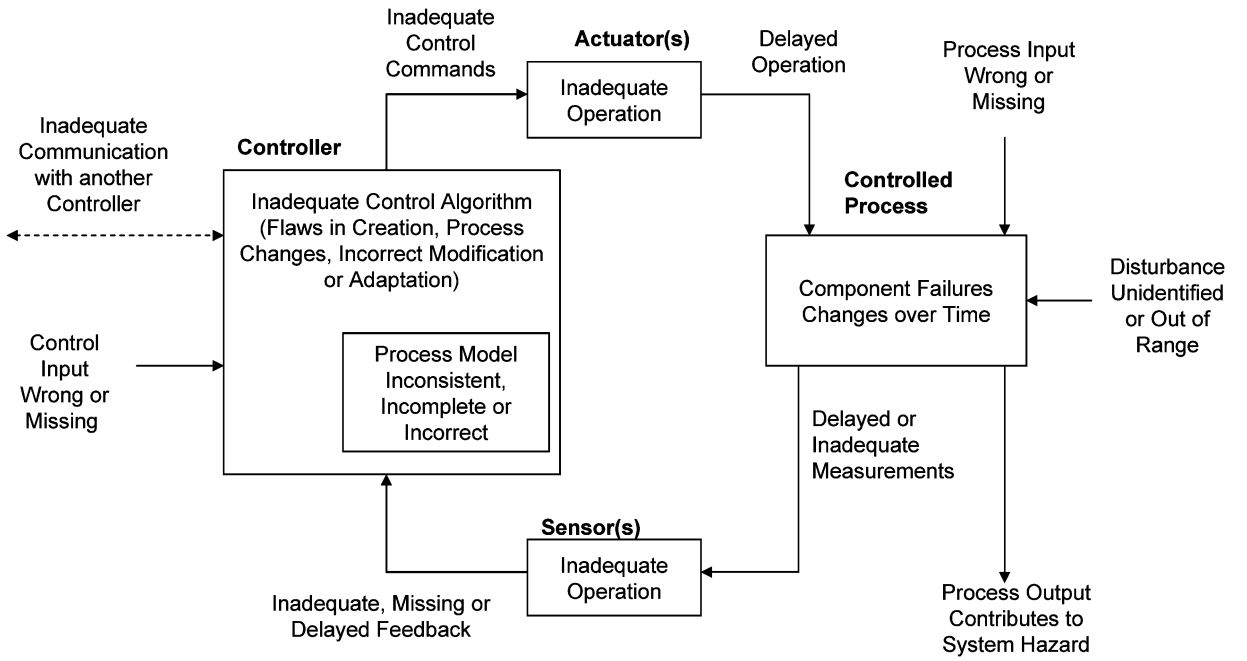

Fig. 3. ***STPA analysis superimposed on a control loop.***

controlled through design or, if that is not possible, through operations. In the safety-driven design process, the engineer may:

1) Create a new safety constraint, modify the related safety constraint, or refine the related safety constraint to better enforce control.
2) Create new design or modify existing design to eliminate, prevent, or mitigate the effect of the control flaw or inadequate control execution.
3) Accept the design as is and record the rationale for doing so and, if possible, suggest ways to control the hazard through operations.

The following example illustrates the refinement of two safety constraints and the design decisions made to enforce them. Again, pointers show traceability among the hazards, design constraints, and design decisions along with a recording of additional rationale underlying the decision.

SC 1: The spacecraft must control its terminal descent to the surface of Mars. (↓DD 1, DD 2, DD 3)

SC 1.1: The spacecraft controller's estimated velocity must be accurate to within 0.2 m/s. (↓DD 4)

DD 4: The spacecraft's velocity is calculated using a measurement device with accuracy of $+/-0.05$ m/s. (↑SC 1.1)

SC 2: The spacecraft must be protected from impact with the surface. (↓DD 2)

Rationale: The spacecraft structure is susceptible to damage from rocks even at low impact velocity.

SC 2.1: Impact balloons must be capable of withstanding contact with rocks at speeds of up to 10 m/s. (↓DD 5)

Rationale: The reverse thrusters are only capable of landing the spacecraft at a speed in the range of 2–10 m/s. There must be a system in place to lessen the impact of landing.

