A Systems-Based Analysis Method for Safety Design in Rocket Testing Controllers

by

Jeremy Paquin

B.S. Mechanical Engineering, United States Military Academy, 2009

Submitted to the Department of Aeronautics and Astronautics and the MIT Sloan School of Management in Partial Fulfillment of the Requirements for the Degrees of

Master of Science in Aeronautics and Astronautics

and

Master of Business Administration

In conjunction with the Leaders for Global Operations Program at the

Massachusetts Institute of Technology

June, 2019

Signature of Author Signature redacted

Department of Aeronautics and Astronautics
MIT Sloan School of Management
May 10, 2019

Certified by Signature redacted

Paulo Lozano, Thesis Supervisor
Professor of Aeronautics and Astronautics

Certified by Signature redacted

Roy Welsch, Thesis Supervisor
Professor of Statistics and Engineering Systems

Accepted by Signature redacted

Sertac Karaman, Chair of the Graduate Program Committee
Associate Professor of Aeronautics and Astronautics

Accepted by Signature redacted

Maura Herson, Assistant Dean, MBA Program
MIT Sloan School of Management

This page has been intentionally left blank.

A Systems-Based Analysis Method for Safety Design in Rocket Testing Controllers
by
Jeremy David Paquin

Submitted to the Department of Aeronautics and Astronautics and the MIT Sloan School of Management on May 10, 2019 in Partial Fulfillment of the Requirements for the Degrees of Master of Science in Aeronautics and Astronautics
and
Master of Business Administration

Abstract

Boeing is the prime contractor for building the National Aeronautics and Space Administration (NASA) Space Launch System (SLS) core stage for upcoming exploration missions beyond low earth orbit. Due to the rigorous demands of safety on crew-rated spacecraft, the entire vehicle undergoes captive hot-fire testing before being delivered to NASA for actual flight operations. The hot-fire test is controlled by a suite of computers used to control the rocket segment and critical infrastructure interactions during the test. The complexity of the software and hardware used to control the test makes it difficult for traditional safety approaches to identify potentially unsafe system interactions by focusing only on component failures rather than overall system interactions. Traditional chain-of-failure safety analyses and reviews take significant resources and time to conduct while leaving possible gaps.

This thesis discusses a method for analyzing safety of rocket test controllers by characterizing key indicators and developing a systems-based approach for hazard analysis using Systems-Theoretic Process Analysis (STPA).

A resulting case study is applied for examination of a portion of the rocket testing controller system for comparison to traditional chain-of-failure events analyses. Appling STPA in the case-study resulted in 83% of the total work time needed to complete a comparable "ascent phase" analysis using FMEA. The STPA results are the same or meet a similar intent to those resolved in the FMEA with not gaps between the two methods. The recommended mitigation and constraints resulting from STPA are arguably more intuitive than those of the FMEA.

Thesis Supervisor: Paulo Lozano
Title: Professor of Aeronautics and Astronautics

Thesis Supervisor: Roy Welsch
Title: Professor of Statistics and Engineering Systems

The author wishes to acknowledge
the Leaders for Global Operations Program
for its support of this work.

Acknowledgments

To my family and wife, Jessica, thank you for your enduring support in whatever tasks I set out to accomplish. You keep me focused and grounded, especially in your support during this project.

I would like to thank David Hahs from the Boeing Company and the Leaders for Global Operations staff, especially Ted Equi, for their support in helping to align this opportunity.

Thank you very much to the Boeing Company for hosting me on this project and their continuing support of the Leaders for Global Operations program at MIT. I owe thanks to my manager, Kevin Fogarty, who ultimately afforded me this opportunity and the ability to explore areas of interest within the Space Launch System program. He made me part of the team and challenged me to help where it was needed.

Thank you to my advisors at MIT, Paulo Lozano and Roy Welsch, for their support during the project in my pursuit of knowledge and growth.

Lastly, thank you to the broader NASA, Redstone Arsenal, and Boeing Defense, Space, and Security community members, whom there are too many to list, that interacted with me on professional basis no matter how seemingly complex or unusual my request.

This page has been intentionally left blank.

Table of Contents

Abstract ... 3

Acknowledgments ... 5

List of Figures ... 9

List of Tables ... 11

1 Introduction ... 13

- **1.1 Thesis Objective** ... 13
- **1.2 Project** ... 14
- **1.3 Research** ... 15
- **1.4 Major Findings** ... 16
- **1.5 Content Summary** ... 16

2 Background ... 19

- **2.1 Former and Current Industry** ... 19
  - 2.1.1 NASA and Disruption ... 19
  - 2.1.2 Shuttle Retirement ... 26
  - 2.1.3 Project Constellation ... 28
- **2.2 NASA and the Space Launch System** ... 28
  - 2.2.1 New authorization for Space launch system ... 29
  - 2.2.2 SLS Development ... 30
  - 2.2.3 The Current NASA Design Process ... 31
- **2.3 The Boeing Company** ... 38
- **2.4 Rocket Test and Control** ... 40

3 Literature Review ... 43

- **3.1 Sources of Hazards** ... 43
  - 3.1.1 Effect of Concurrent Engineering on Requirements for n-Order Subsystems ... 44
  - 3.1.2 Requirements Volatility Correlated to Lower Quality ... 46
  - 3.1.3 Requirements, Quality, and the Link to Safety ... 47
  - 3.1.4 Measuring Requirements Volatility ... 48
  - 3.1.5 Effect of Requirements Volatility on Schedule (Service Length) ... 49
  - 3.1.6 Queuing Theory and Service Length ... 51
  - 3.1.7 Discussion on Requirements Volatility as a Source of Hazards ... 56
- **3.2 Current Safety Standards** ... 56
- **3.3 Traditional Accident Models** ... 59
- **3.4 Systems-Based Hazard Theory** ... 61
- **3.5 Systems-Based Hazard Analysis Applicability** ... 69
  - 3.5.1 Mars Polar Lander Failure Application ... 69
  - 3.5.2 Boeing 787 Dreamliner Battery Failure Application ... 71

4 Hazard Source Data Analysis and Simulation Results 73

4.1 Requirements Data Collection 73
4.2 Requirements Data Analysis 75
4.3 Data Comparison 78
4.4 Modeling and Simulation Results 80

5 Analysis Approach – Systems-Theoretic Process Analysis 87

5.1 Description of STPA Methodology 87
5.1.1 Define Purpose of the Analysis 88
5.1.2 Model the Control Structure 89
5.1.3 Identify Unsafe Control Actions 90
5.1.4 Identify Loss Scenarios 91
5.2 STPA Implementation and Analysis Strategy 93

6 Case Study of STPA Application to SLS Rocket Test Controller 95

6.1 Purpose, Environment, and Boundary Definition 95
6.2 Identifying Losses, Hazards, and Constraints 95
6.3 Control Structure Modelling 97
6.4 Identifying Unsafe Control Actions 99
6.5 Identifying Loss Scenarios 105
6.6 Deriving System Requirements and Mitigations 110
6.7 Analysis and Results 110

7 Conclusion and Recommendations for Forward Work 113

7.1 Benefits 113
7.2 Challenges and Forward Work 114
7.3 Integrating STPA into Your System Engineering Process 116

List of Acronyms 119

Bibliography 122

List of Figures

Figure 1. The Lean Innovation Cycle [1] 23
Figure 2. Space Shuttle Probability of Catastrophic Loss of Vehicle and Crew [2] 27
Figure 3. Basic Systems Engineering "V-Model" [4] 31
Figure 4: The NASA Systems Engineering Design Process [5] 33
Figure 5. NASA Project Life-cycle takes a program from concept study through program closeout [5] 35
Number 6. NASA Program Tiered Organizational Hierarchy [5] 36
Figure 7. The implementing organization designs their end product based on a requirements flow-down [5] 37
Figure 8. Sub-tier components coming together to create an end product [5] 38
Figure 9. Test Controller Employees are made up of >81% Software Engineers 41
Figure 10. Requirements Propagation through an n-Order System 45
Figure 11. Schematic View of a Queuing System [14] 51
Figure 12. Process Flow Diagram used for Queuing Theory 52
Figure 13. Tandem Process Flow Diagram 53
Figure 14. G-G-N Queuing Theory Diagram 54
Figure 15. Sum of Service and Queue Length to Determine Work-in-Progress Length 55
Figure 16. Increasing Variability of Requirements (Volatility) Increase the Queue Size and Waiting Time 56
Figure 17. Emergent Properties Arise From Complex System Interactions [4] 63
Figure 18. Control Actions and Feedback Imposed on a System [4] 63
Figure 19. Basic Control Loop [4] 65
Figure 20. Mars Polar Lander Accident Illustration 70
Figure 21. Test Controller Requirements by Type 74
Figure 22. Test Controller Software Requirements Over Five Years (System and User Apps and Displays) 75
Figure 23. Test Controller Requirements Volatility over an Eight Month Window 76
Figure 24. Test Controller Software Requirements Volatility over a Seven Month Window. 77
Figure 25. Test Controller Requirements Volatility Overlay during an Eight Month Window 78
Figure 26. Schedule Changes for the Test Controller System over a Seven Month Window 79
Figure 27. Comparison of Software Requirements Volatility to Schedule Delays 80
Figure 28. Fitted GEV CDF of Requirements Changes 84
Figure 29. Schedule Delay Risk Simulation Results 85
Figure 30. Visualization of STPA Steps 1 through 4 [4] 87
Figure 31. STPA Iterative Methodology and Steps 88
Figure 32. STPA Step 1: Define the Purpose of the Analysis [4] 89
Figure 33. STPA Step 2: Model the Control Structure [4] 90
Figure 34. STPA Step 3: Identify Unsafe Control Actions [4] 90
Figure 35. Identifying Types of Unsafe Control Actions 91

Figure 36. STPA Step 4: Identify Loss Scenarios [4] 91
Figure 37. Identifying Types of Loss Scenarios 92
Figure 38. Systems-based Analysis of Unsafe Control Actions 92
Figure 39. STPA Case Study: High-level Environment Feedback Control Structure Model 98
Figure 40. STPA Case Study: Low-level Feedback Control Structure Model 99
Figure 41. Using metrics to minimize schedule and cost impacts. 115
Figure 42. Relative amount of time spent on different tasks [4]. 118

List of Tables

Table 1. NASA missions run over budget and behind schedule leading up to the 2000s [1] 20
Table 2. NASA Mission results leading up to the 2000 [1] 22
Table 3. Boeing Progress on SLS. 39
Table 4. Baseline Model Values for G-G-N Queuing Model with no Variation 81
Table 5. Results of G-G-N Queuing Model with no Variation 81
Table 6. Model Values for G-G-N Queuing Model with Variation Data 82
Table 7. Results of G-G-N Queuing Model with Variation Data 82
Table 8. Schedule Impact Results from Requirements Volatility 83
Table 9. STPA Case Study: Identification of System-level Losses. 96
Table 10. STPA Case Study: Identification of System-level Hazards 96
Table 11. STPA Case Study: Derivation of System-level Constraints from Hazards .. 97
Table 12. STPA Case Study: "Ascent Phase" Control Actions for a Rocket Test Controller 100
Table 13. STPA Case Study: Unsafe Control Actions Table. 101
Table 14. STPA Case Study: Controller Constraints Identified for each Unsafe Control Action. 103
Table 15. STPA Case Study: Determining Causes of Losses for each Unsafe Control Action 105

This page has been intentionally left blank.

Chapter 1

1 Introduction

This chapter gives a synopsis of the project motivation, objectives of the thesis, and provides an overview of the thesis approach.

1.1 Thesis Objective

Boeing is the prime contractor for building the National Aeronautics and Space Administration (NASA) Space Launch System (SLS) core stage for upcoming exploration missions beyond low earth orbit. Due to the rigorous demands of safety on crew-rated spacecraft, the core stage undergoes captive hot-fire testing of its four RS-25 engines and all of its systems before being delivered to NASA for actual flight operations on Exploration Mission-1 (EM-1). The hot-fire test is controlled by a suite of computers that control the vehicle and critical interactions with the test infrastructure. Traditional chain-of-failure safety approaches such as Failure Modes and Effect Analysis (FMEA) for analysis of such complex software and hardware focus on component failure rather than unsafe system interactions. A newer systems-based approach, System-Theoretic Process Analysis (STPA), may be applied for examination of the rocket testing controller system to improve efficiency and safety-engineering of hot-fire testing operations of the SLS core stage in future iterations.

With new space launch programs, the engineering and manufacturing organization must not only build the first rocket, but must construct a completely new system to design, build, and test a new rocket line. This includes methods of production and validation. Validation can be done in one of four ways: test (against measurable requirements), analysis, demonstration, or inspection. Testing is the most thorough and required for the SLS core stage final validation. The program components of the launch system are developed through concurrent engineering and understood iteratively as the project matures. Often, at the tail end of the evolving requirements churn is the rocket test and finalization of the supporting test equipment due to the nature of its reliance on vehicle specifications. Due to natural schedule constraints, the testing phase is often under the most condensed timeline pressure as preceding vehicle development encounters changes or delays. This environment of requirements volatility and intensifying schedule pressure leads to higher potential for introducing unsafe conditions into the launch vehicle testing controller as it is at the tail end of the requirements churn.

Full-scale hot-fires and the rocket launch itself are two of the most dangerous aspects of space launch development. The high-mix, low-volume production of such vehicles does not enable a universal approach to testing since each mission design requires extensive technical modification from one iteration of the vehicle to the next. Thus application of the same test controller is insufficient to apply to multiple platforms. Testing of the launch vehicle to ensure its safety and reliability is extensive, time-consuming, and expensive. The test controllers of such tests are critical to performing a safe hot-fire test and can be extremely complex. The safety of the launch vehicle test is a balance between applying appropriate corrective action to potentially unsafe actions of the test article system and avoiding erroneous test aborts due to improper or overly conservative safety constraints.

1.2 Project

The main focus and basis of research involves the test controller component of the captive hot-fire test, which encompasses the full suite of validation testing of the Space Launch System core stage, as controlled by the "Stage Controller" (SC) computer architecture and software.

Improving the ability to perform test firing of the current and future iterations of the Space Launch System may be approached in a way that minimizes schedule risk and cost by examination of the test controller using a newer system-based method of hazard analysis.

1.3 Research

This thesis will perform an examination of potential leading indicators that may introduce unsafe system interactions into the rocket test controller design and product. Next, it will develop a methodology for a system-based approach of rocket testing controllers, and then discuss methods of improving safety engineering from a systems-safety approach.

The examination method will utilize a model and data analysis approach to understand sources of introducing unsafe interactions into the final product. The thesis will utilize this model to develop a simulation to determine risk to indicators along with a discussion on strategy to deal with sources of risk.

A contemplation of traditional hazard analyses will examine merits and issues of various methods. A basis for a newer approach for systems-based hazard analysis, using Systems-Theoretic Process Analysis, will be introduced along with a detailed methodology for analysis.

A case study will apply a systems-based hazard analysis to the rocket testing controller during the "ascent phase" of the captive hot fire test. A discussion of the results will determine the merits and issues with the methodology.

Further discussion will introduce techniques to minimize leading indicators of inducing hazards and improving safety-engineering using a systems safety approach. This method will be applicable to a number of products that may be at the tail end of a concurrent engineering project such as rocket testing controllers.

1.4 Major Findings

Appling STPA in the case-study resulted in 83% of the total work time needed to complete a comparable "ascent phase" analysis using FMEA. The STPA results are the same or meet a similar intent to those resolved in the FMEA with no gaps between the two methods. The recommended mitigation and constraints resulting from STPA are arguably more intuitive than those of the FMEA. This is likely due to the methodology of STPA which employs logical systems-based reasoning to explore the system's emergent properties and all possible causes as opposed to FMEA's failed-component approach which focuses mainly on hardware component failures.

Analyzing the data for the overall development of the system over an eight month period reveals requirements volatility ranging from <1% to 26.8% for the total system requirements from all four categories (overall system, hardware, system software, and user apps and displays). This volatility represents a risk to program schedule and further increases risk to inducing defects that cause safety-critical hazards.

Probabilistic simulation results show that the probability of schedule delays due to requirements volatility is >64%. The combined effects of requirements volatility increases schedule risk and defect rates. Schedule risk may lead to other sources of hazards being introduced into the system by management attempting to make-up lost schedule time by directing employees to work overtime. This may further lead to compounding sources of induced hazards into the system during development, as shown in Section 3.1.

1.5 Content Summary

Chapter 1 of this thesis provides an overview of the project context and goals. Chapter 2 covers background for research and details the current space launch industry, NASA and the SLS program, and Boeing's history with the program. Chapter 2 also includes an overview of rocket testing and control approaches and methods. Chapter 3 covers the literature review, detailing the current engineering process, sources for introducing risk in concurrent engineering products, and methods for analysis of leading indicators. It also gives an overview of common methods for traditional hazard analyses and covers the philosophy behind system-based hazard analysis. Chapter 4 covers analysis of hazard sources being introduced into the design. Chapter 5 provides in-depth coverage of the procedures for Systems-Theoretic Process Analysis (STPA). Chapter 6 performs a case-study applying STPA to the rocket testing controller during the "ascent phase" of the SLS captive hot-fire test. Chapter 7 concludes with a discussion of the merits and challenges of these methods along with techniques to incorporate them into system design and safety engineering practices for future interactions.

In this thesis, several terms are used interchangeably, particularly with regards to the products being discussed, the SLS core stage and the rocket testing controller. The SLS core stage is referred to interchangeably throughout the thesis as the test article, rocket, vehicle, core stage, and SLS. While the most accurate description of the test article under testing is the SLS core stage, these terms refer to this section of the overall Space Launch System. The captive hot-fire test controller is referred to as the test controller, stage controller, and rocket testing controller.

This page has been intentionally left blank.

Chapter 2

2 Background

This chapter provides the necessary background for follow-on literature review and research discussion. The first section covers the current space launch industry and past human exploration launch vehicle development. Section 2.2 provides an overview of NASA and the SLS program's history. Section 2.3 covers the SLS prime contractor, the Boeing Company, and its relationships with NASA and its objectives. Section 2.4 provides a primer on the foundational components of rocket testing and control.

2.1 Former and Current Industry

From the perspective of space exploration as a current industry, many facets of historical and contemporary underpinnings have an effect the Space Launch System (SLS) program. NASA's history with successes and failures have some interesting implications on the manner of how NASA defines and approaches risk whether from a program success or safety perspective. The next sub-section discusses the retirement of the Space Transportation System (STS), typically referred to as the Space Shuttle, the reasons why it was deemed unsafe, and the need for a replacement capability to get beyond low earth orbit (LEO). Finally, the last sub-section provides context behind the start of the Constellation Project and the transition to the current authorized Space Launch System (SLS).

2.1.1 NASA and Disruption

In the years leading up to the turn of the century, NASA enjoyed a multitude of mission successes. The Viking program launched in 1975, landed successfully on Mars in 1976 on two separate missions. The immensely public success opened an entire scientific enterprise with the next steps ostensibly imminent. However, the subsequent decades endured major failures.

The Viking program was $5 Billion[1] running over 15 years from inception through operational completion. In 1977, Voyager, a $1 Billion program was successful, on time, and on budget. However, beginning with Galileo in 1986, a trend developed with cost overruns and schedule delays as shown in the table below [1].

Table 1. NASA missions run over budget and behind schedule leading up to the 2000s [1].

Program	Year	Cost[1]	Years to build	Issues
Voyager	1977	$1 Billion	5 Years	On time, on schedule
Galileo	1986	$3 Billion	10 years	3x cost overrun, 7 years behind
Mars Observer	1992	$2 Billion	8 years	2x cost overrun, 4 years behind
Hubble Space Telescope	1993	$3 Billion	15 years	6x cost overrun, 6 years behind

[1] All costs are adjusted to 2018 dollars

Cassini/CRAF	1997	$5 Billion	7 years	2x cost overrun, 3 years behind

These programs were technically and scientifically sound, well-staffed with highly competent scientists and engineers, but they went over budget and fell well behind schedule. One interesting analysis as to why these programs went over schedule and budget is from Rob Manning, the Chief Spacecraft Engineer, NASA Jet Propulsion Laboratory:

> The more money that's involved, the less risk people want to take. The less risk people want to take, the more they put into their designs, to make sure their subsystem is super-reliable. The more things they put in, the more expensive the project gets. The more expensive it gets, the more instruments the scientists want to add, because the cost is getting so high that they're afraid there won't be another opportunity later on- they figure this is the last train out of town. So little by little, the spacecraft becomes gilded. And you have these bad dreams about a spacecraft so bulky and so heavy it won't get off the ground- never mind the overblown cost. [...] That boils down to the higher the cost, the more you want to protect your investment, so the more money you put into lowering your risk. It becomes a vicious cycle [1].

While Manning is referring to scientific robotic missions, it is important to mark the distinction between human and robotic spaceflight. Human spaceflight is more risk-averse by nature of having to protect the livelihood of a crew. However, some points of analysis from Manning's reflection may hold merit for human spaceflight programs. The results of the missions leading up to the 2000s are successive failures, as shown in the table below:

Table 2. NASA Mission results leading up to the 2000 [1]

Program	**Year**	**Issues**	**Mission Results**
Voyager	1977	On time, on schedule	Success
Galileo	1986	3x cost overrun, 7 years behind	Antenna Failure
Mars Observer	1992	2x cost overrun, 4 years behind	Lost in Space
Hubble Space Telescope	1993	6x cost overrun, 6 years behind	Flawed Mirror
Cassini/CRAF	1997	2x cost overrun, 3 years behind	Hugely De-scoped

The consecutive results of these over-budget and delayed programs impacted NASA's reputation, government funding, and rippled through the organization. In 1993, after widely publicized issues with the Hubble Space Telescope, NASA, the United States Congress, and President agree that NASA needed to move away from multi-billion dollar science missions and impose strict budgetary amounts and rules. The resulting budgets were to be small and inflict harsh program cancelation policies if they went over budget.

In 1995, the Jet Propulsion Laboratory, came up with Mars Pathfinder for $0.15 Billion. It was on time (34 months), on budget, and 100% successful. The Pathfinder program quickly built a small, focused spacecraft, taking risks only small budgets allow, and bounced the lander onto the surface using an innovative combination of new technology, proven engineering, and hard-won experience. The result was for 1/20th of the price and for 1/3rd of the development time of the Viking program. A team of around 100 employees were able to plan, design, build, and execute the successful landing of a rover on Mars with compelling scientific results. The success defined the era of *Better, Faster, and Cheaper* for robotic space programs [1].

The method that captures the success of Pathfinder can be characterized by the Lean Innovation Cycle, as shown in Figure 1:

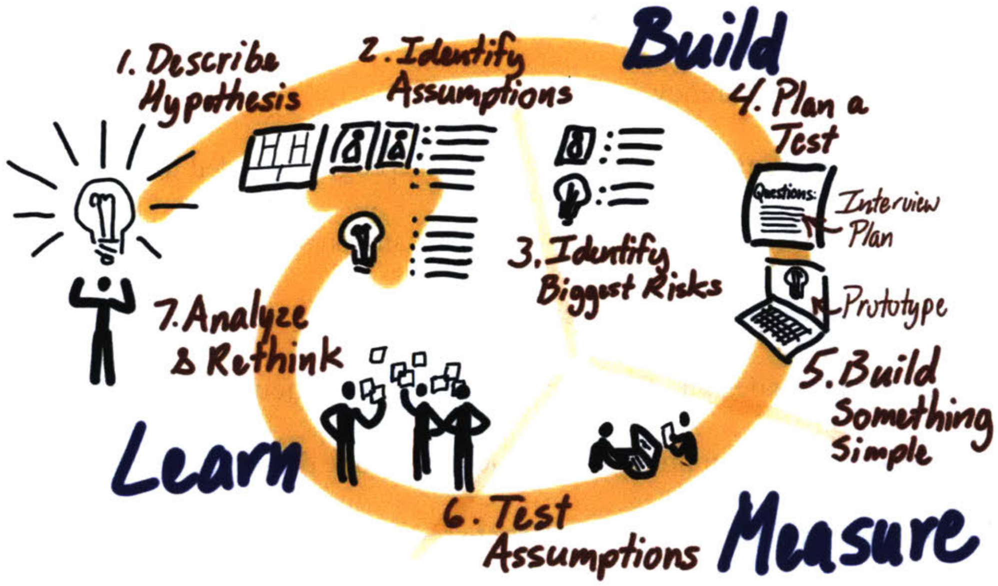


Figure 1. The Lean Innovation Cycle [1]

Using the framework of *build, measure, learn*, and the corresponding seven steps from the Lean Innovation Cycle, Pathfinder was able to break from the traditional program mold and achieve success. The steps as applied to Pathfinder were:

1. Describe the Hypothesis: Landing on Mars and roving (as opposed to orbiting) provided a compelling technical demonstration for scientific exploration on another planet, paving the way for further exploration.

2. Identify Assumptions: Using innovation and targeted risk acceptance enabled the program to keep a very low budget compared to other programs.

3. Identify Biggest Risks: The highest cost and largest risk is entry, descent, and landing of the rover. Maintaining low cost options, required new innovations. It was feasible to conduct a direct entry from deep space and use airbags to "just get it down," as a trade-off between cost, risk, and schedule.

4. Plan a Test: The program tested things extensively instead of adding redundancy. They emphasized reuse of spare parts where possible. The program also engaged the public early and often, with full transparency.

5. Build Something Simple: To keep on track, the program needed to build the smallest, simplest system possible. The need to remain on budget and on schedule was highly emphasized. The key was to prevent scope creep, by not allowing afterthoughts or nice-to-have features to be included in the design. Cheaper science missions should open the possibility of many more inexpensive follow-on missions in which to include those additional features.

6. Test Assumptions: The program's ultimate test was the first lending attempt on Mars using the new system.

7. Analyze and Rethink: The science community, the public, and NASA were excited that they had proven way to land and conduct science on Mars for relatively low cost. This led to the approval to launch more rovers, landers, and orbiters. This proposal involves repeating this cycle, each time building on past success, retiring risks, and raising excitement.

The strength of the Lean Innovation Cycle stems from the advantages of a prioritized task list that avoids making a big list of features and prevents scope-creep. The iterative nature allows a natural way to test if the program's assumptions were correct along the way. Other similar names for this cycle are the virtuous feedback cycle and agile development. Space Exploration Technologies Corporation (SpaceX) is a notorious proponent of this method, using a fail-fast technique to keep cost down and schedule on track. Their application of the lean innovation cycle steps may take the form of the following:

1. Describe Hypothesis: The major barrier to commercial space access is the widespread disbelief that a single private company can develop and launch a rocket

2. Identify Assumptions: A successful launch of a small rocket will lead to money for increasingly larger rockets.

3. Identify Biggest Risks: The largest risk is system integration which is mitigated by using the least costly test as possible, an inchoate small rocket.

4. Plan a Test: Build a company to design, prove, and integrate the completely new engine and rocket

5. Build Something Simple: SpaceX develops and builds the first Falcon 1 rocket.

6. Test Assumptions: The first three launches fail. The cycle iterates and the fourth is a success. This leads to an investment surge and burgeoning development of the larger Falcon 9 rocket.

7. Analyze and Rethink: SpaceX's strategy is to maintain the current pace with larger rockets, cycling to learn, with the eventuality of getting to Mars.

NASA's unmanned past and reoccurring entanglement with the Lean Innovation Cycle is now being used in a similar capacity by companies like SpaceX to develop manned spacecraft and rockets. As shown, the lean innovation cycle worked well for unmanned space missions in NASA's past. Perhaps in NASA's future manned programs such cycles may be used to help prevent scope creep and minimize the complicated requirements developments that introduce sources of hazard into the rocket's design. Leading into current development of SLS was a story of triumph and tragedy with the Space Shuttle and the canceled Constellation Program of the early 2000s.

2.1.2 Shuttle Retirement

The Space Transportation System, commonly referred to as the Space Shuttle program, was operational from 1982 to 2011, flying 135 missions. It was a partially reusable spacecraft system capable of accessing low earth orbit and landing to a runway for retrofit. The shuttle system included the Orbiter Vehicle (OV) with three Rocketdyne RS-25 main engines, two recoverable four-segment Solid Rocket Boosters (SRBs), and an expendable External Tank (ET) for liquid oxygen (LOX) and liquid hydrogen (LH2) fuel for the main engines.

Of the total missions, two were lost in mission accidents, including lives of all seven astronauts on each orbiter. The first was the *Challenger* SRB O-ring disaster in 1986, where a ruptured O-ring caused vehicle breakup and disintegration on launch. The second was the *Columbia* Thermal Protection System (TPS) failure in 2003, where ET foam shedding on launch struck the leading edge of the OV's wing, degrading the viability of the TPS, and causing breakup and disintegration on re-entry.

A NASA risk assessment study concluded that the agency had underestimated the level of risk involved in operating the Shuttle. There was a 1 in 10 chance of a catastrophic disaster during the first 25 flights but safety improvements had later improved the risk ratio to 1 in 90 as shown in Figure 2. The corresponding underestimations on part of the original Probabilistic Risk Assessment (PRA) studies conducted are indicated by a red "x" on Figure 2.

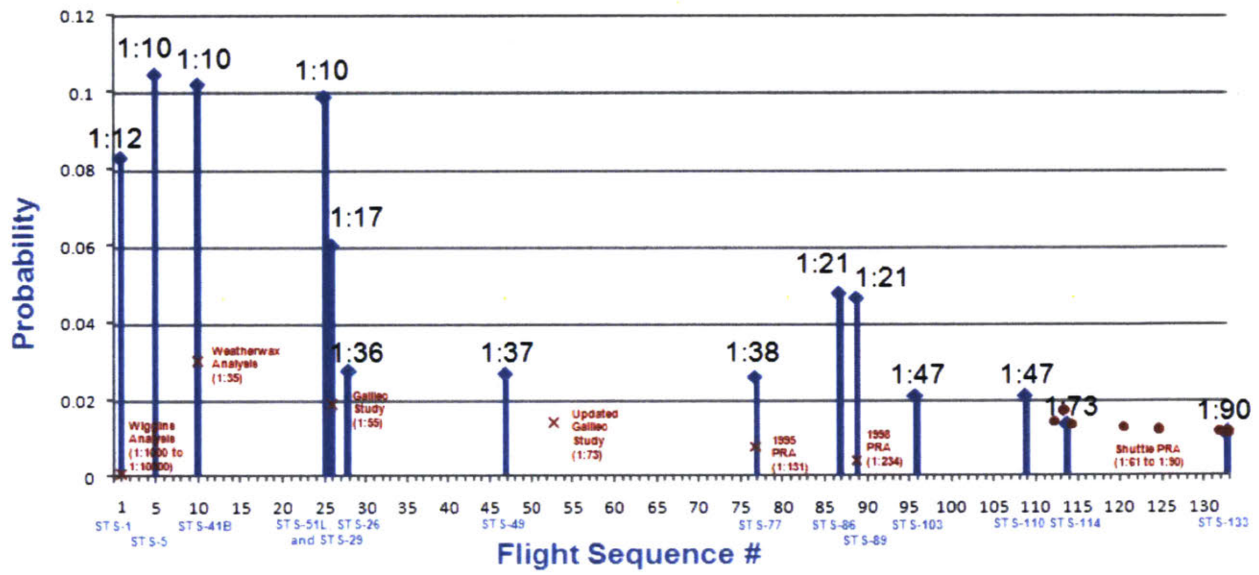


Figure 2. Space Shuttle Probability of Catastrophic Loss of Vehicle and Crew [2]

A NASA report on Shuttle Risk Progression states [2]:

> The results indicated that the Shuttle risk tends to follow a step function as opposed to following a traditional reliability growth pattern where risk exponentially improves with each flight. In addition, it shows that risk can increase due to trading safety margin for increased performance or due to external events. Due to the risk drivers not being addressed, the risk did not improve appreciably during the first 25 flights. It was only after significant events occurred such as Challenger and Columbia, where the risk drivers were apparent, that risk was significantly improved. One lesson learned from the [Space Shuttle Program] is understanding risk drivers are essential in order to considerably reduce risk. This will enable the

new program to focus time and resources on identifying and reducing the significant risks.

For several reasons, safety included, congress cancelled the program with its final mission taking place in 2011. The Space Transportation System was justified with the intent to provide frequent low-cost, low-risk, high-capacity access to low earth orbit. It ended with reusability as the fatal flaw in its design, however. With the knowledge of risks and hazards associated with STS, NASA moved to replace it with a more traditional rocket/booster launch vehicle within Project Constellation. While supporting space shuttle flights for roughly $1 Billion per launch, NASA was struggling to make the budget work to begin developing a new vehicle without canceling the shuttle program first.

2.1.3 Project Constellation

Prior to the cancelation of the Space Shuttle, the United States planned to build a new manned spaceflight program called Constellation. The goal was to be able to regain astronaut experience beyond low earth orbit with completion of the International Space Station, return to the moon, and then on to Mars. In 2009, the Augustine Committee found that with an already staggering budget estimated to cost $230 Billion (2004) however, the project was deemed infeasible without a substantial increase in funding. The project was effectively cancelled between 2010 and 2011 along with the retirement of the space shuttle, with no replacement for the United States capability to execute manned spaceflight launches domestically.

2.2 NASA and the Space Launch System

With budget constraints and a political wind shift, NASA and the U.S. Congress were forced to significantly reduce the cost of their manned spaceflight launch program if they wanted to pass a new authorization act. The Space Launch System is the first exploration-class rocket since the Apollo-era Saturn V rocket. Designed to take humans beyond low earth orbit, SLS is designed for deep space missions that will send Orion or other cargo to the Moon and beyond. By comparison with the International Space Station (ISS), the launch vehicle needs to reach a speed of 24,500 mph to achieve successful trans-lunar injection, 7,000 mph faster than ISS and nearly 1,000 times farther. Future upgrades will provide more power for human exploration to Mars and robotic missions to Saturn and Jupiter. To achieve the power required, the Core Stage has two 5-segment Solid Rocket Boosters and four RS-25 Rocket Engines that provide the necessary thrust [3].

2.2.1 New authorization for Space launch system

To reduce the cost, NASA announced its selection for a new launch system design in September 2011, similar in capability to Constellation, but for much less cost, on the order of estimated $7 Billion (2014). The Space Launch System (SLS) planned for re-use of space shuttle components to help offset the cost. Three block upgrades were initially planned for the SLS: Block 1 (95 metric ton payload capacity), Block 1B (105 metric ton payload capacity), and Block 2 (130 metric ton payload capacity). Each block will use the same core stage rocket produced by Boeing, with four main RS-25 engines and shuttle-derived solid rocket boosters, but Block 1B will feature the Exploration Upper Stage (EUS), a more powerful second stage than Block 1, and Block 2 will include upgraded advanced solid rocket boosters.

The NASA Authorization Act of 2010 included the first provisions for the new Space Launch System. SLS is being built by more than 1,000 companies from across the United States. Every NASA center plays a role in some form in supporting its development. The SLS Program at NASA's Marshall Space Flight Center works closely with the Orion Program, managed by

NASA's Johnson Space Center in Houston, and the Exploration Ground Systems at Kennedy Space Center (KSC). All three programs are managed by the Exploration Systems Development Division within the Human Exploration and Operations Mission Directorate at NASA Headquarters in Washington D.C. The Core Stage and Exploration Upper Stage are being designed in Huntsville, Alabama and built at NASA's Michoud Assembly Facility in New Orleans. The program introduces a host of new technological improvements, such as new manufacturing techniques that include a friction-stir welding tool that is the largest of its kind in the world [3].

2.2.2 SLS Development

Aerojet Rocketdyne of Sacramento, California, is upgrading an inventory of 16 RS-25 shuttle engines to SLS performance requirements, including a new engine controller, nozzle insulation and required operation at 512,000 pounds of thrust. Two shuttle-derived solid rocket boosters are being used for the initial flights of the SLS. To provide the additional power needed for the rocket, the prime contractor for the boosters, Northrop Grumman, of Redondo Beach, California, has modified the original shuttle's configuration of four propellant segments to a five-segment version. The design includes new avionics, propellant design and case insulation, and eliminates the recovery parachutes [3].

In the recent years, the SLS program has progressed similarly to other rationally large NASA missions, with delays, over-runs, and occasional issues. Being crew-rated however, the threshold for risk is higher than that of United States commercial rocket sector and unmanned robotic science missions. SLS is different from commercial rockets because of its scale and complexity. Commercial rockets are not yet crew-rated and do not require the more stringent risk mitigation efforts in their design and testing that SLS does. Although commercial crew programs are being developed for low earth orbit ferry missions, the mission variables are far less complex than the exploration class rocket that is SLS. Despite the challenges and high threshold, the program is continuing through several milestones. In 2013, the program completed the Preliminary Design Review (PDR) which encompasses all aspects of the system design. In 2014, Block 1 entered into full-scale development. In 2017, however, NASA announced that the schedule for the maiden flight would be delayed to 2019.

2.2.3 The Current NASA Design Process

Understanding the current NASA design process requires a foundation in modern systems engineering. Systems engineering started to become prevalent in the 1950s when systems were becoming significantly more complex, especially in the missile industry. Program managers struggled with an informal design and manufacturing process that often resulted in systems that did not completely satisfy the objectives and intent of their program. This lack of a structured process also led to projects being late and over budget. System engineering is the attempt to put structure into the design and manufacturing of a system in order to improve the results of the engineering effort [4]. The process and phases of system engineering are commonly illustrated as the most basic "V-Model" as shown in Figure 3.

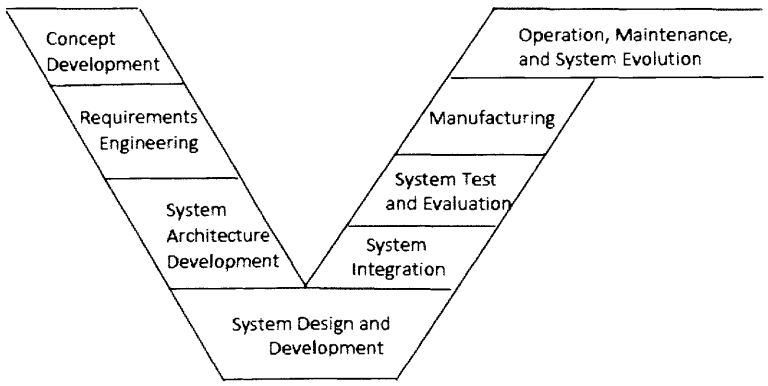

Concept Development
Requirements Engineering
System Architecture Development
System Design and Development
System Integration
System Test and Evaluation
Manufacturing
Operation, Maintenance, and System Evolution

Figure 3. Basic Systems Engineering "V-Model" [4]

The “V-Model’s” basic premise is that the best results ensue if a structured step-wise process is used from beginning to end. The left side of the “V” represents the basic design process. The right side of the “V” shows the later stages in development when assurance and manufacturing take place. The process begins with developing the system concepts first and then defining the basic goals and constraints on the system. From there, various feasibility and other analyses may be conducted to refine the program’s design and form, thereby saving later resources from low-viability design options. The next step involves requirements engineering where detailed requirements are developed for the system based on the earlier definitions of basic goals and constraints in the system concept phase. The last step in the left side of the “V” is the system architecture development where a high-level design of the system is created before detailed design and development. In right side of the “V-Model” process, the various components are integrated and go through a testing and evaluation phase. Lastly, the model is completed with final manufacturing and usage of the product [4]. When iterated, the process becomes a product life-cycle which is important to understanding for future applications of systems-based hazard analysis approaches.

Further applicability to the software intensive systems is an understanding of requirements engineering. The “V-Model” only has requirements listed on the left side of the “V,” however, requirements are rarely all-inclusive from the program’s outset. As the design matures, the refinement process naturally generates more understanding of the problem being solved which results in additions, deletions, or modifications of previously existing requirements and constraints. In some instances, serious problems during development may occasionally require the project to completely revert to an earlier phase [4].

The current systems engineering process is adapted from this model for NASA programs, as illustrated in Figure 4 below.

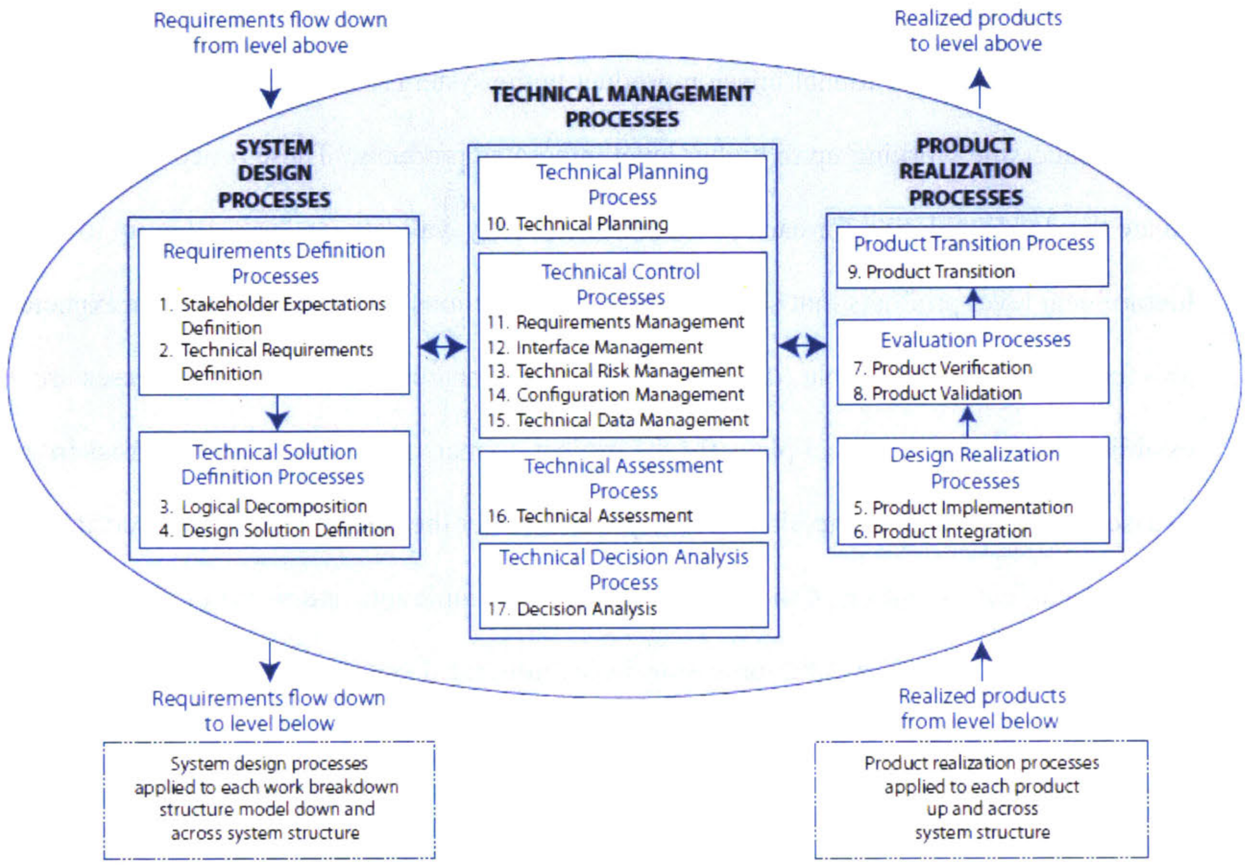

Requirements flow down from level above
Realized products to level above
TECHNICAL MANAGEMENT PROCESSES
SYSTEM DESIGN PROCESSES
PRODUCT REALIZATION PROCESSES
Technical Planning Process
10. Technical Planning
Requirements Definition Processes
1. Stakeholder Expectations Definition
2. Technical Requirements Definition
Technical Control Processes
11. Requirements Management
12. Interface Management
13. Technical Risk Management
14. Configuration Management
15. Technical Data Management
Product Transition Process
9. Product Transition
Evaluation Processes
7. Product Verification
8. Product Validation
Technical Solution Definition Processes
3. Logical Decomposition
4. Design Solution Definition
Technical Assessment Process
16. Technical Assessment
Design Realization Processes
5. Product Implementation
6. Product Integration
Technical Decision Analysis Process
17. Decision Analysis
Requirements flow down to level below
Realized products from level below
System design processes applied to each work breakdown structure model down and across system structure
Product realization processes applied to each product up and across system structure

Figure 4: The NASA Systems Engineering Design Process [5]

NASA categorizes this as a systems engineering engine. It includes three major processes:

1. System Design Process
2. Technical Management Process
3. Product Realization Process

The steps within the System Design Process relate to the left side of the "V-Model." Their intent is to define and baseline stakeholder expectations, generate and baseline technical requirements, and convert the technical requirements into a design solution that will satisfy the base-lined stakeholder expectations. Designers not only develop the design solutions to the products intended to perform the operational functions of the system, but also establish requirements for the products and services that enable each operational/mission product in the system structure. The steps within the product realization process represent the right side of the "V-Model." They are applied to each operational/mission product in the system structure starting from the lowest level product and working up to higher level integrated products. These processes are used to create the design solution for each product and to verify, validate, and transition up to the next hierarchical level products that satisfy their design solutions and meet stakeholder expectations as a function of the applicable life-cycle phase. The technical management processes are used to establish and evolve technical plans for the project, to manage communication across interfaces, to assess progress against the plans and requirements for the system products or services, to control technical execution of the project through to completion, and to aid in the decision making process. This is a communication-based integrated system of processes used to manage an iterative "V-Model." The three major processes within the NASA systems engineering engine are used both iteratively and recursively:

Iterative - the application of a process to the same product or set of products to correct a discovered discrepancy or other variation from requirements.