DD 2: When the spacecraft is within TBD meters of the planet surface, pressurized, gas-filled balloons will inflate around the spacecraft to protect the spacecraft structure during the impact of landing. (↑SC 1, SC 2)

DD 5: The balloons must be inflated to a TBD pressure and made of a tear-resistant material. (↑SC 2.1)

SC 1.1 is a refinement of SC 1, and was created to prevent ICA 1 and ICA 4. In this case, one new safety constraint (and a new design decision to enforce the new safety constraint) helps to eliminate two inadequate control actions.

After the high-level system design is complete, a functional analysis is performed to assign system functions to components and the safety constraints on the components must be generated. These will be used, along with STPA, in making component design decisions. The safety-driven design process is an iterative one, with each subsystem design leading to further refinement of the requirements and constraints, further application of STPA, and further system design.

The results of STPA are documented in the system hazard log. Fig. 4 shows part of an example hazard log. For each high-level hazard, the functional element pertaining to the hazard is listed as well as the relevant operation or mission phase. The causal factors shown in the hazard log are pointers to the control flaws identified in the STPA analysis. The hazard log usually also captures other information such as the hazard severity and type of potential loss resulting from the hazard.

The examples of using the control loops and the Control Flaw taxonomy to find control flaws in the design presented above are by no means complete. The STPA taxonomy is a useful guide in the discovery of control flaws and provides a rigorous basis for categorizing control flaws; however, the discovery of control flaws always relies on domain knowledge. Only the proper application of domain knowledge to a design can ferret out how instances of inadequate control can arise. This limitation is, of course, true for any hazard analysis technique. System engineers will need to collaborate with other experts in order to analyze system and subsystem designs for control flaws.

STPA and safety-driven design should be performed iteratively and opportunistically. Engineers can either drill down into a particular hazard they wish to control or apply STPA more broadly across several hazards. In the early stages of design, few design decisions have been made and control flaws and inadequate control executions may not yet be identified. However, performing STPA early will allow the results of the hazard analysis to inform the design process. Note that iteration of the design via STPA can cause high-level products and design decisions to change. Feedback to the first stages in the system engineering process can occur as engineers create new requirements and constraints as a result of STPA. Attempts to control hazards in the design may inspire engineers to modify system-level goals, constraints, or programmatic considerations. Tracing and linking the safety analysis and the design decisions will help mitigate the negative impact of such changes.

In summary, STPA starts with a hazard and its related requirements or constraints. The STPA taxonomy is used to identify inadequate control actions and the control flaws and inadequate control executions that lead to inadequate control actions. From there, in safety-driven design, engineers create new constraints or refine the existing constraints and create new design or modify the existing design until all hazards are eliminated, mitigated or controlled. A chart describing the process can be seen in Fig. 5. Engineering judgment is used to determine when the design is "safe and complete enough."

VI. RELATED RESEARCH

Most other attempts to perform hazard analysis on software have involved extending the current techniques

H 2: Inability of Mission to collect data.

System Element(s): Spacecraft, Mars Lander, Launch Vehicle, and Mission Operations Center (MOC)

Operation/Phase: Pre-Launch, Post-Launch/Pre-Mars-Descent, Mars Exploration, Disposal

Causal Factors:

Spacecraft CF 1.1: The spacecraft does not make orbit correction maneuvers as often as required. (Around once every 24 hours.)
Spacecraft CF 2.1: A change in the state of the spacecraft or spacecraft environment that invalidates assumptions of directives occurs during the time delay between MOC transmittal of the directives and spacecraft reception of directives.
Spacecraft CF 3.1: The MOC directives are lost or corrupted in transmission to the spacecraft.
Spacecraft Communications Unit CF 1.1: The pointing of the spacecraft is not sufficient for transceiving.
…

Level and Effect: Potential loss of data collection opportunities. Loss of mission.