Recursive - is adding value to the system by the repeated application of processes to design next lower layer system products or to realize next upper layer end products within the system structure. This also applies to repeating application of the same processes to the system structure in the next life-cycle phase to mature the system definition and satisfy phase success criteria.

When applied iteratively and recursively, a product lifecycle emerges. As the program's lifecycle moves forward through maturation, a series of milestones are put in place to manage the evolution. The program lifecycle and its phases are illustrated in Figure 5.

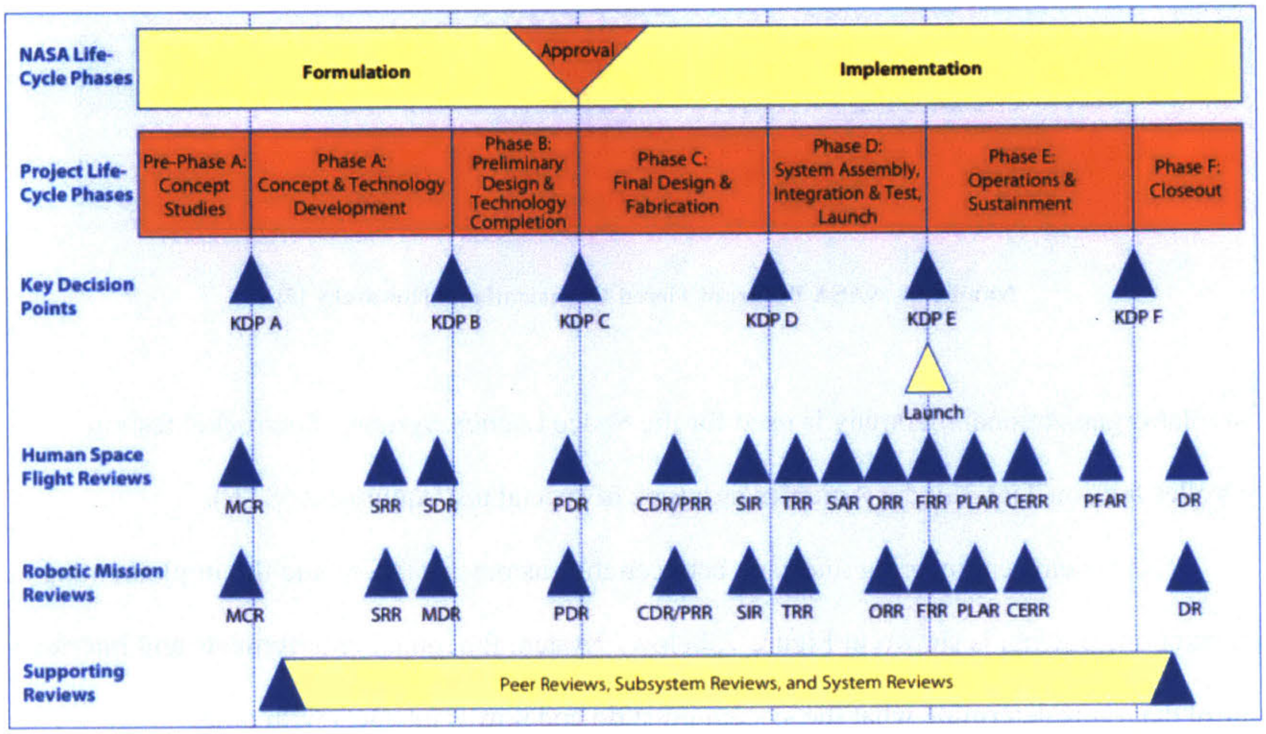
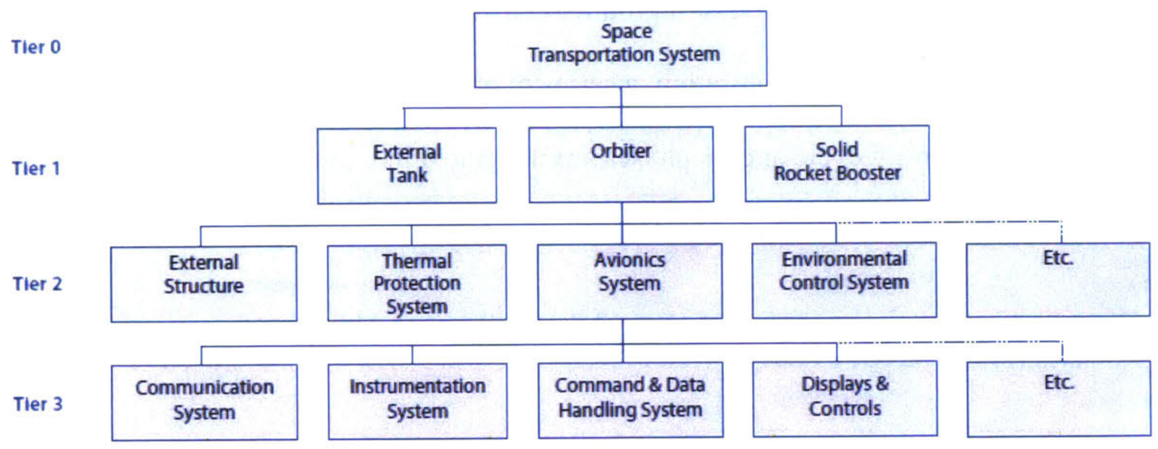

NASA Life-Cycle Phases
Formulation
Approval
Implementation
Project Life-Cycle Phases
Pre-Phase A: Concept Studies
Phase A: Concept & Technology Development
Phase B: Preliminary Design & Technology Completion
Phase C: Final Design & Fabrication
Phase D: System Assembly, Integration & Test, Launch
Phase E: Operations & Sustainment
Phase F: Closeout
Key Decision Points
KDP A
KDP B
KDP C
KDP D
KDP E
KDP F
Launch
Human Space Flight Reviews
MCR SRR SDR PDR CDR/PRR SIR TRR SAR ORR FRR PLAR CERR PFAR DR
Robotic Mission Reviews
MCR SRR MDR PDR CDR/PRR SIR TRR ORR FRR PLAR CERR DR
Supporting Reviews
Peer Reviews, Subsystem Reviews, and System Reviews

Figure 5. NASA Project Life-cycle takes a program from concept study through program closeout [5]

Several milestones are required for human space flight review as show by the blue triangles. At the time of this thesis, the SLS core stage is currently in Phase C: Final Design and Fabrication and past the Critical Design Review (CDR) milestone.

NASA organizes the program structure into a tiered hierarchy based on major systems, sub-systems, and sub-components as illustrated in Number 6 below for the Space Shuttle Program which fell under the Space Transportation System (STS).



Number 6. NASA Program Tiered Organizational Hierarchy [5]

A similar organizational hierarchy is used for the Space Launch System. The rocket testing controller falls on Tier 3 under Avionics as a type of special test equipment (STE).

The systems engineering interface between the customer, NASA, and the implementing organization, Boeing, is shown in Figure 7, below. System functional requirements and interface control diagrams determine what the system must do and how it interacts with other organizations' systems. The requirements are further expanded into each sub-tier with more particularization.

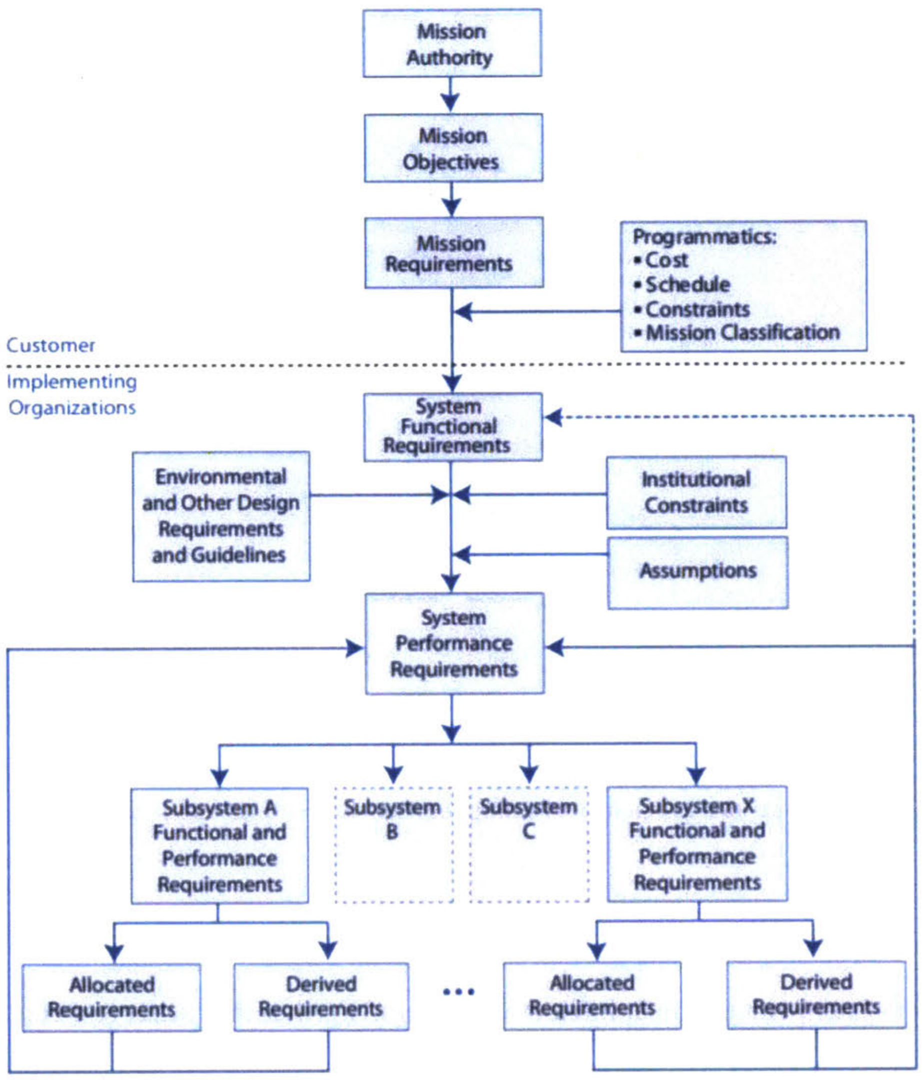

Mission Authority
Mission Objectives
Mission Requirements
Programmatics:
• Cost
• Schedule
• Constraints
• Mission Classification
Customer
Implementing Organizations
System Functional Requirements
Environmental and Other Design Requirements and Guidelines
Institutional Constraints
Assumptions
System Performance Requirements
Subsystem A Functional and Performance Requirements
Subsystem B
Subsystem C
Subsystem X Functional and Performance Requirements
Allocated Requirements
Derived Requirements
Allocated Requirements
Derived Requirements

Figure 7. The implementing organization designs their end product based on a requirements flow-down [5]

The major components fit together in a bottom-up realization process from the smallest components coming together with each other to form the end product in the next higher level. The lower tier end products are verified against their specified requirements and validated against stakeholder expectations and concept of the operation (ConOps) as shown in Figure 8, below.

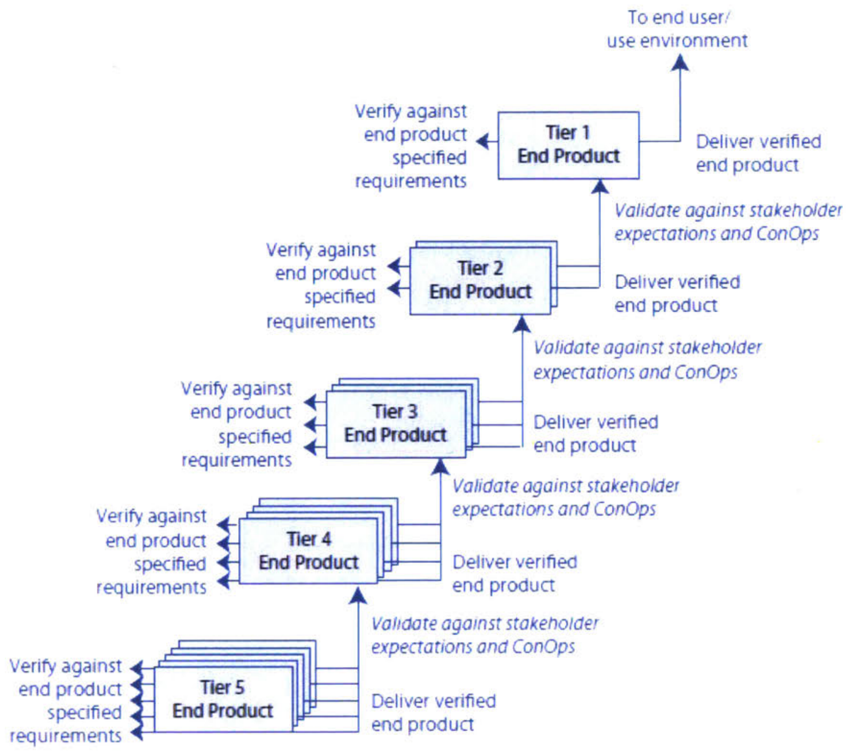

To end user/ use environment
Verify against end product specified requirements
Tier 1 End Product
Deliver verified end product
Validate against stakeholder expectations and ConOps
Verify against end product specified requirements
Tier 2 End Product
Deliver verified end product
Validate against stakeholder expectations and ConOps
Verify against end product specified requirements
Tier 3 End Product
Deliver verified end product
Validate against stakeholder expectations and ConOps
Verify against end product specified requirements
Tier 4 End Product
Deliver verified end product
Validate against stakeholder expectations and ConOps
Verify against end product specified requirements
Tier 5 End Product
Deliver verified end product

Figure 8. Sub-tier components coming together to create an end product [5]

The rocket testing controller is designed within this structure against requirements derived from the entire rocket system since it must interact with them. Further the rocket test controller is used in the final validation of Boeing's Tier 1 End Product, the SLS core stage integrated with four RS-25 rocket engines. The requirements change over the life-cycle of the program with design maturation. Internal Requirements Revision (IRR) and Engineering Change Process (ECPs) drive these requirements changes in the development cycle.

2.3 The Boeing Company

Boeing is the prime contractor with NASA for the design, development, test and production of the launch vehicle cryogenic stages, as well as development of the avionics suite which include the Core Stage (CS) and Exploration Upper Stage (EUS). Boeing Defense, Space, and Security out of Huntsville, Alabama, is designing the SLS core stage, including the avionics that will control the vehicle during flight. The SLS avionics computer software is being developed by NASA at Marshall Space Flight Center in Huntsville. The core stage is being built by Boeing at NASA's Michoud Assembly Facility in New Orleans. All major structures are built and are being outfitted for Exploration Mission-1, and Boeing has started building structures for Exploration Mission-2. Part of the contract for the core stage includes ground support equipment and structural testing. Structural test articles are built and tested to ensure loading, stresses, and environmental effects are reasonable. Boeing's progress is given in the timeline below [6]:

Table 3. Boeing Progress on SLS.

Date	News Release
June 21, 2012	Boeing Successfully Completed Key Reviews of Space Launch System
Dec 21, 2012	Boeing Completed Preliminary Design Review for Space Launch System Core Stage
July 2, 2014	NASA and Boeing Signed Space Launch System Contract
April 23, 2015	Boeing Named John Shannon Space Launch System Vice President
June 8, 2016	Boeing Awarded $200 million in Space Contracts to Small Businesses
April 3, 2017	Boeing Unveiled Deep Space Concepts for Moon and Mars Exploration

Core Stage 1 poses a unique challenge because it is a first-time build that is also a flight article. With the first round of building, Boeing is experimenting while also learning how to build the first flight vehicle, maintaining configuration and quality controls sufficient to enable the EM-1 test flight with high confidence. With the first iteration, a responsibility to develop a system to build several more rockets is being designed and improved concurrently alongside the first rocket itself. Part of maintaining high confidence is the testing phase. A captive hot-fire test will be used to provide Boeing and NASA with the confidence to go ahead with EM-1. To test the rocket, it needs some form of special test equipment to control it.

2.4 Rocket Test and Control

As of 2012, the John C. Stennis Space Center (SSC), located on the banks of the Pearl River at the Mississippi–Louisiana border, is NASA's largest rocket engine test facility. Presently, over 30 local, state, national, international, private, and public companies and agencies use SSC for their rocket testing facilities. The B1/B2 test stand was originally built in the 1960s to simultaneously test the five F1 engines of the complete Saturn V first stage from 1967 to 1970. During the shuttle era it was modified to test the RS-25 Space Shuttle Main Engine.

The stand is a dual-position, vertical, static-firing stand supporting a maximum dynamic load of 11 million pounds-force (lbf). Stennis now leases the B1 test position to Pratt & Whitney Rocketdyne for testing of RS-68 engines for the Delta IV launch vehicle. NASA is currently preparing the B2 test position to test the SLS core stage during the upcoming captive hot-fire testing. It has been an involved process that mirrors work done during the Main Propulsion Test Article (MPTA) Project at Stennis in the 1970s, when a space shuttle external tank, simulated orbiter and three main engines were installed on the stand and test fired just as during an actual launch. Now, the stand will provide an upgraded rendition of the shuttle main propulsion test with its test of the SLS core stage with one very large difference. The SLS being tested is not a mockup; the stage to be tested will actually fly. Because this will be an actual flight article, the plan calls to exercise extreme care. The SLS core stage, with four RS-25D rocket engines, will be installed on the stand for propellant fill and drain testing and two hot-fire tests.

The work to prepare the stand for the stage testing was divided into three phases: restoration of the stand to its original design condition, buildout of the stand to accommodate the larger SLS core stage and completion of the special test equipment interfaces (structural, mechanical, and electrical) needed for testing, such as extending the derrick crane atop the stand by 50 feet, repositioning the 1.2-million-pound original shuttle MPTA framework structure, adding another 1 million pounds of steel to extend the structure to accommodate the larger SLS stage and upgrading the massive high-pressure industrial water system to provide as much as 335,000 gallons of water per minute to the stand during test operations [7].

Once installed, the core stage will undergo chill-down and hot-fire tests. To control of the complex system, special test equipment is being developed by Boeing. As shown in Figure 9, modern complexity of the SLS core stage requires significant resource allocation toward software development.

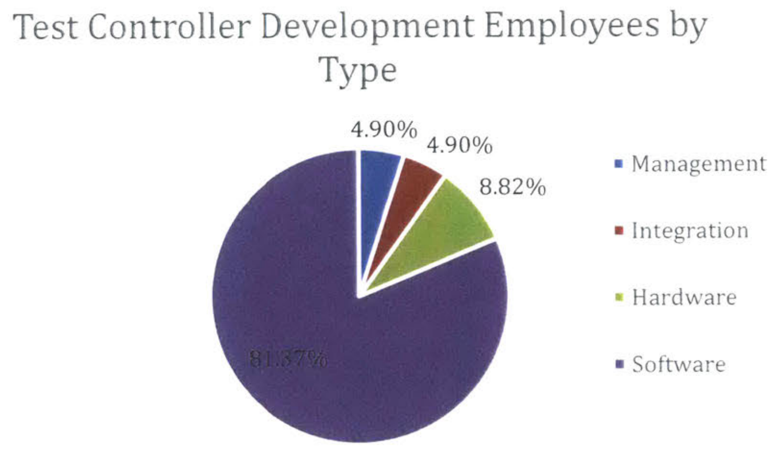


Figure 9. Test Controller Employees are made up of >81% Software Engineers

The proportionality of management, integration, and hardware-type employees to the number of software-type employees indicates the potential for increased focus needed for software-related safety considerations of the special test equipment for the SLS core stage rocket testing controller design.

Chapter 3

3 Literature Review

This chapter will provide a review of the academic literature surrounding safety engineering in general, and specifically, in the field of hazard analysis. The first section covers the current hazard analysis standards used within NASA. Section 3.2 covers traditional accident models and a look at their issues and merits. Section 3.3 provides the basis and philosophy behind systems thinking applied to hazard analysis and introduces Systems-Theoretic Accident Modeling and Processes (STAMP) and System-Theoretic Process Analysis (STPA). Section 3.4 covers prior case studies and discussion of these types of hazard analyses.

3.1 Sources of Hazards

To begin to understand safety of the rocket testing controller, an examination of sources of hazards is needed. Hazards can arise from several different areas and different reasons within the design of a product. The following definitions are required to understand hazards before determining their sources:

System - a set of components that act together as a whole to achieve some common goal, objective, or end. A system may contain subsystems and may also be part of a larger system [4].

Loss - involves something of value to stakeholders. Losses may include a loss of human life or human injury, property damage, environmental pollution, loss of the mission, loss of reputation, loss or leak of sensitive information, or any other loss that is unacceptable to the stakeholders.

Hazard - a system state or set of conditions, together with a particular set of worst-case environmental conditions, that will lead to a loss.

The rocket testing controller has the following qualities:

- Exists at a low tier in the product structure
- Is driven by requirements from multiple stakeholders
- Is software intensive
- Is involved with a highly dangerous test

These qualities make the rocket testing controller susceptible to a few specific sources of hazards that are described in the following sections. The following sections will outline potential sources of hazards, how these hazards are induced in a system such as a rocket testing controller, and some review and discussion from literature on ways to analyze these sources and mitigate them.

3.1.1 Effect of Concurrent Engineering on Requirements for n-Order Subsystems

The method being used for development of the Space Launch System is called concurrent engineering, as opposed to traditional serial or sequential engineering. In serial engineering, products are designed in sequence or in a linear fashion. Concurrent engineering is a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. Concurrent engineering is also known as simultaneous engineering. During development, different stages run simultaneously, rather than consecutively. The intent is to decrease product development time and also the time to market or delivery, leading to improved productivity and reduced costs. This approach is intended to cause the developers from the outset, to consider all elements of the product life cycle from conception to disposal, including quality, cost, schedule, and user requirements [8].