Safety Constraints:

SC 1: The spacecraft must have the necessary functionality for data acquisition at the required times.
SC 1.1: The spacecraft data controller must validate the time and state dependent assumptions of directives generated by the MOC.
SC 1.2: In the absence of valid MOC Directives, the spacecraft must initiate the proper functionality for reception of MOC Directives.
…

Fig. 4. ***Partial hazard log example.***

to include software e.g., [2], [4], [14], [15]. In these approaches, high-level hazards are identified and functional failure analysis is performed on the completed high-level system design to find system failure modes. An augmented traditional hazard analysis technique is then used to trace the failure events to the software components (which implies the software architecture has also already been defined). Finally, fault protection is then created to mitigate the failure events or reduce its likelihood.

As we have argued, these techniques will have limited success because they omit the basic causes of software contributions to accidents [6]. Considering only scenarios where software does not fulfill its requirements, i.e., it fails, would have missed most of the cases where software has in the past been a major contributor to accidents. There needs to be a way to perform a hazard analysis that includes requirements errors and design errors, not just component failure. And such a technique should allow creating the design in hand with the analysis—changing design elements after the major design decisions have been made is almost always very costly and usually less effective than if the designers had considered the safety constraints from the beginning.

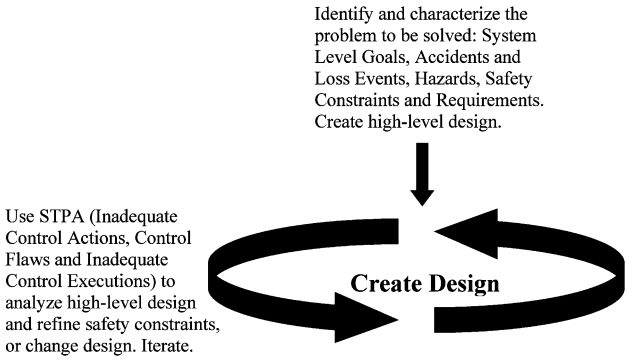

Fig. 5. ***Safety-driven design process.***

VII. DOES THIS WORK?

It is difficult to perform experimental analyses of hazard analysis techniques due to the scarcity of accidents. In the 50 years of modern safety engineering, almost no such analyses have been performed. The few that have been done have not shown the techniques to be very successful [7].

It is possible, however, to compare the scenarios generated by the analyses. We have compared STPA and fault tree analysis using TCAS, a collision avoidance system required on most commercial aircraft (more than 15 passengers). TCAS was chosen because an excellent fault tree exists (and was done by people other than those doing

the STPA, thus eliminating some potential bias in the comparison), and the TCAS fault tree used is one of the best the authors have ever seen for any system, particularly those having large amounts of software. In this comparison, STPA generated all the potential scenarios that are in the fault tree but also generated additional scenarios, at least one of which resulted in an aircraft collision and great loss of life. These additional scenarios were not included in the fault tree because the losses did not involve component failure but instead resulted from unsafe interactions among components [11].

Another way to determine whether something works is to try to use it on a very complex system and to evaluate the results, although this approach is less satisfactory than a carefully controlled experimental comparison or evaluation. STPA has been successfully used to perform a non-advocate safety assessment for the new U.S. Ballistic Missile Defense System (BMDS). STPA was selected after it was determined that it was not practical to try to use fault trees or other common hazard analysis techniques on a system of such complexity [18]. The BMDS is an enormously complex "system of systems," some new and some existing for decades as parts of other systems (such as NORAD and early warning systems) and shipboard control systems such as AEGIS. Deployment and testing of the BDMS was delayed for six months due to the large number of plausible scenarios for inadvertent launch identified by STPA [18]. Such delays and related costs might have been averted if the hazard analysis had not been performed after design of the system had been completed.

The safety-driven design process has also been applied to a demonstration project at NASA's Jet Propulsion Laboratory (JPL) [16], [20]. This effort was largely conducted by two graduate students, with the assistance of spacecraft engineering domain experts from JPL, over the course of one year. Without benefit of a career in the space industry, the graduate students were able to hone in on design areas of concern to JPL stakeholders. Using the safety-driven design, the students were able to select a design among several generated that had a higher level of safety at lower cost and equal performance to others.