For a subsystem at a lower tier, with multiple stakeholders such as the rocket test controller this may be a source of fluid requirements as the design matures through the product design. Viewing the tiers as an n-order system, lower tiers experience a higher order of effects due to requirements propagation. Figure 10 demonstrates this example of propagation of requirements through a 4-order tiered system:

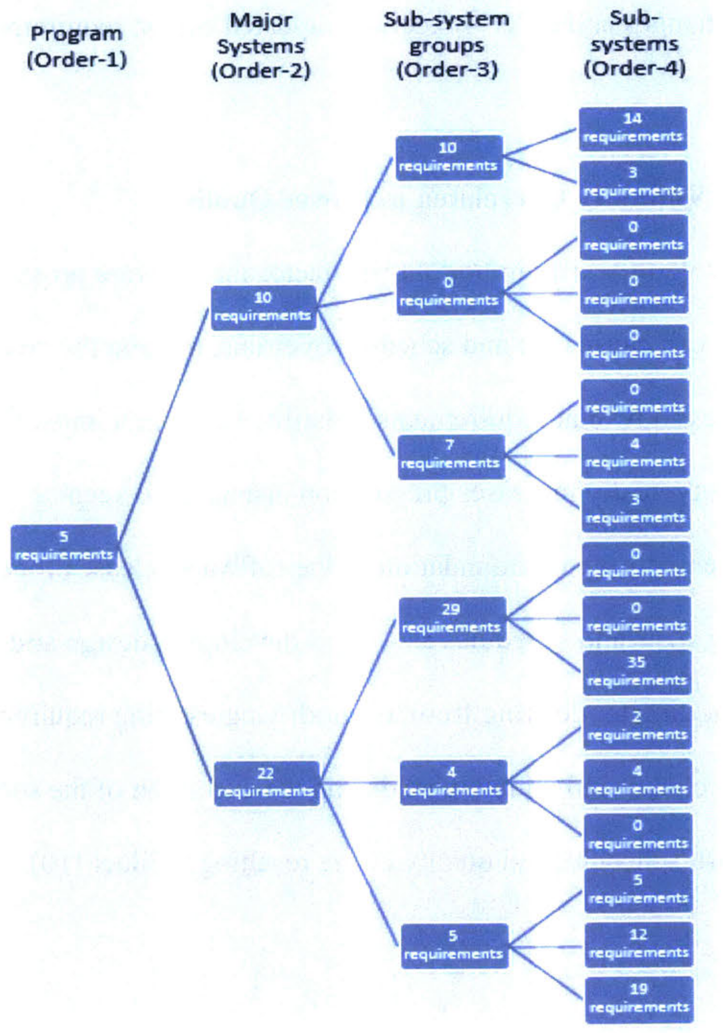


Figure 10. Requirements Propagation through an n-Order System

In this example, five (5) requirements modifications, additions, or deletions at the program level has anywhere from zero to 35 requirements changes at the sub-system level by the time it propagates throughout the lower levels, depending on how the sub-system interacts with the overall change as defined through ConOps and interface control documents.

The propagation is not instantaneous but takes time to filter downstream to lower tiers. A sub-system group in order-3 may require numerous work hours to determine what changes must be made to an order-4 sub-system. This propagation effect leads to a time-lag between order-1 issuing requirements changes and order-4 receiving updated design requirements as it is at the tail end of the change.

3.1.2 Requirements Volatility Correlated to Lower Quality

Requirements volatility is an important risk factor for software projects. High requirements volatility can cause cost and schedule overruns, making the goals of the project hard to achieve. Studies show that requirements volatility has a high impact on project performance [9]. This typically increases pressure on managers to keep up and make up lost schedule time. Requirements are the foundation of the software release process. They provide the basis for estimating costs and schedules as well as developing design and testing specifications. Thus, adding to, deleting from, or modifying existing requirements that have been agreed to by both clients and maintainers during the execution of the software maintenance process impacts the cost, schedule, and quality of the resulting product [10].

3.1.3 Requirements, Quality, and the Link to Safety

Complex systems will tend to have higher volatility due to the number of interacting components. Another cause of volatility is the weak domain knowledge by the development team which performs an incomplete analysis of the requirements [11]. This may be higher in a concurrent engineering environment such as SLS. Estimating requirements volatility is important in order to predict the risk that the project is going off track. High requirements volatility has an impact on the quality of the software product (i.e., increasing defect density) and the project performance (i.e., schedule delays, cost overruns, and amount of rework). Requirements volatility is considered to be a major source of risk to the management of large and complex software projects [11]. The following definitions help define metrics to assess these risks:

> **Defect density** – a quality metric, defined as number of confirmed defects detected in software during a defined period of development divided by the size of the software. It is measured in defects per number of lines of code, often in thousands of lines of code (KLOC) or million lines of code (MLOC).
>
> **Safety-critical defects** – a defect to a safety-critical system or life-critical system whose failure or malfunction may result in one (or more) of the following outcomes: death or serious injury to people, or loss or severe damage to equipment or property.

Intuitively, a higher defect density causes schedule delays by causing rework to fix the defects and further increases the probability of a safety critical defect not being observed and passing onto the final release. Studies provide a positive correlation between defect density and post-release safety critical defects. In one case for software development for the U.S. Department of Defense, as many as 20 post-release Safety Critical Defects were found in a program with 2.8

MLOC and defect density of 46.1 defects/MLOC [12]. Defects occur for similar reasons to safety oversights. Thus, volatility can result in unwanted induced hazards by changes not being fully corrected throughout the existing system design.

3.1.4 Measuring Requirements Volatility

In order to analyze volatility, the following definitions are required for understanding of the measurement.

Requirement Change – an added, deleted, or modified requirement from the original source document

Requirement change types are:

Added – a new requirement placed into the revision after the initial plan of record is agreed upon

Deleted – a previously agreed on requirement that is removed from the new revision after the initial plan of record is agreed upon

Modified – a requirement in which the original intent was determined to be incorrect and thus altered after previous agreed upon revision [10].

Each of these changes require additional work for the end-product developers that they apply to. The amount of work fluctuates with the volatility in these changes. To analyze the volatility a measurement is needed. One widely-accepted measure of volatility is from [10] is:

$$Requirements\ Volatility = \frac{Added+Deleted+Modified}{Total\ Requirements\ in\ Revision}\ x\ 100 \quad (1)$$

Requirement volatility represents the total change traffic applied to an agreed-to-release plan. It can be greater than 100% if changes occurred more than the total number of original requirements planned for the release [10]. One consideration is to use standard deviation or the total requirements by revision, σ. However, standard deviation of total requirements by revision, as a measure of volatility does not effectively account for the time factor of requirements changes since revisions are not always released on an equal time scale. The author of [9] suggests another possibility to measure volatility is to use the sizes of revisions which can also semi-automate the metrics collection. The measures of size of requirements documents are good indicators of the number of changes for (use case-based) requirements documents. The results show that the size measure "number of lines" (of a requirements document) is a good predictor of volatility [9]. The author of [9] proposes requirements volatility to be defined as the amount of changes to a requirements document over time and measure it as the sum of the change densities of a requirements document.

$$Volatility = \sum_{i=1}^{N_{revision}} \frac{N_{change}}{N_{words}} \quad (2)$$

Equation (2) provides an alternative operational definition of requirements volatility. It is a function of number of changes (N_{change}), time measured in number of revisions ($N_{revision}$), and size of the requirements document measured in N_{words} [9]. Depending on what data is available, both equations (1) and (2) provide applicable functions to analyze requirements volatility.

3.1.5 Effect of Requirements Volatility on Schedule (Service Length)

Requirements volatility effects the service length of development which can impact program schedule and cost. With increasing program schedule comes increased pressure on tail-end development sub-systems such as the rocket test controller to push for completion toward the originally scheduled completion date. If the completion date does not move, a number of factors may be managed in order to meet the completion date including: working longer hours or overtime, hiring more employees, value stream mapping and program risk reduction measures, and reducing scope of the project through re-prioritization of efforts. Examining each of these in more detail one can see the benefits and risks to each one with regard to the safety concerns.

- Working longer hours or overtime: Several adverse effects have been determined through research between extended hours or overtime and software development. Notable areas that are adversely effected are cognitive effects, mood and interpersonal effects, health effects and team performance. While the total number of faults associated with the implementation activities was lower than the design phase, one study noted that stress and human nature from such effects contributed to 73% of the implementation faults [13].
- Hiring more employees: The training time for new employees requires pulling resources away from ongoing projects for on-the-job training and onboarding. To get a new hire indoctrinated on the current complexity of the project takes significant time and resources. This approach is often too late in the program life-cycle to be an effective solution.
- Value stream mapping and program risk reduction measures: Attempting to find waste and optimize work flow is one approach that can be effective in reducing the development time. It takes resources away from current development but can often help streamline efforts and improve overall quality. An issue with this approach is the required resources and time away from development that employees will need to interact with the value-stream mapping team. After changes are recognized, implementing them

takes time and resources to enact through effective change management that may detract from ongoing projects.

- Reducing scope of the project through re-prioritization of efforts: Managers may be able to effectively reduce the scope of the project by prioritization of changed and added requirements. This can help to focus the team on what critical functions or features must be completed and which of those are perhaps not as important. This leaves room for interpretation and may avoid missing the desired intent of the overall design by potential mis-prioritization, however it can be an effective strategy for optimizing the team's resourcefulness towards more value-added development.

3.1.6 Queuing Theory and Service Length

Queuing theory and simulation of requirements volatility may be used to demonstrate sources of schedule impact which is linked to a higher defect rate and safety concerns. If requirements flow is viewed from an engineering level as a process flow, then queuing theory as show in Figure 11, can be applied to characterize the flow:

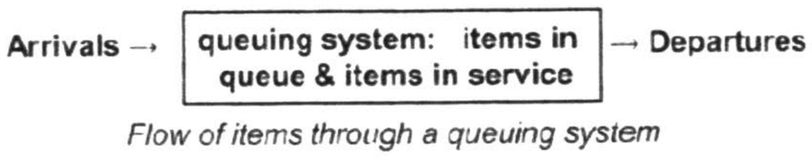

Arrivals
queuing system: items in queue & items in service
Departures
Flow of items through a queuing system

Figure 11. Schematic View of a Queuing System [14]

This flow of arrivals and departures are effected by work ongoing within the box, that represent the queue and work. This system is characterized by the process flow diagram in Figure 12 where the triangle represents the queue and the box represents the work being done on the units being processed.

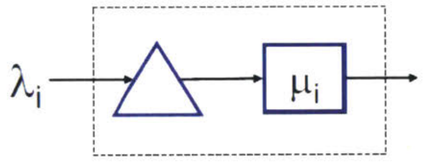


Figure 12. Process Flow Diagram used for Queuing Theory

The system is further characterized by the following variables and equation.

$$\rho_i = \frac{\lambda_i}{\mu_i} \tag{3}$$

where,

λ_i is the arrival rate measured in units of work per unit time

μ_i is the service rate measured in units of work per unit time

ρ_i is capacity utilization of the system

The effected buildup rate is thus:

$$buildup\ rate = \lambda_i - \mu_i \tag{4}$$

For this model we assume no storage capacity constraint, as requirements do not take up physical inventory space. If we characterize requirements as the unit of work, this model may be used to analyze the effects of requirements changes over time as arriving units of work and the developers as performing the required service to write code for those requirements.

Queuing theory, may be used to determine the length or volume inside the triangle portion of the process flow using the principle of the conservation of work flow. Conservation of work flow is defined by Little's law [14]:

$$L_s = \lambda W \tag{5}$$

where,

L_s = average number of items in the queuing system, also referred to as Work in Process (WIP)

W = average waiting time in the system for an item, also referred to as cycle time

λ = average number of items arriving per unit time, also referred to as throughput

Through the propagation of requirements due to concurrent engineering as show above, the model can be further expanded with multiple tiers to show the delay in design propagation and filter downstream of changes using an expanded tandem process flow diagram as shown in Figure 13.

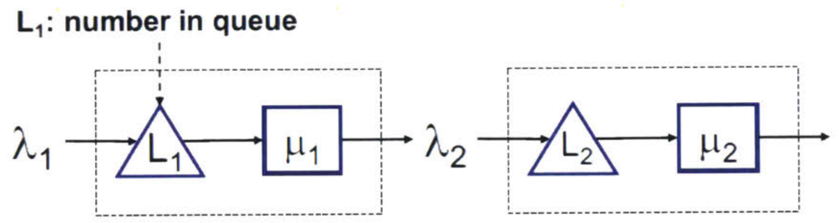


Figure 13. Tandem Process Flow Diagram

Thus, the arrival rate at process two is determined to be the minimum of the arrival rate of process one or service rate of process one, as show in equation (6).

$$\lambda_2 = \min(\lambda_2, \mu_1) \tag{6}$$

However, process 1 work is not characterized as full work in engineering the requirement, but could be merely a systems engineering analysis of downstream effects on sub-system designs. Therefore tandem queuing theory would be difficult to characterize using the above method since characterizing the upstream μ_i would be inconsistent.

As a result of requirements volatility, to further describe the system, G-G-N queuing theory may be applied. Since this volatility creates an unpredictable arrival rate it may be further characterized as the stochastic inter-arrival of requirements. The figure below illustrates the process flow as characterized by the G-G-N queuing theory.

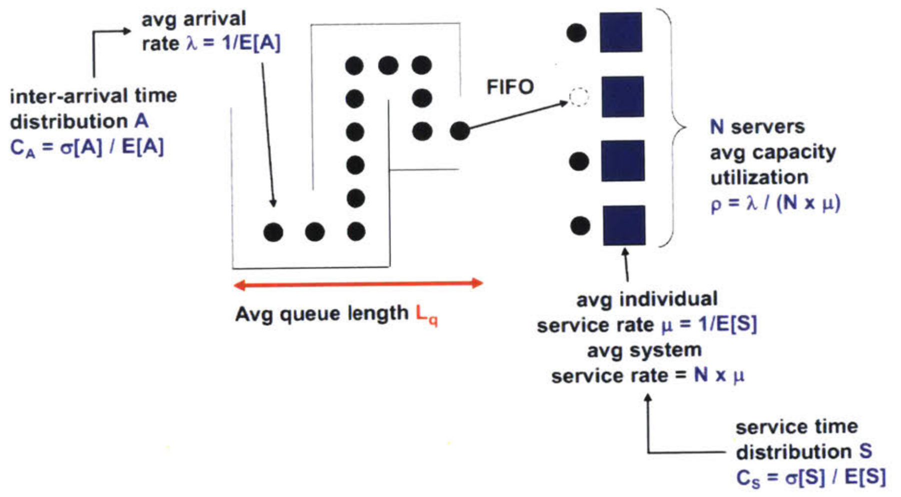


Figure 14. G-G-N Queuing Theory Diagram

This model is simplified by assuming and infinite queue size, first-in-first-out (FIFO), and $\rho < 1$. An acceptable approximation is therefore [14]:

$$L_q = \frac{\rho^{\sqrt{2(N+1)}}}{1-\rho} \, x \, \frac{C_A^2 + C_S^2}{2} \tag{7}$$

where,

L_q is the average queued number waiting to be worked on (requirements needing fulfillment)

N is the number of servers (developers with a capacity to perform work on requirements)

C_A is the coefficient of variation for inter-arrival times

C_S is the coefficient of variation for service times

The number at the server location, L_s as provided by Little's Law, and the number in the queue, L_q can therefore be used to determine the overall number in the system, L, as shown in the diagram and equation below.

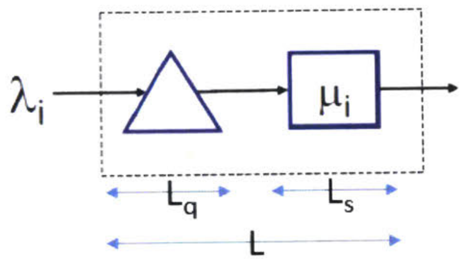


Figure 15. Sum of Service and Queue Length to Determine Work-in-Progress Length

$$L = L_q + L_s \tag{8}$$

The arrival distribution as characterized by the stochastic arrival of requirements changes that drives volatility is defined as distribution, A. Determining the standard deviation of distribution A, σ[A] and the expected value of distribution A, E[A], it is possible to calculate the coefficient of variation of inter-arrival as shown if the following equation:

$$C_A = \frac{\sigma[A]}{E[A]} \tag{9}$$

Further, the same can be applied for a stochastic distribution of variable service, S to determine the coefficient of variation of service, C_S. This can be assumed to be constant and thus $C_s = 1$. Through statistical analysis of requirements data, distribution A can be modeled to follow a probability distribution function to determine the characteristics of the arrival process.

As the variability of arrival is increased, the coefficients of variation affect the number in the queue and thereby the wait time, as illustrated in Figure 16 below.

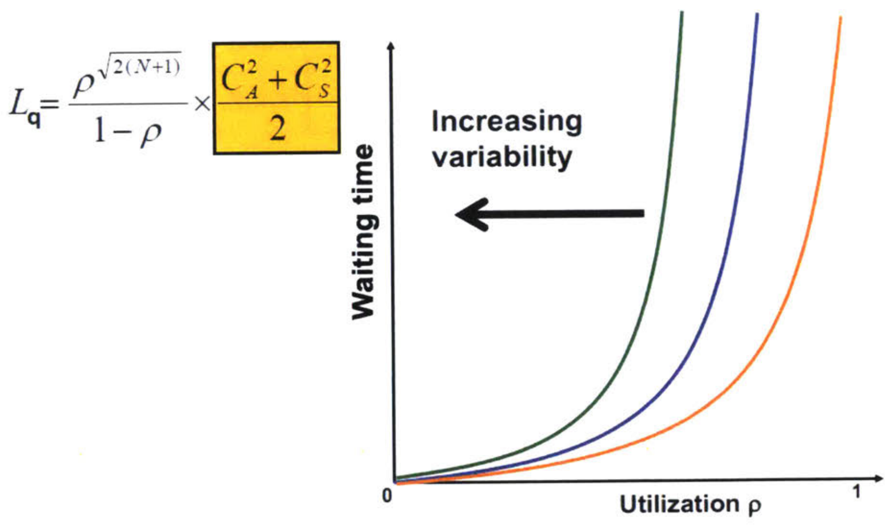


Figure 16. Increasing Variability of Requirements (Volatility) Increases the Queue Size and Waiting Time

Thus, the volatility and variability of requirements change has an increasing impact on the program schedule.

3.1.7 Discussion on Requirements Volatility as a Source of Hazards

The combined effects of requirements volatility increases schedule risk and defect rates. Schedule risk may lead to other sources of hazards being introduced into the system by management attempting to make-up lost schedule such as using employee overtime. This may further lead to compounding sources of induced hazards into the system during development as discussed earlier in Section 3.1.5.

3.2 Current Safety Standards

The current NASA philosophy on safety is geared towards reducing probabilities of failure to an acceptable level through reliability, simplicity, robustness, and fault tolerance. As briefly mentioned, crew-rated spaceflight focuses on safety and reliability as a primary objective, whereas scientific and robotic missions may have higher levels of risk acceptance commensurate to the program's mission and budget [5]. Risk and reliability analysts are understood to be critical for integration into the design phase to avoid adding in reliability features as an after-thought. However, it is acknowledged that risk and reliability analysts often perform their analysis on the design after it has been formulated, rather than designing with safety as a design driver.

Current practice specifies that preliminary reliability analyses occur as the design matures. The design and concept of operations should also be thoroughly examined for accident initiators and hazards that could lead to mishaps. Additionally, current practice calls for conservative estimates of likelihood and consequences of the hazards to be used as a basis for applying design resources to reduce the risk of failures. Finally, current practice aspires to fully consider all failure modes and take the entire system into account.

The NASA Systems Engineering Handbook characterizes a reliable system as one that "ensures mission success by functioning properly over its intended life" [5]. Effort should be made to reduce the probability of failure to the lowest acceptable level through simplicity, proper design, and proper application of reliable parts and materials. Additional consideration should be given to ensure robustness and fault tolerance, meaning it can tolerate failures and variations in its operating parameters and environments.

The level of safety and reliability that is designed and built into the system depends on the information needed and the type of mission. For crew-rated systems, this is the primary objective throughout the design process. For science missions, safety and reliability is usually commensurate with the funding and level of risk a program is willing to accept. These considerations should be an intricate part of the system design processes. Current methods attempt to maximize the benefit from the reliability analysis by integrating the risk and reliability analysts within the design teams. However, in many cases, the reliability and risk analysts perform the analysis on the design after it has been formulated. In such a case, safety and reliability features are added on or outsourced rather than designed in which may result in an unrealistic analysis that is not focused on risk drivers and does not provide full value to the design. Current analyses are designed to evolve to answer key questions about design trades as the design matures.

NASA-STD- 8729.1, Planning, Developing, and Maintaining an Effective Reliability and Maintainability (R&M) Program outlines engineering activities that should be tailored for each specific project. In the early phases of a project, risk and reliability analyses help designers understand the relationship between requirements, constraints, and resources. This is intended to uncover key relationships and drivers so they can be properly considered. Current standards call for designers to go beyond the requirements in order to understand any implicit dependencies that may emerge as the design concept matures. The standard acknowledges that the design requirements will not necessarily correctly capture all risk and reliability issues. It gives guidance that the systems engineer should develop a system strategy on how to allocate and coordinate reliability, fault tolerance, and recovery between systems both horizontally and vertically within the architecture to meet the total mission requirements. This is intended to enable designers to be aware of the impacts that their decisions have on overall safety.

The design and concept of operations should be thoroughly examined for accident initiators and hazards that could lead to potential mishaps. During the latter phases of a project, conservative estimates of likelihood and consequences of the hazards are used as a basis for applying design resources to reduce the risk of failures. The team uses risk assessments and reliability techniques to verify that the design is meeting its risk and reliability goals and to help develop mitigation strategies when the goals are not met or discrepancies occur [5].

3.3 Traditional Accident Models

Traditional accident models are chain-of-failure-events causality models. They work by decomposing the system into smaller components, examining and analyzing each component in isolation, and then combining the results in order to understand the behavior of the composed components. System reliability is usually evaluated by assessing the reliability of the individual components and then the component reliabilities are combined mathematically to evaluate the system reliability. Several assumptions are required for this traditional approach to be valid:

- Separation and individual analysis does not distort the phenomenon or property of interest
- Each component or subsystem operates independently. If events are modeled, then they are independent except for the immediately preceding and following events.
- Components act the same when examined separately as when they are playing their part in the whole.
- Components and events are not subject to feedback loops and other indirect interactions.
- The interactions among the components or events can be examined pairwise and combined into a composite value.

Such traditional hazard analysis methods are based on decomposition and therefore on these assumptions. The basic approach involved is to divide the system into components, assume that accidents are caused by component failure, calculate the probability of failure of each component separately, and later combine the analysis results (based on assumptions about the types of interactions among components that can occur) into a system reliability figure, which is assumed to be a measure of safety or risk. Common approaches that use this theoretical basis are Failure Modes and Effects Analysis (FMEA) and Failure Modes and Effects Criticality

Analysis (FMECA). Further modifying the event-chain model to include “deviations” as the event or condition to be considered, is the basis for fault tree analysis (FTA), event tree analysis (ETA), fault hazard analysis (FHA), and hazard and operability analysis (HAZOP).

Preliminary Hazard Analysis (PHA) – Preliminary hazard analysis is a “what if” process that considers the potential hazard, initiating event scenarios, effects, and potential corrective measures and controls. The objective is to determine if the hazard can be eliminated, and if not, how it can be controlled. A PHA is performed early based on the functions performed during the mission [5].

Failure Modes and Effects Analyses (FMEAs) – FMEAs are bottom-up analyses that identify the types of failures that can occur within a system and identify the causes, effects, and mitigating strategies that can be employed to control the effects of the failures [5].

The initial estimates of failure rates or failure probability might be based on comparison to similar equipment, historical data (heritage), failure rate data from handbooks, or expert elicitation. They do not account for emergent properties of including existing parts in a new system.

Various mitigation efforts are used as to improve the results of traditional analyses. If accidents are assumed to be caused by failure events, then it makes sense that the approach to preventing accidents is to eliminate the events or to put barriers between the events so that one failure does not cause another event down the chain. These include accident prevention design techniques used such as redundancy, barriers, high component integrity and overdesign, fail safe design, and, for humans, the use of operational procedures, checklists, and training. These design features are used to reduce the probability of the failure events occurring or of them propagating.

The failure events require the assumption that they are stochastic for probabilities or likelihood to be determined. However, software and humans do not satisfy this assumption. The greatly increased complexity in our systems today (which is made possible primarily by the use of software) is creating systems where the approach is no longer as effective. It is much more difficult today to anticipate, understand, plan, and guard against all potential system behavior before operational use of our systems. Complexity is creating "unknowns" that cannot be identified by breaking the system behavior into chains of events. In addition, complexity is leading to important system properties (such as safety) not being related to the behavior of individual system components but rather to the interactions among the components. Accidents can occur due to unsafe interactions among components that have not failed and, in fact, satisfy their requirements.

Problems did not stem from individual component failures but from flaws in the system engineering process that resulted in flawed system designs. Decomposition-based analysis cannot identify such causes of accidents including human error, software requirements errors, and system design flaws.

3.4 Systems-Based Hazard Theory

As introduced earlier, systems theory was developed post-World War II as a means to understand and cope with nascent complexity of increasingly intricate technological systems. In particular, some of the initial uses of systems theory were part of the 1950s missile development programs. The following definitions are required to understand systems theory:

System – A set of things (referred to as system components) that act together as a whole to achieve some common goal, objective, or end.

State – A set of relevant properties describing the system at any time. The components of the state that are relevant depend on how the boundaries are drawn between the system and its environment.

Environment – Usually defined as the set of components (and their properties) that are not part of the system but whose behavior can affect the system state. Therefore the system has a state at a particular time and the environment has a state. The concept of an environment implies that there is a boundary between the system and its environment.

A system can be divided into subsystems and components, but may also be part of a larger system. This proposes that a "system of systems" requires a treatment different than that of the original system. However general system engineering and analysis methods and techniques are applicable to all systems, including a "system of systems" which is actually still just a system as much as any other. Unique aspects of systems theory that make it interesting for application to hazard analysis are that system components do not necessarily need to be directly connected to interact, making non-linear interdependencies possible. Systems can be treated as whole, rather than a sum of their parts. This means that systems can exhibit emergent properties, a crucial aspect of applying systems theory to hazard analysis.

Emergent Properties – These properties "emerge" when the components of a system interact, which may be different than the properties resulting from the summation of the individual components. Emergent properties arise from relationships among the parts of the system, that is, by how they interact and fit together. Full treatment of emergent properties can only be done by taking all their technical and social aspects into account that effect the system as illustrated in Figure 17.

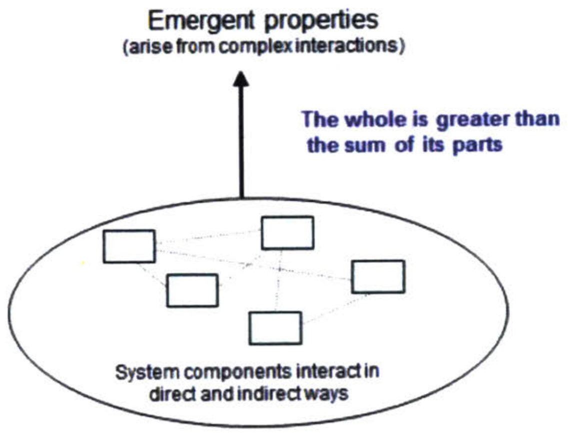

Emergent properties
(arise from complex interactions)
The whole is greater than the sum of its parts
System components interact in direct and indirect ways

Figure 17. Emergent Properties Arise From Complex System Interactions [4]

When a controller is added to the system, it provides control actions and gets feedback to determine the impact of the control actions. The basis of theory implies that this arrangement acts like a standard feedback control loop as illustrated in Figure 18.

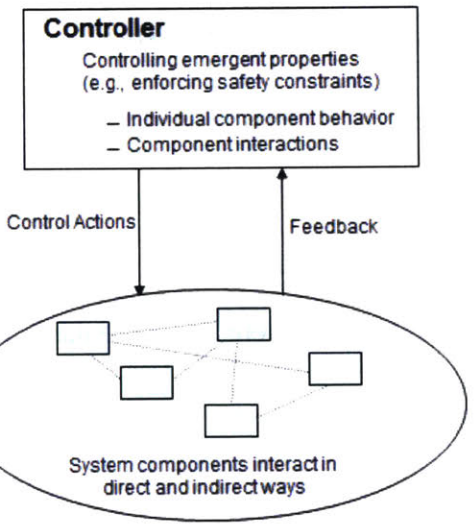

Controller
Controlling emergent properties
(e.g., enforcing safety constraints)
– Individual component behavior
– Component interactions
Control Actions
Feedback
System components interact in direct and indirect ways

Figure 18. Control Actions and Feedback Imposed on a System [4]

If safety is considered early in the design process, relatively simple design features can act as controls to improve safety without also increasing complexity and cost. The effect is maximized by including it in a design that completely eliminates all hazardous states. A compromise to this may be a design where such hazards are rare and easy to handle if they do occur. On a final level, a design may react to the occurrence of hazards by minimizing the resulting damage (although the least desirable option it may be required in particular degraded modes of operation).

Three types of design techniques are currently provided in an attempt to "control" safety. The first is to use an intrinsically or inherently safe design. Such a design is not capable of producing the hazard of concern. For example, the system design may not have sufficient energy to cause an explosion. Potential toxic substances may be replaced with non-toxic substitutes. Risk of lost information over a communication network may be solved with more direct communication methods where feasible. However, if an intrinsically safe design is not possible, the use of passive or active control mechanisms may be possible.

> **Passive controls** – A passive control mechanism is a physical interlock that prevents the system from entering a hazardous state without any overt control action. They do not require a positive action in order to be effective, but instead rely on basic physical principles, such as gravity.

Such controls are often designed into the system to ensure that it essentially fails into a safe state, hence the term *fail-safe*.

> **Active Control** – in contrast to passive control, active control requires a (hazardous) condition to be detected and corrected. These often involve the use of computers and the basic control loops.

Within a system, an active control commands, directs, or regulates the behavior of other devices or processes using control loops as shown in Figure 19. The controller hosts a process model to understand the feedback and a control algorithm to provide the appropriate response.

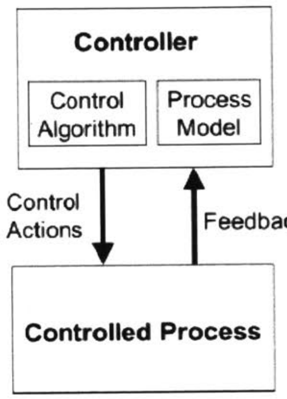


Figure 19. Basic Control Loop [4]

In order to control a process, the controller must have a purpose, which can include maintaining constraints on the behavior of the controlled process. Control actions are implemented by *actuators*. In order to know what control actions are necessary, the controller must be able to determine the current state of the system. Such information about the state of the system is provided by *sensors* and is called *feedback*.

As a comparison, passive controls are more reliable that active controls. Passive controls usually depend on physical principles providing for a simpler design, while active controls depend on less reliable detection and recovery mechanisms usually resulting in more complex designs. For instance, in an active control, the failure or hazardous state may not be detected or it may be detected but not corrected or it may not be corrected in time to prevent a loss. Unfortunately, passive controls tend to be more restrictive in terms of design freedom. Thus they are not always practical to implement in complex systems.

Control loops have two operational modes that are important for understanding STPA: feedback control and feed-forward control. Often feedback and feed-forward control are used in tandem. The following definitions are important to understanding control modes:

> **Feedback Control** – this mode of control uses feedback to update the controller's model of the process. Other information may also be part of the process model, such as environmental information (outside air temperature) or basic information about natural temperature fluctuations. The controller uses the process model to decide what control actions to provide, in order to change the state of the (controlled process) to stay within the required range.
>
> **Feed-forward Control** – this mode of control uses a model of the current state of the process and the future and then provides a control action without specific feedback to identify the need. This is used to anticipate an upcoming state.
>
> **Shaping Actions** – control actions that attempt to maintain a safe state by maintaining assumptions, preventing hazards (unsafe states in the controlled process), and controlling migration of the process to states of higher risk. Shaping actions provide a type of feed-forward control and may be passive or active.
>
> **Hedging or Contingency Actions** – control actions that provide a type of feedback control as they involve monitoring the system for signs of increasing risk and responding accordingly. Hedging actions that include monitoring the effectiveness of the shaping actions provide a combination of feed-forward and feedback control.

Controllers may use the following means to reduce or eliminate hazards from component failures and unsafe interactions [15]:

1. Design: this includes redundancy, interlocks, barriers, or fail-safe design.

2. Process: this includes development processes, manufacturing processes and procedures, maintenance processes, and general system operating processes.
3. Social controls: this includes government regulation, culture, insurance, law, courts, or individual self-interest. Human behavior can be partially controlled through the design of the societal or organizational incentive structure.

System-Theoretic Accident Model and Processes (STAMP) is a relatively new model based on systems theory. It expands the traditional model of causality beyond a chain of directly-related failure events or component failures to include more complex processes and unsafe interactions among system components. STAMP also provides the theoretical foundation for STPA among other hazard analysis tools by treating safety as a dynamic control problem rather than a failure prevention problem. Noted advantages of using STAMP are [15]:

- It works on very complex systems because it works top-down rather than bottom-up.
- It includes software, humans, organizations, safety culture, and more as causal factors in accidents and other types of losses without having to treat them differently or separately.
- It allows creating more powerful tools, such as STPA, accident analysis (CAST), identification and management of leading indicators of increasing risk, and organizational risk analysis among other things.
- It includes chain-of-failure events model as a subset so tools built on STAMP can include as a subset all the results derived using the older safety analysis techniques making it a powerful characteristic.
- It can be used for any system property, including cybersecurity, because STAMP applies to any emergent property.

STAMP provides a set of assumptions about how accidents occur that underlie other tools mentioned already: STPA (System-Theoretic Process Analysis) and CAST (Causal Analysis based on Systems Theory).

CAST – is a retroactive analysis method that examines an accident/incident that has occurred and identifies the causal factors that were involved.

STPA – is a proactive analysis method that analyzes the potential cause of accidents during development so that hazards can be eliminated or controlled.

STPA-type analysis is increasingly needed for today's complex, software-intensive systems and is most effective when integrated into the organization. Structuring the STPA process into systems engineering processes is the most effective. Additional benefits of STPA over traditional hazard analysis techniques are [15]:

- Very complex systems can be analyzed. "Unknown unknowns" that were previously only found in operations can be identified early in the development process and either eliminated or mitigated. Both intended and unintended functionality are handled.
- Unlike the traditional hazard analysis methods, STPA can be started in early concept analysis to assist in identifying safety requirements and constraints. These can then be used to design safety (and security) into the system architecture and design, eliminating the costly rework involved when design flaws are identified late in development or during operations.

Many evaluations and comparisons of STPA to more traditional hazard analysis methods, such as fault tree analysis (FTA), failure modes and effects criticality analysis (FMECA), event tree analysis (ETA), and hazard and operability analysis (HAZOP) have been done. STPA consistently provides better results, costs less, and requires less time. In all of these evaluations, STPA found every causal scenario found by more traditional analyses, but also identified many more, often software-related and non-failure type scenarios that the traditional methods did not find. For these reasons, an exponentially increasing amount of companies are now starting to use STPA for other system properties such as quality, security, and production engineering [15].

3.5 Systems-Based Hazard Analysis Applicability

This section covers real-world scenarios where STPA is applicable. Section 3.5.1 covers the application of STPA to the failure of the Mars Polar Lander in 1999. Section 3.5.2 covers the Boeing 787 Dreamliner Battery Failure Application.

3.5.1 Mars Polar Lander Failure Application

On January 3, 1999, NASA launched the Mars Polar Lander, a 290-kilogram robotic lander designed to study the soil and climate of Planum Australe, a region near the southern pole of Mars. On December 3, 1999, after completing the descent phase, the lander communication was never reestablished with Earth. A post-mortem analysis determined the most likely root cause was premature termination of the descent engines prior to the lander touching the surface, causing it to strike the planet at a velocity outside of the design envelope as shown in Figure 20.

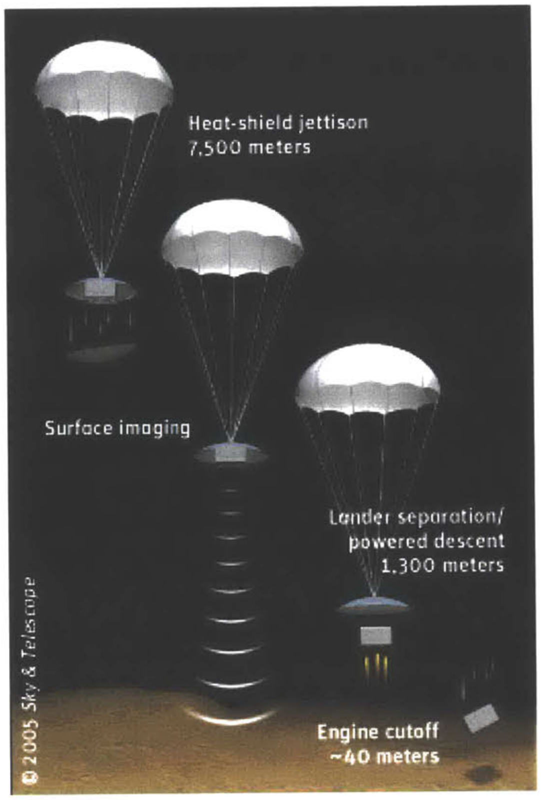


Figure 20. Mars Polar Lander Accident Illustration

Typical means of arresting descent speed involve the Martian atmosphere, a parachute, and descent engines. As soon as the spacecraft lands, the software must immediately shut down the descent engines to avoid damage to the spacecraft. The feedback for the engine control unit software to process this command was provided by sensors on the landing legs that were highly sensitive. Upon deployment of the landing legs, false sensor signals generated noise as feedback to the engine control unit [16]. This was expected behavior but was not in the software requirements.

Traditional chain-of-failure models did not detect the lacking requirement. Systems-based analysis however offers a means to detect such an emergent property of the landing sequence which may have prevented the missing requirement. More thoroughly, the software was operating too soon in the descent sequence, another property that is captured in STPA but not

FMEA. In an attempt to more evenly distribute the computational load on the control unit processor, the software engineers decided to begin sensing feedback from the landing legs earlier than designed. For this reason, the software thought that the spacecraft had already landed and shut off the descent engines while the spacecraft was still 40 meters about the planet surface [4].

3.5.2 Boeing 787 Dreamliner Battery Failure Application

On January 7, 2013, smoke was discovered in the aft cabin of a Japan Airlines (JAL) Boeing 787-8 while parked at a gate at Boston's Logan International Airport (BOS) in Massachusetts. Maintenance crew in the cockpit also observed that the auxiliary power unit (APU) had automatically shut down. Upon immediate inspection of the aft electronic equipment bay, heavy smoke was identified coming from the cover of the APU battery case as well as two distinct flames at the electrical connector located at the front. The battery model used for the APU was also the same model used for the 787 main battery.

Eleven days later, on January 16, 2013, an incident involving the main battery occurred aboard a 787 airplane being operated in Japan during a flight from Yamaguchi to Tokyo. The pilots reacted quickly to make an emergency landing at Takamatsu Airport (TAK), Japan, shortly after takeoff. After the BOS and TAK events, the FAA subsequently grounded the US-based 787 fleet.

Prior to product release of the 787, as part of a compliance demonstration with specific FAA battery requirements and 14 CFR Part 25 airworthiness standards, Boeing performed a safety assessment to determine the potential hazards of various Electrical Power System (EPS) failure conditions. Boeing determined that the rate of occurrence of cell venting for the 787 battery would be about 1 in 10 million flight hours. However, at the time of the BOS and TAK incidents (both of which involved cell venting), the in-service 787 fleet had accumulated less than 52,000 flight hours.

The NTSB determined the root cause of the BOS incident was an internal short circuit within a cell of the APU lithium-ion battery, which led to thermal runaway that cascaded to adjacent cells, resulting in the release of smoke and fire [17]. However, the problems did not stem from individual component failures but from flaws in the system engineering process that resulted in flawed system designs. The incident resulted from Boeing's failure to incorporate design requirements to mitigate the most severe effects of an internal short circuit within an APU battery cell and the FAA's failure to identify this design deficiency during the type design certification process.

Traditional decomposition-based post-accident analyses cannot identify these types of causes of accidents which include human error, software requirements errors, and system design flaws. In the event of a battery malfunction, part of the environmental control system was designed to remove any smoke through cooling duct fans by actuating electric valves. However, the unit providing power to the environmental control system simultaneously shut down due to the battery malfunction. As a result, the valves could not be actuated and smoke being generated by the APU battery could not be effectively directed outside the passenger cabin and battery compartment [4]. It is possible that the emergent properties and human errors in the systems engineering leading up to these incidents may have been captured by a systems-based hazard analysis such as STPA.

Chapter 4

4 Hazard Source Data Analysis and Simulation Results

This chapter covers the data gathered on requirements for the rocket test controller, analysis of the data, modeling of the data, and the simulation results.

4.1 Requirements Data Collection

Literature on requirements volatility shows that it increases schedule pressure and defect rates in software production. To determine if this is a potential source in the rocket test controller being studied, data is compiled on the requirements.

The test controller has four major sources of requirements that are broken into four categories:

1. Overall System
2. Hardware
3. System Software
4. User Apps and Displays (UAD) Software

Using the source revisions published for each requirement category and methodology established in Chapter 3.1.4, the requirements can be counted. Figure 21 shows the total number of requirements for the rocket test controller over a five year period.

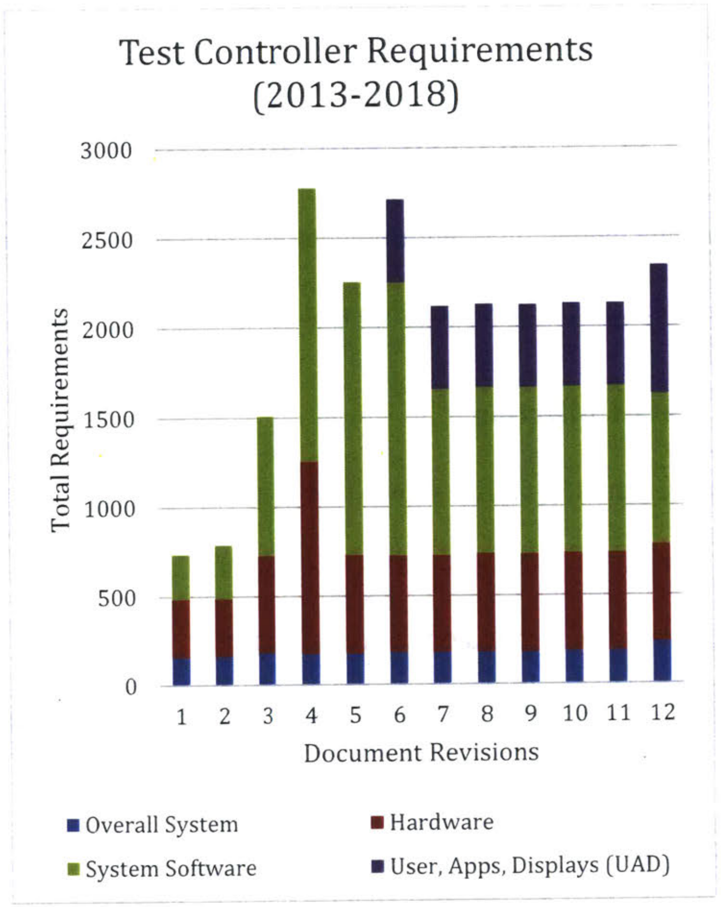


Figure 21. Test Controller Requirements by Type

The number of requirements categorizes this as a large project. The graph appears to show an exponential increase in the total number of requirements for the rocket test controller as the program matured. Additional introspection of the software components of the system reveals a more drastic case as shown in Figure 22.