VIII. SUMMARY

This article has described an approach to safety-driven design using a new hazard analysis method called STPA, which is based on systems theory and on STAMP [10], a more comprehensive model of accident causality than the standard linear event-chain models. STPA considers the non-linear inter-relationships among events and system components rather than just linear cause-effect chains, it looks at the processes behind the events, and it includes the entire socio-technical system rather than just the hardware or physical process. The result is that STPA can identify more causes of accidents, particularly those related to software and human decision-making, and can be used early in concept formation and throughout system development to guide design for safety rather than simply evaluating designs after the fact. While only physical processes were considered in this article, we have successfully applied STPA to social and organizational structures and systems such as Space Shuttle Operations [12], the development process for the replacement for the Space Shuttle [3], and such diverse systems as pharmaceutical safety, food safety, hospital safety, and even corporate fraud.

We are currently working on more sophisticated ways to evaluate human errors and unsafe decision-making. We are also devising practical methodologies and tools for applying STPA in the workplace and expanding it to human-intensive systems. ■

Acknowledgment

Inputs on STPA have been provided over the past few years by Dr. Nicolas Dulac and other members of the MIT Complex Systems Research Laboratory. Inputs on an early version of the safety-driven design process were provided by Dr. Michael Ingham and Dr. Kathryn Weiss of JPL.

REFERENCES

[1] R. C. Conant and W. R. Ashby, "Every good regulator of a system must be a model of that system," *International Journal of System Science*, vol. 1, pp. 89–97, 1970.

[2] J. Dehlinger and R. R. Lutz, "Software fault tree analysis for product lines," in *Proc. 8th IEEE Symposium on High Assurance Systems Engineering (HASE '04)*, Tamp, FL, 2004, pp. 12–21.

[3] N. Dulac, B. D. Owens, N. G. Leveson, and J. S. Carroll, "A formal modeling approach to risk management in the development of space exploration systems," in *Proc. Int. Assoc. Adv. Space Safety Conf.*, Chicago, IL, May 2007.

[4] R. D. Hawkins and J. A. McDermid, "Performing hazard and safety analysis of object oriented systems," in *Proc ISSC*, Denver, CO, Aug. 2002.

[5] JPL Special Review BoardReport on the Loss of the Mars Polar Lander and Deep Space 2 Missions NASA Jet Propulsion Laboratory, Mar. 22, 2000.

[6] N. G. Leveson and P. R. Harvey, "Analyzing software safety," *IEEE Trans. Software Engineering*, vol. SE-9, no. 5, pp. 569–583, Sep. 1983.

[7] N. Leveson, *Safeware: System Safety and Computers*. Addison-Wesley, 1995.

[8] N. G. Leveson, "Intent specifications: An approach to building human-centered specifications," *IEEE Trans. Software Engineering*, vol. 26, no. 1, pp. 15–35, 2000.

[9] N. G. Leveson, "The role of software in spacecraft accidents," *AIAA Journal of Spacecraft and Rockets*, vol. 41, no. 4, pp. 564–575, Jul./Aug. 2004a.

[10] N. G. Leveson, "A new accident model for engineering safer systems," *Safety Science*, vol. 42, no. 4, pp. 237–270, 2004b.

[11] N. G. Leveson, "Model-Based Analysis of Socio-Technical Risk," Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep., ESD-WP-2004-08, 2004c.

[12] N. G. Leveson, N. Dulac, J. Cutcher-Gershenfeld, B. Barrett, J. Carroll, D. Zipkin, and S. Friedenthal, "Modeling, analyzing, and engineering safety culture," in *1st Int. Conf. Int. Assoc. Adv. Space Safety*, Nice, France, Oct. 2005.

[13] N. G. Leveson, *Engineering a Safer World: System Safety for the 21st Century*, 2009.

[14] O. Lisagor, J. A. McDermid, and D. J. Pumfrey, "Safety analysis of software architectures—"Lightweight PSSA"," in *Int. Conf. of the System Safety Society*, 2004.

[15] J. McDermid, *Software Hazard and Safety Analysis, in Formal Techniques in Real-Time and Fault-Tolerant Systems*. Berlin: Springer, 2002, pp. 23–34.