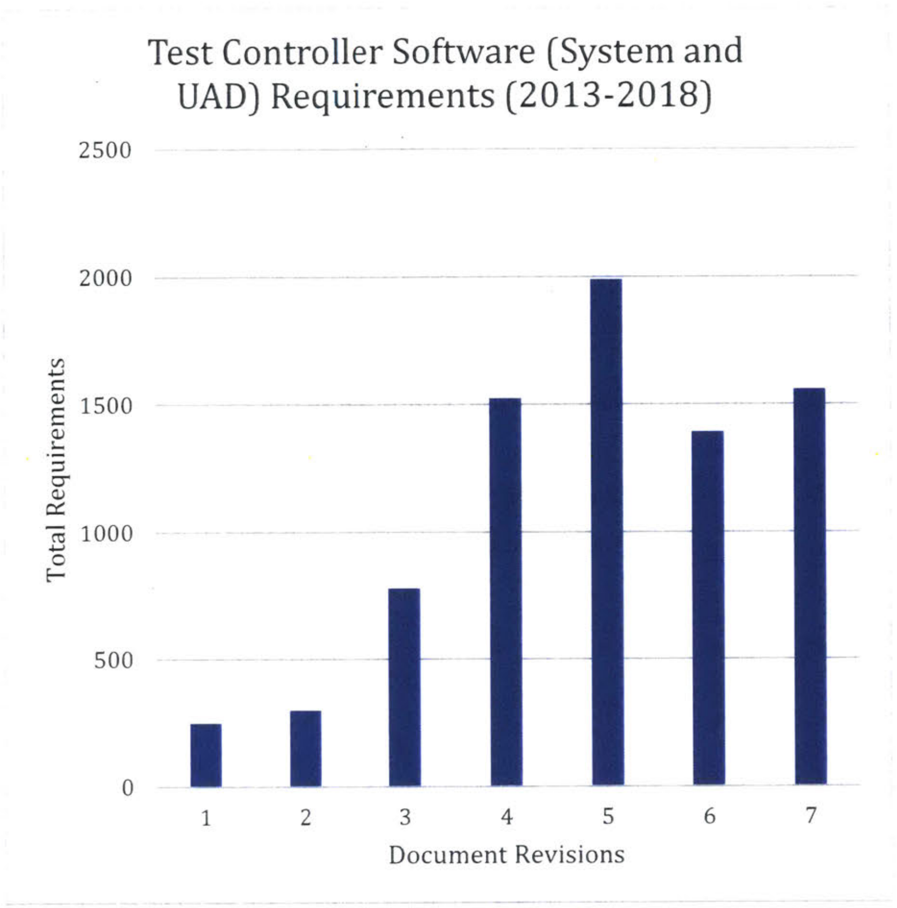


Figure 22. Test Controller Software Requirements Over Five Years (System and User Apps and Displays)

The graph reveals an exponential increase and later fluctuation in software requirements. This appears consistent with project maturation for a lower-tier end product undergoing concurrent engineering. The later fluctuations are an indication of software requirements volatility that requires further analysis as a potential source of increased risk to inducing hazards into the system.

4.2 Requirements Data Analysis

Since revisions are not released on a regular time basis and are released on an as-needed basis, the revisions do not provide an accurate representation of how the rate of requirements count increase or decrease. Further, the number of requirements does not account for all of the work needed because it does not include requirements modifications or deletions. To do that requires analyzing a constant time scale and a count of changes (additions, deletions, and modifications).

Analyzing the data for the overall system over an eight month period reveals volatility ranging from <1% to 26.8% for the total system requirements from all four categories (overall system, hardware, system software, and user apps and displays).

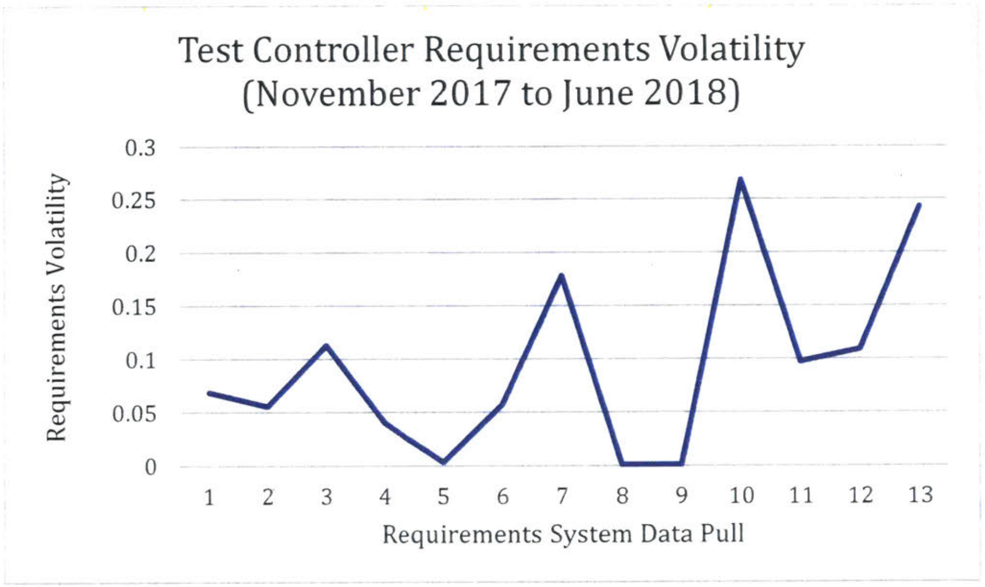


Figure 23. Test Controller Requirements Volatility over an Eight Month Window

Narrowing the scope to the software over a seven month period during the same timeframe reveals volatility ranging 6.1% to 39.8% for the combination of both system software and user apps and displays requirements.

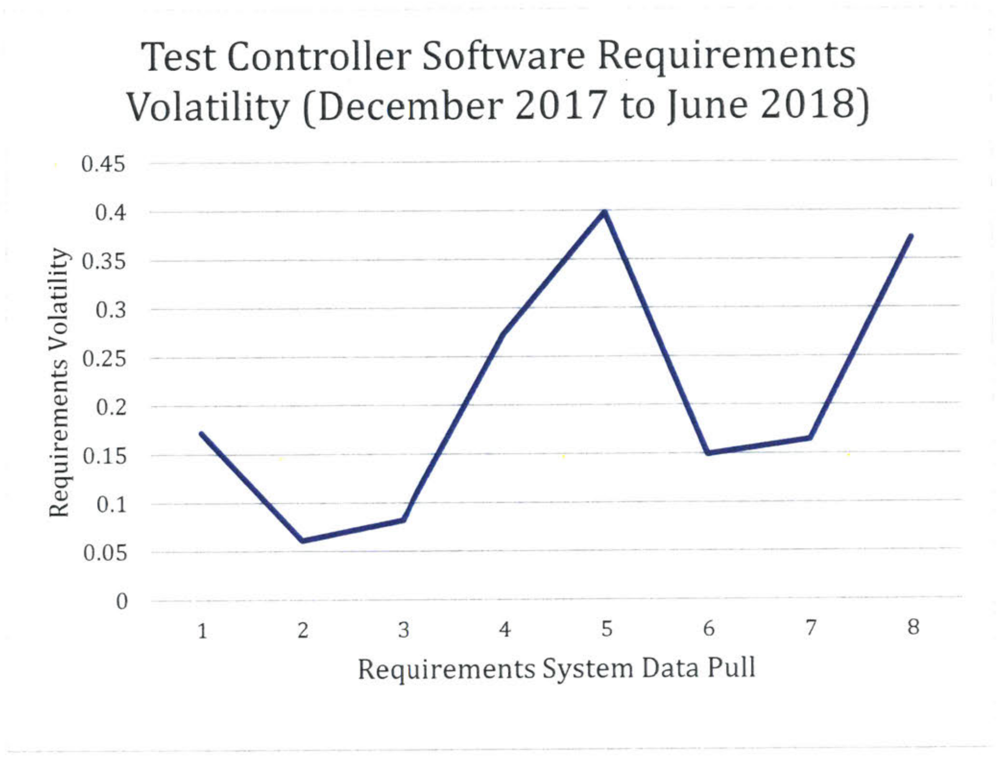


Figure 24. Test Controller Software Requirements Volatility over a Seven Month Window.

This volatility represents a risk to program schedule and further increases risk to inducing defects that cause safety-critical hazards.

To compare the two, the graphs from Figure 23 and Figure 24 may be overlaid. The result indicates that software requirements are a major source of the total system requirements volatility as shown in Figure 25.

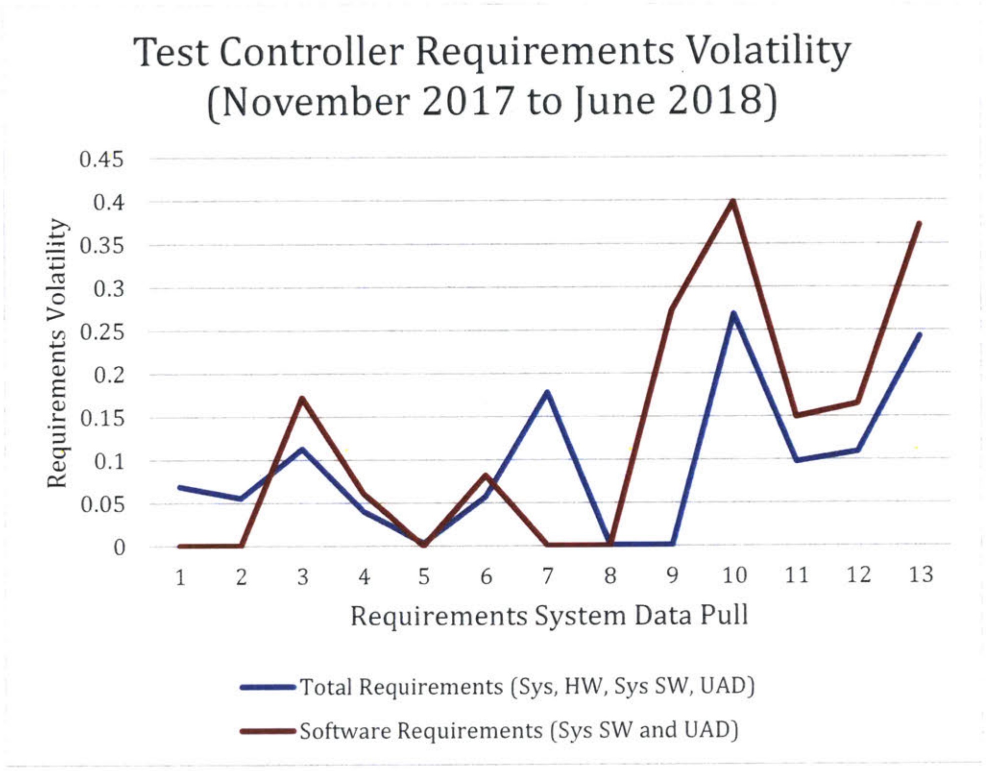


Figure 25. Test Controller Requirements Volatility Overlay during an Eight Month Window

Intuitively, this makes sense, because the basis for the hardware remains relatively stable as the major components and connections to the rocket are established but the nuanced changes of the rocket design are handled by the test controller's software.

4.3 Data Comparison

The risk to schedule that requirements volatility poses to a program is a particular source of risk that can result in potential induced hazards into the end product system. Requirements data may be compared to schedule data to garner further insight. Figure 26 shows a graph of the schedule change data collected from monthly management reviews for a seven month period. Negative numbers are associated with schedule improvements, whereas a positive number of days is associated with a schedule delay.

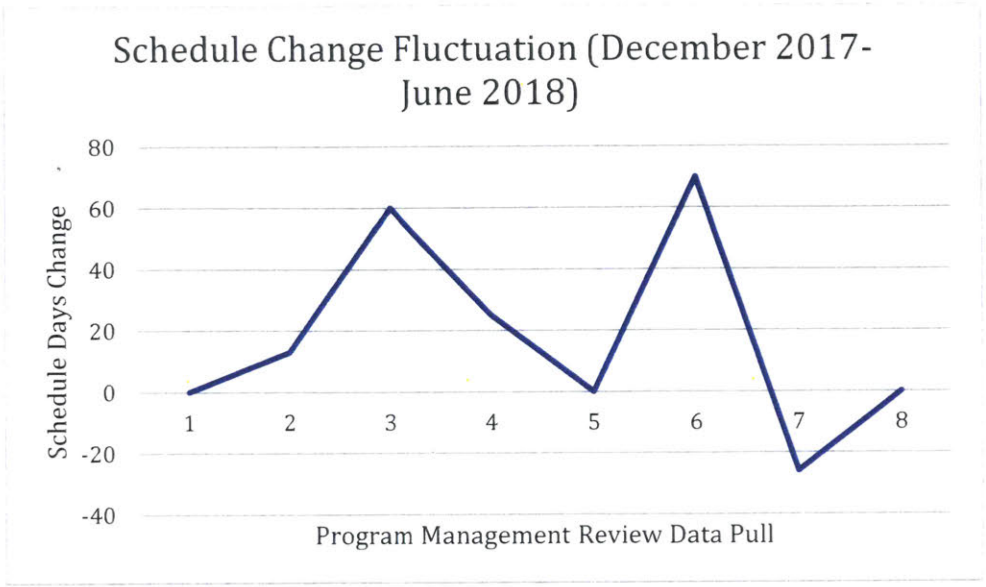


Figure 26. Schedule Changes for the Test Controller System over a Seven Month Window

The data shows two 60 to 70 day schedule delays within the seven month window. Further context reveals managerial decisions as outlined in earlier discussions in Chapter 3.1.5 were applied to "course correct" the schedule to make up for the schedule delay, as represented in the respective decreases on the graph in Figure 26.

A comparison of the software requirements volatility to changes in the schedule is given in Figure 27.

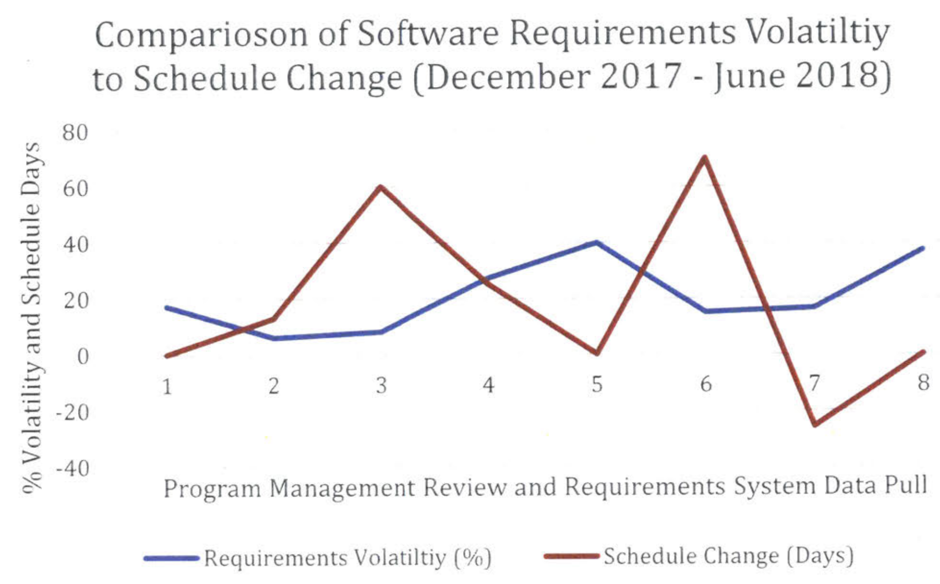


Figure 27. Comparison of Software Requirements Volatility to Schedule Delays

The graph in Figure 27 shows that a schedule delay reported in period 6 is correlated to a 40% software requirements volatility in period 5. Schedule delays are further associated with cost overruns and lower quality. Although lost schedule days may be regained through managerial efforts, these techniques perhaps bring negative conditions that are conducive to inducing hazards into the software development.

4.4 Modeling and Simulation Results

To further investigate potential scenarios and the relationship with schedule delays, a model may be constructed using queuing theory as outlined in Chapter 3.1.6.

A baseline model may be created using that has no variation, using the following values show in Table 4 below.

Table 4. Baseline Model Values for G-G-N Queuing Model with no Variation

Variable	Value	Units
Arrival Rate, λ	112	reqs/month
Service Rate, μ	5	reqs/month
Number of developers, N	30	servers
Capacity utilization, ρ	0.746667	
Coefficient of Variation of Arrivals, C_A	1	
Coefficient of Variation of Service, C_S	1	
Standard Deviation, σ[A]	1	reqs/month
Expected Value, E[A]	1	reqs/month

Interpolating data from monthly management review timelines, the original burn down plan for requirements can be estimated at 2700 requirements over two years. This results in an original arrival rate of 112.5 requirements per month. Assuming a 40-hour work week and 20 work days per month, results in 800 work hours per month. This results in an estimated 80 m-hours, on average, needed per requirement. Based on these values from Table 4 and calculation of m-hours required based on the given service rate, the following queue results are obtained:

Table 5. Results of G-G-N Queuing Model with no Variation

Variable	Value	Units	Time	Units
Length of Queue, L_q	0.395646	reqs	31.65168	hours
Length of Service, L_s	22.4	reqs	1792	hours
Total Production, L	22.79565	reqs	1823.652	hours

The result in the total production time with no variability is 1823 m-hours needed to be scheduled amongst employees as per the original burn down plan with no variability.

As a comparison, the G-G-N queue may be modified to show the impact of variability on the schedule. Assuming due to variability, a doubled effort through factors such as overtime and process streamlining is provided by management, causing an effective increase in the service rate, μ, from 5 requirements per month to 10 requirements per month. The mean and standard deviation are calculated from the requirements change data to determine the new arrival rate, λ, and to characterize the arrival cumulative distribution function, A.

Table 6. Model Values for G-G-N Queuing Model with Variation Data

Variable	Value	Units
Arrival Rate, λ	294	reqs/month
Service Rate, μ	10	reqs/month
Number of developers, N	30	servers
Capacity utilization, ρ	0.98	
Coefficient of Variation of Arrivals, C_A	0.6156	
Coefficient of Variation of Service, C_S	1	
Standard Deviation, σ[A]	181	reqs/month
Expected Value, E[A]	294	reqs/month

Based on the values from Table 6 and the calculation of m-hours required based on the given service rate, the following queue results are obtained:

Table 7. Results of G-G-N Queuing Model with Variation Data

Variable	Value	Units	Time	Units
Length of Queue, L_q	29.405	reqs	2352	hours
Length of Service, L_s	29.4	reqs	2352	hours
Total Production, L	58.805	reqs	4704	hours

The results from Table 5 can be compared to the results from Table 7 to determine the impact to schedule induced by the volatility of software requirements over the seven month window. The result is a total schedule increase (delay) of 12 days per the model as shown below in Table 8.

Table 8. Schedule Impact Results from Requirements Volatility

Value	Units
2880.767494	m-hours
360.0959367	m-days
12.00319789	total days increase with N servers

This model may be used to examine various scenarios using a simulation. To perform a simulation, first, the requirements distribution must be determined from the data in Chapter 4.1 to determine the requirements arrival rate, λ_i, measured in units of work per unit time. Fitting the data to an applicable best-fit distribution required checking several common distribution types to arrive at one that best approximates the arrival set, including normal, exponential Beta, Gamma, Extreme Value, and Weibull. Comparing Probability-Probability Plot (P-P Plot) and Quantile-Quantile Plot (Q-Q Plot) results among distributions arrives at a generalized extreme value (GEV) distribution introduced by Jenkinson (1955) that best characterized the cumulative distribution function [18]:

$$F(s;\, \xi) = \begin{cases} \exp(-(1+\xi s)^{-1/\xi} & \xi \neq 0 \\ \exp(-\exp(-s)) & \xi = 0 \end{cases} \tag{10}$$

using the standardized variable,

$$s = \frac{x - \mu}{\sigma} \tag{11}$$

where,

$\mu \in \mathbb{R}$, is the location parameter

$\sigma > 0$, is the scale parameter

$\xi \in \mathbb{R}$, is the shape parameter

The software requirements changes data is fit to a distribution with the values $s =$ 214.63 and $\xi = 133.40$ as shown in graph of the CDF in Figure 28 below.

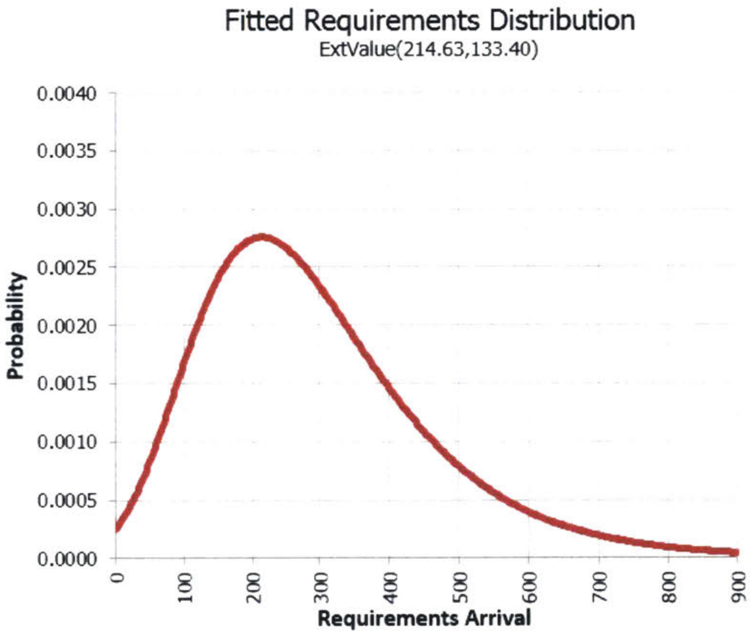


Figure 28. Fitted GEV CDF of Requirements Changes

Using the distribution as a substitute for the arrival rate, the risk to schedule delays from requirements volatility can be simulated. When iterated over 100 times, the probability of schedule delays due to requirements volatility is >64% as shown in Figure 29 below.

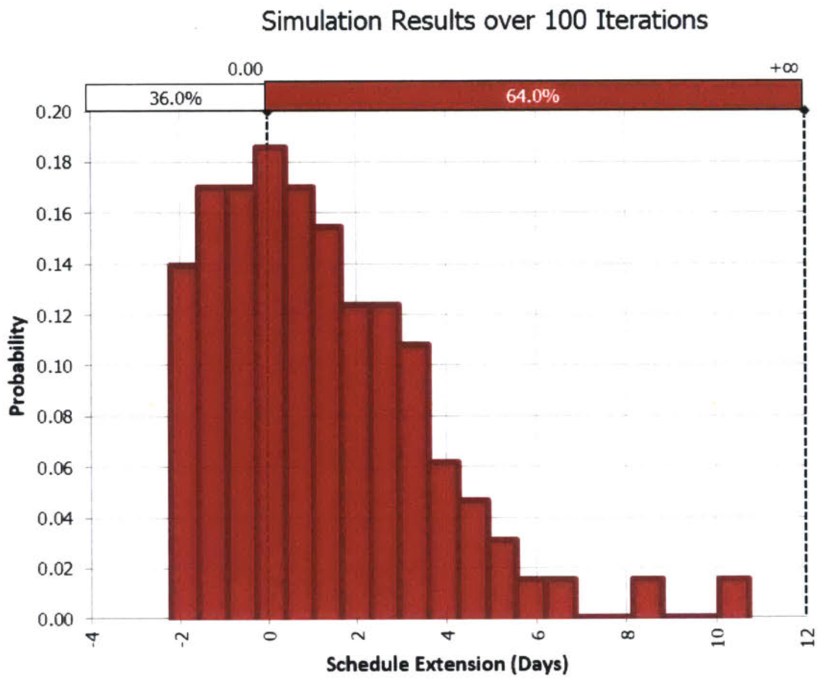


Figure 29. Schedule Delay Risk Simulation Results

Intuition behind the results suggests that there is a 36% probability that requirements may change in a drastic manner by way of more deletions than there are additions or modifications, resulting in some reduction in schedule. In the majority of cases, the simulation of requirements volatility on schedule impact indicates adverse schedule sway.

This page has been intentionally left blank.

Chapter 5

5 Analysis Approach – Systems-Theoretic Process Analysis

This chapter covers the methodology and approach to implement STPA on the rocket testing controller. Section 5.1 provides a thorough description of the STPA methodology with a primer on how to apply each step. Section 5.2 provides an overview of the strategy for STPA implementation on the rocket testing controller and means of analysis.

5.1 Description of STPA Methodology

This section describes the methodology to complete a study using System-Theoretic Process Analysis (STPA). The STPA methodology consists of four steps:

1. Define Purpose of the Analysis
2. Model the Control Structure
3. Identify Unsafe Control Actions
4. Identify Loss Scenarios

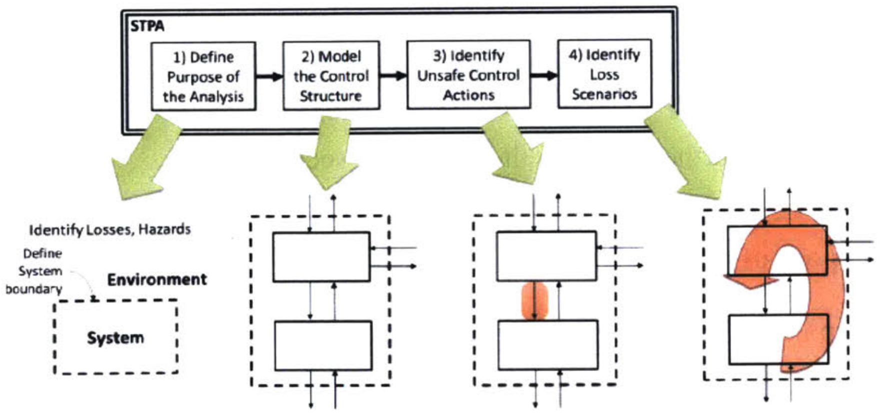

STPA
1) Define Purpose of the Analysis
2) Model the Control Structure
3) Identify Unsafe Control Actions
4) Identify Loss Scenarios
Identify Losses, Hazards
Define System boundary
Environment
System

Figure 30. Visualization of STPA Steps 1 through 4 [4]

Together the four steps form an iterative process that is continuously refined as show in Figure 31 below.

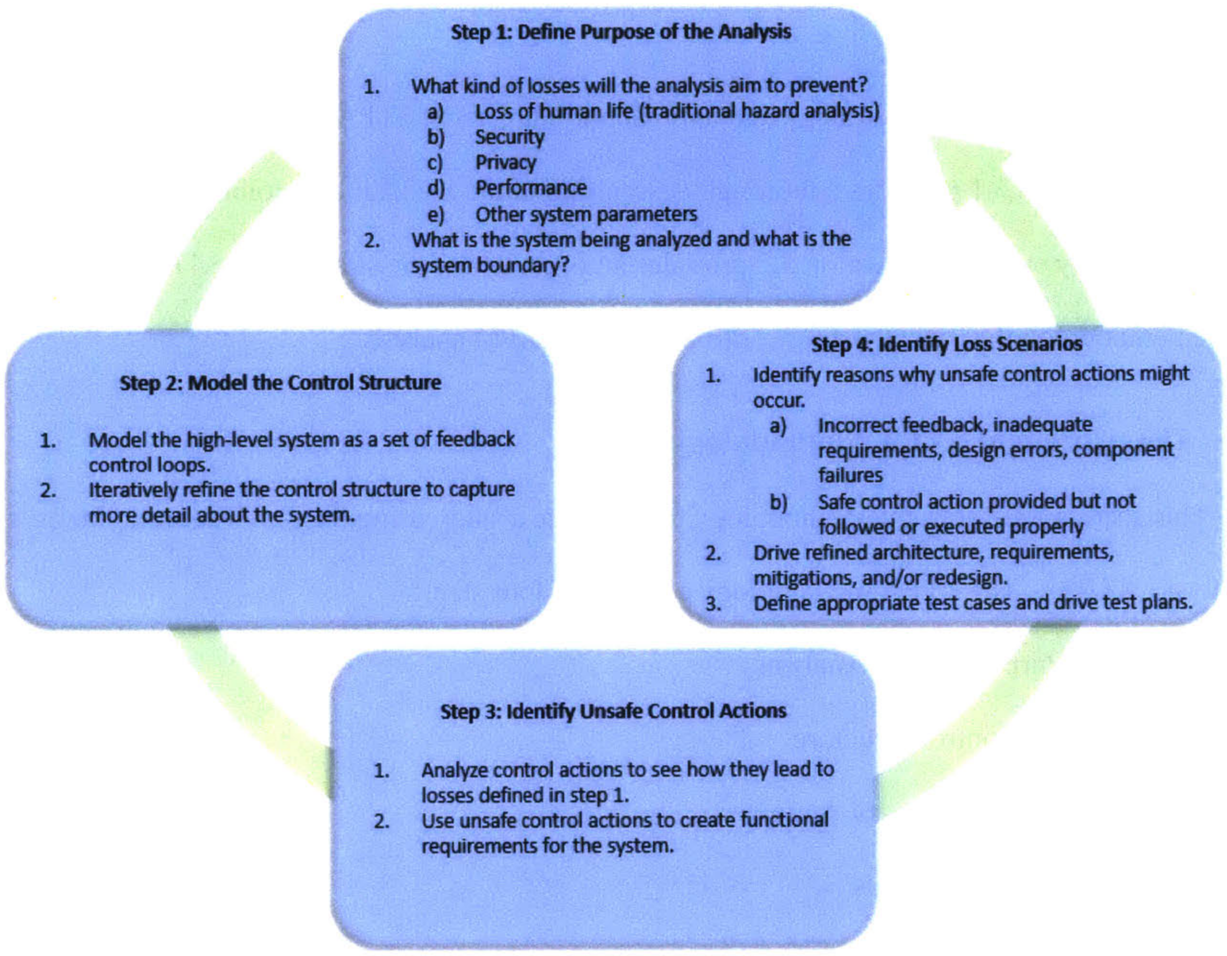

Step 1: Define Purpose of the Analysis
1. What kind of losses will the analysis aim to prevent?
a) Loss of human life (traditional hazard analysis)
b) Security
c) Privacy
d) Performance
e) Other system parameters
2. What is the system being analyzed and what is the system boundary?
Step 2: Model the Control Structure
1. Model the high-level system as a set of feedback control loops.
2. Iteratively refine the control structure to capture more detail about the system.
Step 3: Identify Unsafe Control Actions
1. Analyze control actions to see how they lead to losses defined in step 1.
2. Use unsafe control actions to create functional requirements for the system.
Step 4: Identify Loss Scenarios
1. Identify reasons why unsafe control actions might occur.
a) Incorrect feedback, inadequate requirements, design errors, component failures
b) Safe control action provided but not followed or executed properly
2. Drive refined architecture, requirements, mitigations, and/or redesign.
3. Define appropriate test cases and drive test plans.

Figure 31. STPA Iterative Methodology and Steps

These steps are defined in further detail in the following sections.

5.1.1 Define Purpose of the Analysis

The first step in STPA is to define the purpose of the analysis. This step includes the following sub-steps, as shown in Figure 32:

1. What kind of losses will the analysis aim to prevent?

    a) Loss of human life (traditional hazard analysis)

b) Security

c) Privacy

d) Performance

e) Other system parameters

2. What is the system being analyzed and what is the system boundary?

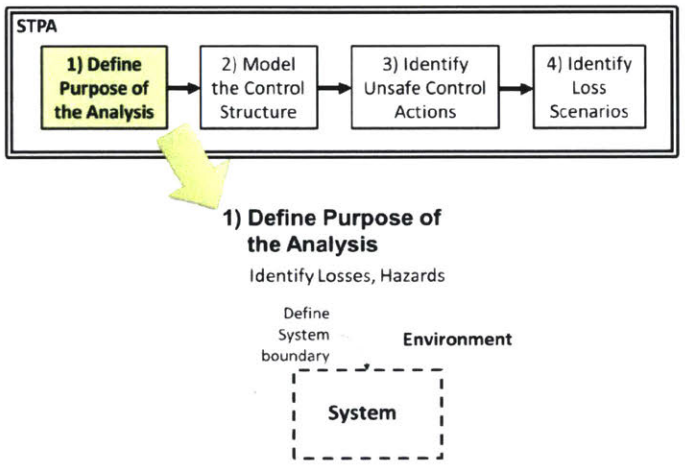

STPA
1) Define Purpose of the Analysis
2) Model the Control Structure
3) Identify Unsafe Control Actions
4) Identify Loss Scenarios
1) Define Purpose of the Analysis
Identify Losses, Hazards
Define System boundary
Environment
System

Figure 32. STPA Step 1: Define the Purpose of the Analysis [4]

5.1.2 Model the Control Structure

Step 2 of STPA is modeling the control structure. This step includes the following sub-steps:

1. Model the high-level system as a set of feedback control loops.

2. Iteratively refine the control structure to capture more detail about the system.

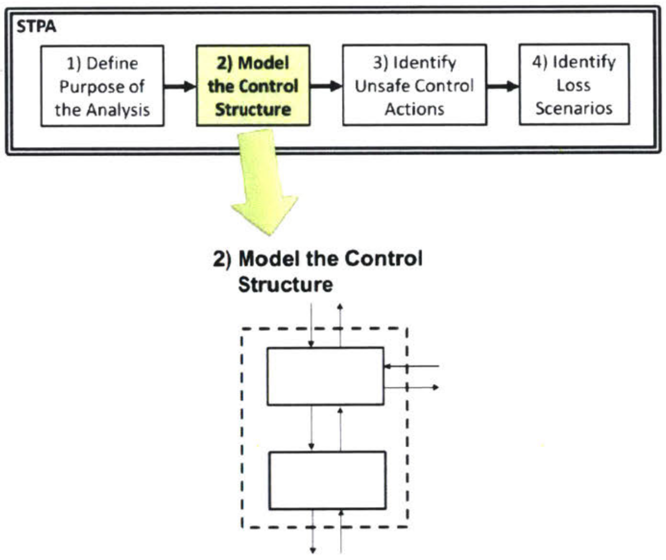

STPA
1) Define Purpose of the Analysis
2) Model the Control Structure
3) Identify Unsafe Control Actions
4) Identify Loss Scenarios
2) Model the Control Structure

Figure 33. STPA Step 2: Model the Control Structure [4]

5.1.3 Identify Unsafe Control Actions

Step 3 of STPA is identifying unsafe control action. This step includes the following sub-steps:

1. Analyze control actions to see how they lead to losses defined in step 1.
2. Use unsafe control actions to create functional requirements for the system.

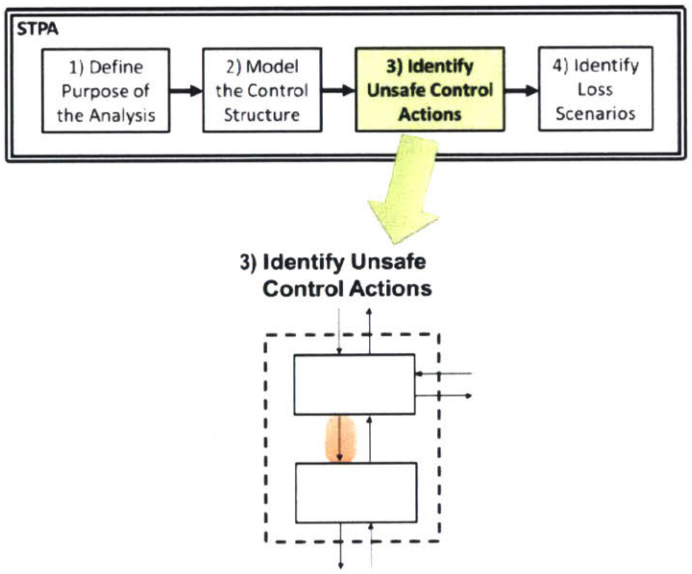
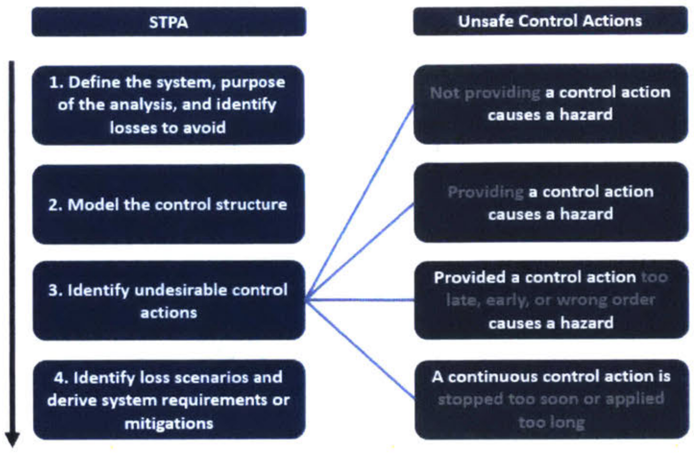

STPA
1) Define Purpose of the Analysis
2) Model the Control Structure
3) Identify Unsafe Control Actions
4) Identify Loss Scenarios
3) Identify Unsafe Control Actions

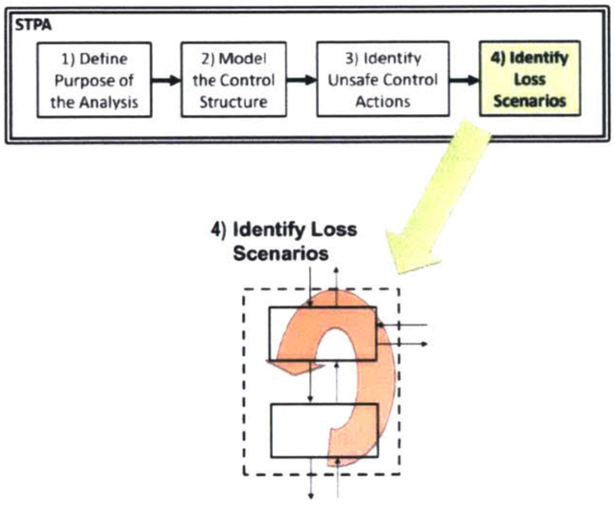
Figure 34. STPA Step 3: Identify Unsafe Control Actions [4]


STPA
Unsafe Control Actions
1. Define the system, purpose of the analysis, and identify losses to avoid
2. Model the control structure
3. Identify undesirable control actions
4. Identify loss scenarios and derive system requirements or mitigations
Not providing a control action causes a hazard
Providing a control action causes a hazard
Provided a control action too late, early, or wrong order causes a hazard
A continuous control action is stopped too soon or applied too long
Figure 35. Identifying Types of Unsafe Control Actions

5.1.4 Identify Loss Scenarios

Step 4 of STPA is identifying loss scenarios. This step includes the following sub-steps:

1. Identify reasons why unsafe control actions might occur.
    a) Incorrect feedback, inadequate requirements, design errors, component failures
    b) Safe control action provided but not followed or executed properly
2. Drive refined architecture, requirements, mitigations, and/or redesign.
3. Define appropriate test cases and drive test plans.


STPA
1) Define Purpose of the Analysis
2) Model the Control Structure
3) Identify Unsafe Control Actions
4) Identify Loss Scenarios
4) Identify Loss Scenarios
Figure 36. STPA Step 4: Identify Loss Scenarios [4]

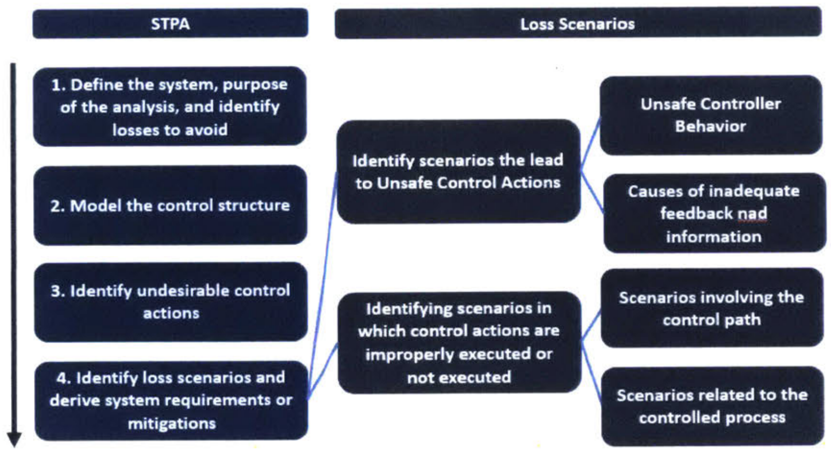

STPA
Loss Scenarios
1. Define the system, purpose of the analysis, and identify losses to avoid
2. Model the control structure
3. Identify undesirable control actions
4. Identify loss scenarios and derive system requirements or mitigations
Identify scenarios the lead to Unsafe Control Actions
Identifying scenarios in which control actions are improperly executed or not executed
Unsafe Controller Behavior
Causes of inadequate feedback nad information
Scenarios involving the control path
Scenarios related to the controlled process

Figure 37. Identifying Types of Loss Scenarios

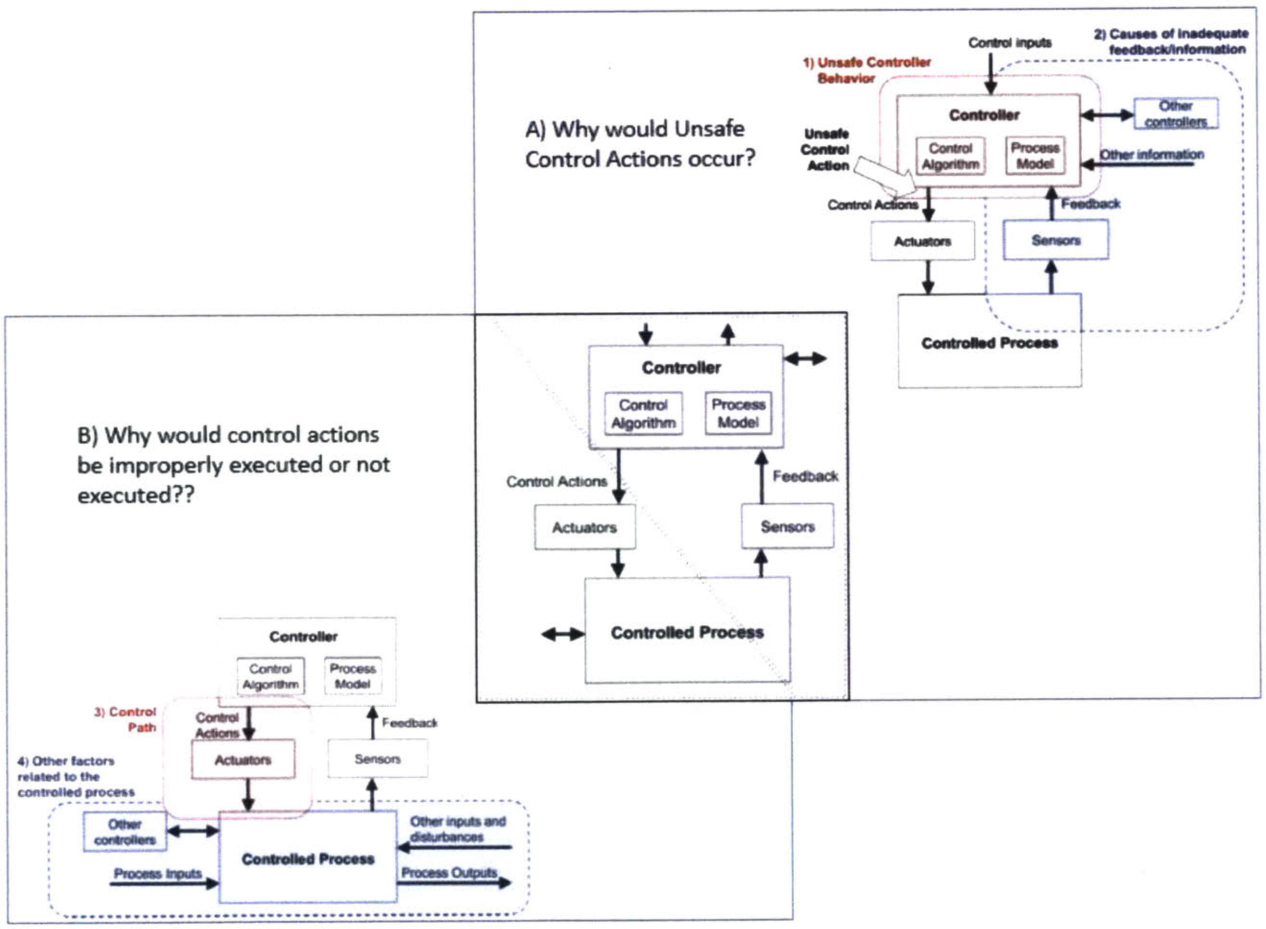


Figure 38. Systems-based Analysis of Unsafe Control Actions

5.2 STPA Implementation and Analysis Strategy

Implementation and analysis occurs in three phases. Phase 1 consists of gathering information on the rocket test controller's concept of operations, design requirements, interface control documents, detailed design drawings, test requirements, hazard reports, and the failure modes and effects analysis. Phase 2 is to implement STPA in a case study involving a particular use case of the rocket's "ascent phase." This particular use case was chosen because it involves the most significant use of the test controller, test article, and test environment for the captive hot-fire test of the rocket's four main engines and core stage. Phase 3 involves comparison of the STPA results to the FMEA results. This includes a comparison of the time required to perform each respective analysis. Finally, the results and analysis will be used to discuss the benefits, challenges, forward work, and implementation plan for incorporating STPA in the system engineering process.

This page has been intentionally left blank.

Chapter 6

6 Case Study of STPA Application to SLS Rocket Test Controller

Chapter 6 covers a case study of applying Systems-Theoretic Process Analysis to a particular use case of the SLS rocket test controller. The steps of STPA are applied to the test controller for the "ascent phase" use case throughout sections 6.1 through 6.6. Finally, section 6.7 details the results and analyzes STPA in comparison to FMEA.