[16] B. Owens, M. Stringfellow, N. Leveson, M. Ingham, and K. Weiss, "A Safety-Driven, Model-Based System Engineering Methodology, Part II: Application of the

Methodology to an Outer Planet Exploration Mission, MIT Tech. Rep., 2007.

[17] B. D. Owens, M. S. Herring, N. Dulac, N. G. Leveson, M. D. Ingham, and K. A. Weiss, "Application of a safety-driven design methodology to an outer planet exploration mission," in *Proc. IEEE Aerosp. Conf.*, Big Sky, MT, Mar. 1–8, 2008, paper 1279.

[18] S. J. Pereira, G. Lee, and J. Howard, "A system-theoretic hazard analysis methodology for a non-advocate safety assessment of the ballistic missile defense system," in *Proc. AIAA Missile Sci. Conf.*, Monterey, CA, Nov. 2006.

[19] J. Rasmussen, "Risk management in a 2 dynamic society: A modelling problem," *Safety Science*, vol. 27, no. 2/3, pp. 183–213, 1997.

[20] M. Stringfellow, B. Owens, N. Leveson, M. Ingham, and K. Weiss, "A Safety-Driven, Model-Based System Engineering Methodology, Part I, MIT Tech. Rep., 2007.

[21] M. V. Stringfellow, "Safety-Driven System Engineering Process," S.M. Thesis, Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 2008.

[22] Y. Thomas, "Mars Program Independent Assessment Team Report," NASA, Mar. 2000, (Chairman).

ABOUT THE AUTHORS

Margaret V. Stringfellow is a doctoral student in the department of Aeronautics and Astronautics. Her research focuses on system safety in complex socio-technical systems. In particular, she is developing a hazard analysis method suitable for organizational design to enable better decision-making in risk and performance for system stakeholders, including management and operators. Her research interests include modeling and simulation, risk, system safety, human factors and software. Prior to her graduate studies, Ms. Stringfellow characterized UAV sense and avoid collision avoidance algorithms at Lincoln Laboratories. She has designed and implemented simulation of next generation deep space network antenna array at NASA's Jet Propulsion Lab. Ms. Stringfellow holds S.M. and S.B. degrees in Aerospace Engineering and an S.B. in Electrical Engineering from MIT.

Nancy G. Leveson (Member, IEEE) is a Professor of Aeronautics and Astronautics and a Professor of Engineering Systems at the Massachusetts Institute of Technology. She is also the director of the Complex Systems Research Laboratory at MIT. She is an elected member of the National Academy of Engineering (NAE). Prof. Leveson conducts research on the topics of system safety, software safety, software and system engineering, and human-computer interaction. In 1999, she received the ACM Allen Newell Award for outstanding computer science research and in 1995 the AIAA Information Systems Award for "developing the field of software safety and for promoting responsible software and system engineering practices where life and property are at stake." In 2005 she received the ACM Sigsoft Outstanding Research Award. She has published over 200 research papers and is author of a book, "Safeware: System Safety and Computers" published by Addison-Wesley. She consults extensively in many industries on the ways to prevent accidents. She has degrees in math, management, and computer science from the University of California, Los Angeles.

Brandon D. Owens is a flight dynamics analyst for the Space Sciences Laboratory at the University of California at Berkeley. His current work involves orbit determination, maneuver planning, and maneuver execution for the THEMIS spacecraft constellation, which is presently in an extended mission phase that will include the low energy transfers of two of the spacecraft into lunar orbit via two Earth-Moon libration points. He previously was a research assistant for the Complex Systems Research Laboratory at the Massachusetts Institute of Technology where his research involved the development of techniques to evaluate the effectiveness of system safety control structures. Prior to that, he was a research assistant for the W. W. Hansen Experimental Physics Laboratory at Stanford University where his responsibilities included mission planning and radiation anomaly investigation for the Gravity Probe B satellite mission. Before that, he served as a co-op student for United Space Alliance in Houston, Texas in the departments of Cargo Operations and Flight Control and Environmental Systems. He has a Ph.D. in Engineering Systems from MIT, an M.S. in Aeronautics and Astronautics from Stanford University, and a B.S. in Aeronautical and Astronautical Engineering from Purdue University.