6.1 Purpose, Environment, and Boundary Definition

Step 1 is to define the purpose, environment, and boundary of the analysis. The purpose of the analysis, the boundary, and the environment of the rocket test controller during the "ascent phase" use case are first defined as:

Purpose: Mitigate loss of life or injury to people, loss of or damage to the test article (rocket stage), loss of or damage to objects outside of the test article (infrastructure), and loss of the mission (test failure).

Boundary: The system includes all of the rocket test controller end-item components and the operator workstations.

Environment: The applicable surrounding systems include the test article, test infrastructure, associated test operators, and immediate vicinity.

6.2 Identifying Losses, Hazards, and Constraints

The second part of step one is then to define particular losses as derived from the purpose of the analysis and then branch those into associated system hazards and system-level constraints. Four particular losses are defined from the purpose as shown in Table 9.Table 9. STPA Case Study: Identification of System-level Losses.

Table 9. STPA Case Study: Identification of System-level Losses.

Losses	Description
L1	Loss of life or injury to people (test operators, technicians)
L2	Loss of or damage to the test article (rocket core stage)
L3	Loss of or damage to objects outside of the test article (test stand and infrastructure)
L4	Loss of mission (incomplete, failed, or aborted test)

The losses are then used to identify the various hazards that would lead to such losses and identifies the associated losses that correspond to each hazard as shown below in Table 10.

Table 10. STPA Case Study: Identification of System-level Hazards.

Hazards	Description	Associated Losses
H1	Controller exposes people (operators) to adverse test emissions, heat, pressure, vibration, or acoustics	L1, L4
H2	Controller allows test article component integrity to be lost	L2, L4
H3	Controller allows test support infrastructure component integrity to be lost	L3, L4
H4	Controller allows test article or supporting infrastructure (test stand controls) to become uncontrolled	L2, L3
H5	Controller causes test abort condition	L4
H6	Controller does not properly capture test data	L4

The hazards are then used to derive the system-level constraints for the associated hazards as shown in Table 11 below.

Table 11. STPA Case Study: Derivation of System-level Constraints from Hazards

System-level Constraints	Description	Associated Hazards
SC1	Controller must not allow people to be exposed to adverse test emissions, heat, pressure, vibration, or acoustics	H1, H5
SC2	Controller must not allow test article component loss of structural integrity	H2, H5
SC3	Controller must not allow test support infrastructure component loss of structural integrity	H3, H5
SC4	Controller must maintain control of the test article and supporting infrastructure	H4, H5
SC5	Controller must not cause a test abort condition	H5
SC6	Controller must properly capture all test data	H6
SC7	If the controller detects a violation of constraints then the controller must safely abort the test	H1, H2, H3, H4

6.3 Control Structure Modelling

Step 2 of STPA is to model the control structure of the rocket test controller. This process is designed to be iterative. Figure 39. STPA Case Study: High-level Environment Feedback Control Structure Model shows the high-level environment and analysis boundary which is represented by a dashed line. The arrows going down represent control actions and the arrows going up represent feedback.

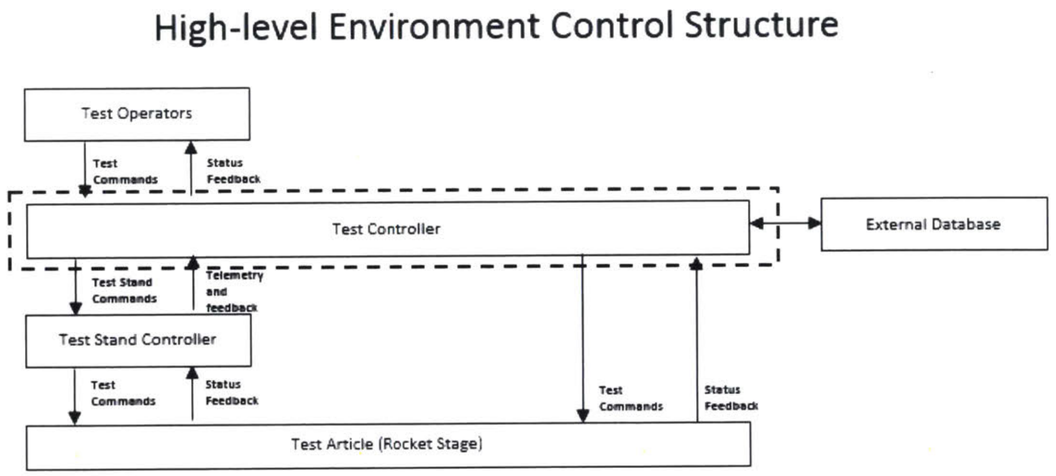


Figure 39. STPA Case Study: High-level Environment Feedback Control Structure Model

Upon iteration, a more-detailed low-level control structure is modeled as shown in Figure 40 below. The physical boundaries are represented by black dashed lines. The analysis boundary of the rocket test controller is represented by the blue dashed line. The "highest" controller within the analysis boundary is the software component of the rocket test controller that interacts with the test operator(s), called user applications and displays (UADs). UADs are part of the test control workstation(s) which represent the physical hardware component that test operator(s) interact with to control the rocket test. The "lowest" level within the analysis boundary is between the power supply and analog signal rack(s), the data processing rack, and the command and control (C2) rack which are generally the controllers within the boundary that interact with the test article and test stand/environment. The general logic flow results in actions given from top to bottom and feedback is received from bottom to top.

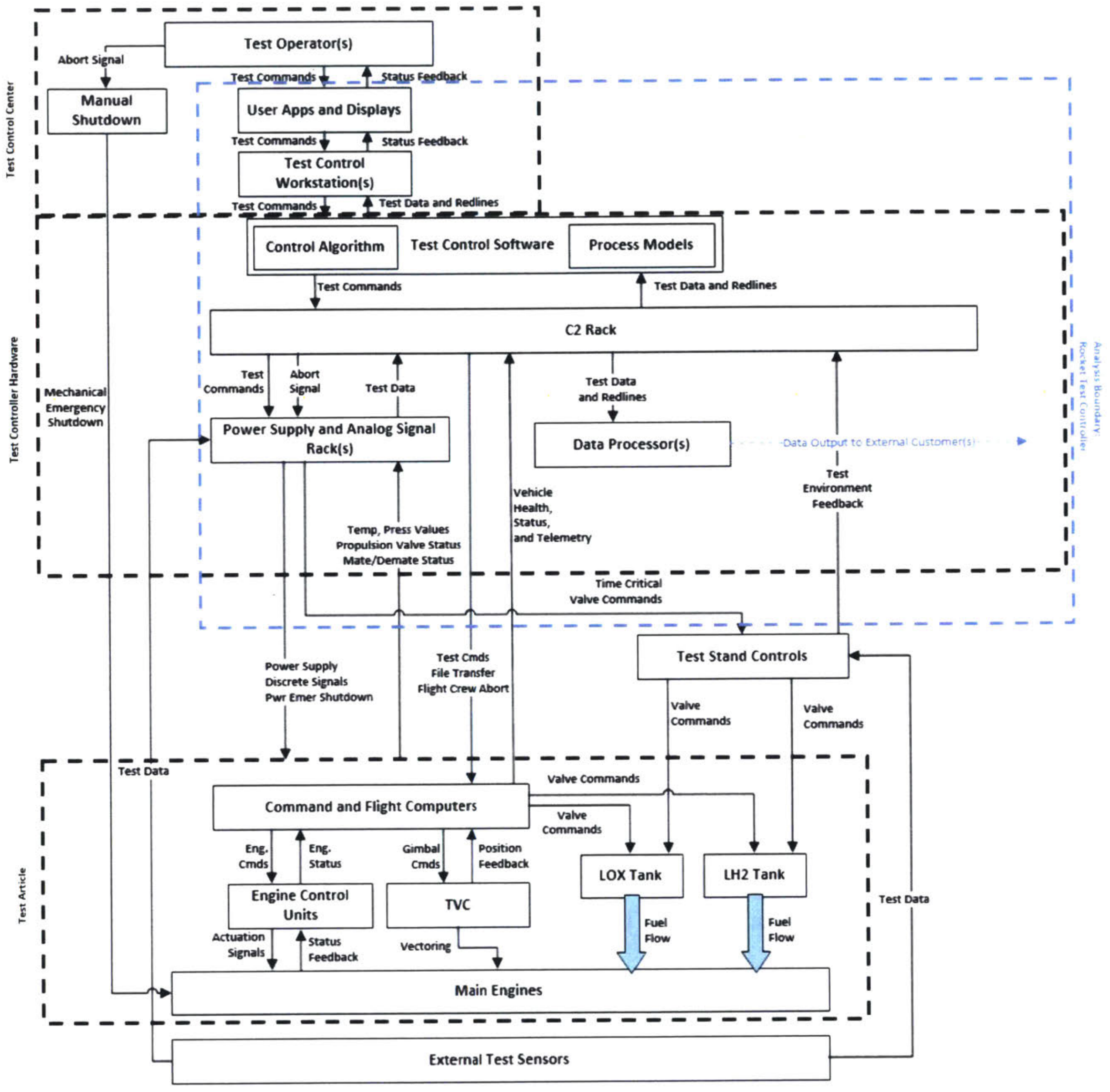


Figure 40. STPA Case Study: Low-level Feedback Control Structure Model

6.4 Identifying Unsafe Control Actions

Step 3 of STPA is to identify unsafe control actions. First, the intended control actions required by the rocket test controller to during the "ascent phase" must be listed. Eight control actions are identified, as listed in Table 12 below.

Table 12. STPA Case Study: "Ascent Phase" Control Actions for a Rocket Test Controller

Control Actions
1. Engine Start Monitoring
2. Engine Throttle Profile
3. Systems Monitoring and Responses
4. Thrust Vector Control Actuation Profile
5. Engine Cutoff Monitoring
6. Safing
7. Recycle
8. Advance to Shutdown Command Sequencing

Next each control action is analyzed using the following logic in conjunction with the feedback control structure from Figure 40 to determine whether an action (or no action) could lead to a hazard:

1. Not providing the control action leads to a hazard
2. Providing the control action leads to a hazard
3. Providing a potentially safe control action too late, too early, or in the wrong order
4. A continuous control action is stopped too soon or applied too long

Unsafe control actions are listed and labeled "UCA #" and each is linked to a higher order hazard from step 1, for traceability. The unsafe control actions represent area for design consideration that will be examined further in step 4. The unsafe control actions for the case study are shown in

Table 13 below.

Table 13. STPA Case Study: Unsafe Control Actions Table.

Control Action	Not providing control causes hazard	Providing control causes hazard	Control provided too early, too late, or out of sequence	Control stopped too soon or applied too long
1. Engine Start Monitoring	UCA 1.1: Engine start monitoring not initiated when commanded during automatic launch sequence [H2, H3, H4, H5]	N/A	UCA 1.2: Engines are started when parameters are not at nominal levels [H2, H4, H5] UCA 1.3: Engines are started before data acquisition is ready [H6]	N/A
2. Engine Throttle Profile	UCA 2.1: Throttling profile is not initiated [H5]	N/A	UCA 2.2: Throttling profile is initiated too early in the test [H5] UCA 2.3: Throttling profile is too late in the test [H5]	N/A
3. Systems Monitoring and Responses	UCA 3.1: Off-nominal thresholds and nominal ranges are not transmitted [H2, H3, H5]	UCA 3.2: Wrong off-nominal thresholds, nominal ranges, or condition sets are selected or incorrectly interpreted [H2, H3, H4, H5] UCA 3.3: Corrupted off-nominal thresholds or nominal ranges are transmitted [H2, H3, H4 H5]	UCA 3.4: Off-nominal thresholds and nominal ranges are set too early, causing a premature abort [H5] UCA 3.5: Off-nominal thresholds and nominal ranges are set too late, resulting in parameters exceeding thresholds with no abort [H2, H3, H4]	N/A
4. Thrust Vector Control Actuation Profile	UCA 4.1: Gimbaling profile is not initiated [H5]	N/A	UCA 4.2: Gimbaling profile is initiated too early in the test [H5] UCA 4.3 Gimbaling profile is initiated too late in the test [H5]	N/A

Control Action	Not providing control causes hazard	Providing control causes hazard	Control provided too early, too late, or out of sequence	Control stopped too soon or applied too long
5. Engine Cutoff Monitoring	UCA 5.1: Engine cutoff monitoring not provided when commanded during main engine cutoff sequence [H3, H6]	UCA 5.2: Engine cutoff monitoring provides false readings and aborts test [H5]	N/A	N/A
6. Safing	UCA 6.1: Safing profile not initiated after the hot fire test [H3, H5] UCA 6.2: Safing not provided while workers are present in the test area [H1]	N/A	UCA 6.3: Safing profile is initiated too early in the test causing an abort [H5] UCA 6.4 Safing profile is initiated too late in the test [H3, H5]	N/A
7. Recycle	UCA 7.1: LOX and LH2 draining is not commenced when required by test profile resulting in unstable configuration [H2, H3, H4]	N/A	UCA 7.2: LOX and LH2 draining is commenced when other systems are not ready [H2, H3]	N/A
8. Advance to Shutdown Command Sequencing	UCA 8.1: Advance to shutdown command sequencing is not initiated when off-nominal redline values are reached or during loss of vehicle control [H2, H3, H5]	UCA 8.2: Advance to shutdown procedure initiated while nominal values are still maintained [H5, H6]	UCA 8.3: Advance to shutdown procedure is provided too late in preempting loss of vehicle or after an off-nominal redline condition is reached [H2, H3]	N/A

The last part of step 3, is to determine controller constraints from the unsafe control actions.

These are shown in Table 14 below.

Table 14. STPA Case Study: Controller Constraints Identified for each Unsafe Control Action.

Unsafe Control Action	Controller Constraints
UCA 1.1: Engine start monitoring not initiated when commanded during automatic launch sequence [H2, H3, H4, H5]	C1.1: Controller must monitoring engine start when commanded during automatic launch sequence [UCA 1.1]
UCA 1.2: Engines are started when parameters are not at nominal levels [H2, H4, H5]	C1.2: Engines must not be started when parameters are off-nominal [UCA 1.3]
UCA 1.3: Engines are started before data acquisition is ready [H6]	C1.3: Engines must not be started before data acquisition is ready [UCA 1.4]
UCA 2.1: Throttling profile is not initiated [H5]	C2.1: Throttling profile must be initiated when commanded [UCA 2.1]
UCA 2.2: Throttling profile is initiated too early in the test [H5]	C2.2: Throttling profile must initiated at the correct moment in the test [UCA 2.2, UCA 2.3]
UCA 2.3: Throttling profile is too late in the test [H5]	C2.2: Throttling profile must initiated at the correct moment in the test [UCA 2.2, UCA 2.3]
UCA 3.1: Off-nominal thresholds and nominal ranges are not transmitted [H2, H3, H5]	C3.1: Off-nominal thresholds and nominal ranges must be transmitted [UCA 3.1]
UCA 3.2: Wrong off-nominal thresholds, nominal ranges, or condition sets are selected or incorrectly interpreted [H2, H3, H4, H5]	C3.2: Correct off-nominal thresholds, nominal ranges, or condition sets must be selected and correctly interpreted [UCA 3.2]
UCA 3.3: Corrupted off-nominal thresholds or nominal ranges are transmitted [H2, H3, H4 H5]	C3.3: Off-nominal thresholds or nominal ranges must be uncorrupted [UCA 3.3]
UCA 3.4: Off-nominal thresholds and nominal ranges are set too early, causing a premature abort [H5]	C3.4: Off-nominal thresholds and nominal ranges must not be set too early [UCA 3.4]
UCA 3.5: Off-nominal thresholds and nominal ranges are set too late, resulting in parameters exceeding thresholds with no abort [H2, H3, H4]	C3.5: Off-nominal thresholds and nominal ranges must not be set too late [UCA 3.5]
UCA 4.1: Gimbaling profile is not initiated [H5]	C4.1: Gimbaling profile must be initiated when commanded [UCA 4.1]
UCA 4.2: Gimbaling profile is initiated too early in the test [H5]	C4.2: Gimbaling profile must be initiated on time during the test [UCA 4.2, UCA 4.3]
UCA 4.3 Gimbaling profile is initiated too late in the test [H5]	C4.2: Gimbaling profile must be initiated on time during the test [UCA 4.2, UCA 4.3]
UCA 5.1: Engine cutoff monitoring not provided when commanded during main engine cutoff sequence [H3, H6]	C5.1: Engine cutoff monitoring must be provided when commanded during main engine cutoff sequence [UCA 5.1]
UCA 5.2: Engine cutoff monitoring provides false readings and aborts test [H5]	C5.2: Engine cutoff monitoring must not provide false readings [UCA 5.2]

Unsafe Control Action	Controller Constraints
UCA 6.1: Safing profile not initiated after the hot fire test [H3, H5]	C6.1: Safing profile must be initiated after the hot fire test [UCA 6.1]
UCA 6.2: Safing not provided while workers are present in the test area [H1]	C6.2: Safing must be provided while workers are present in the test area [H1]
UCA 6.3: Safing profile is initiated too early in the test causing an abort[H5]	C6.3: Safing profile must be initiated on time during the test [UCA 6.2, UCA 6.3]
UCA 6.4 Safing profile is initiated too late in the test [H3, H5]	C6.3: Safing profile must be initiated on time during the test [UCA 6.2, UCA 6.3]* Redundant Control
UCA 7.1: LOX and LH2 draining is not commenced when required by test profile resulting in unstable configuration [H2, H3, H4]	C7.1: LOX and LH2 draining must be commenced when required by test profile [UCA 7.1]
UCA 7.2: LOX and LH2 draining is commenced when other systems are not ready [H2, H3]	C7.2: LOX and LH2 draining must not be commenced when other systems are not ready [UCA 7.2]
UCA 8.1: Advance to shutdown command sequencing is not initiated when off-nominal redline values are reached or during loss of vehicle control [H2, H3, H5]	C8.1: Advance to shutdown command sequencing must be initiated when off-nominal redline values are reached or during loss of vehicle control [UCA 8.1]
UCA 8.2: Advance to shutdown procedure initiated while nominal values are still maintained [H5, H6]	C8.2: Advance to shutdown procedure must not be initiated while nominal values are still maintained [UCA 8.2]
UCA 8.3: Advance to shutdown procedure is provided too late in preempting loss of vehicle or after an off-nominal redline condition is reached [H2, H3]	C8.3: Advance to shutdown procedure must be provided in time to avoid loss of vehicle and after an off-nominal redline condition is reached [UCA 8.3]

6.5 Identifying Loss Scenarios

STPA step 4 is the final step in the analysis cycle before iterating back at step 1. This step begins with identifying loss scenarios that would cause the unsafe control actions to occur and identifies scenarios in which control actions are improperly executed or not executed that may also cause an unsafe control action to occur. The results of the case study are shown in Table 15 below.

Table 15. STPA Case Study: Determining Causes of Losses for each Unsafe Control Action

Unsafe Control Action	Loss Causal Scenarios	Mitigations
UCA 1.1: Engine start monitoring not initiated when commanded during automatic launch sequence [H2, H3, H4, H5]	LCS 1.1.1 Engine start monitoring is not initiated when commanded during automatic launch sequence because loss of signal to test article from improper connection. LCS 1.1.2 Engine start monitoring is not initiated when commanded during automatic launch sequence because C2 rack power interruption due to test article induced vibration or atmospherics.	Hardware quality assurance and testing. C2 Rack power hardening, condition monitoring, and automatic backup source
UCA 1.2: Engines are started before data acquisition is ready [H6]	LCS 1.2.1 Engines are started before data acquisition is ready because the test sequence is incorrectly programmed LCS 1.2.2 Engines are started before data acquisition is ready because vehicle health and status monitoring is incorrectly monitored in test control software process model	Software quality assurance and testing.
UCA 2.1: Throttling profile is not initiated [H5]	LCS 2.1.1 The throttling profile is not initiated because a hardware connector fails in the C2 rack LCS 2.1.2 The throttling profile is not initiated because the test sequence is incorrectly programmed	Hardware quality assurance and testing. Software quality assurance and testing. Redundant C2 rack capability and hard switchover in no throttle profile situation
UCA 2.2: Throttling profile is initiated too early in the test [H5]	LCS 2.2.1 Throttling profile is initiated too early in the test because the test sequence is incorrectly programmed	Software quality assurance and testing.

Unsafe Control Action	Loss Causal Scenarios	Mitigations
UCA 2.3: Throttling profile is too late in the test [H5]	LCS 2.3.1 Throttling profile is initiated too late in the test because the test sequence is incorrectly programmed. LCS 2.3.2 The throttling profile is not initiated because of IRIG timing mismatch between C2 rack and test article	Software quality assurance and testing. Redundant C2 rack capability and hard switchover in late throttle profile situation
UCA 1.1: Engine start monitoring not initiated when commanded during automatic launch sequence [H2, H3, H4, H5]	LCS 1.1.1 Engine start monitoring is not initiated when commanded during automatic launch sequence because loss of signal to test article from improper connection. LCS 1.1.2 Engine start monitoring is not initiated when commanded during automatic launch sequence because C2 rack power interruption due to test article induced vibration or atmospherics.	Hardware quality assurance and testing. C2 Rack power hardening, condition monitoring, and automatic backup source
UCA 1.2: Engines are started before data acquisition is ready [H6]	LCS 1.2.1 Engines are started before data acquisition is ready because the test sequence is incorrectly programmed LCS 1.2.2 Engines are started before data acquisition is ready because vehicle health and status monitoring is incorrectly monitored in test control software process model	Software quality assurance and testing.
UCA 3.1: Off-nominal thresholds and nominal ranges are not transmitted [H2, H3, H5]	LCS 3.1.1 Off-nominal thresholds and nominal ranges are not transmitted because of a hardware connection was improperly made LCS 3.1.1 Off-nominal thresholds and nominal ranges are not transmitted because of a hardware connection was not installed	Hardware quality assurance and testing. Independent verification of successful data transfer to test article
UCA 3.2: Wrong off-nominal thresholds, nominal ranges, or condition sets are selected or incorrectly interpreted [H2, H3, H4, H5]	LCS 3.2.1 Wrong off-nominal thresholds, nominal ranges, or condition sets are selected or incorrectly interpreted because test configuration control improperly programmed and verified LCS 3.2.2 Wrong off-nominal thresholds, nominal ranges, or condition sets are selected or incorrectly interpreted because process model incorrectly programmed	Require test configuration verification Software quality assurance and testing.
UCA 3.3: Corrupted off-nominal thresholds or nominal ranges are transmitted [H2, H3, H4 H5]	LCS 3.3.1 Corrupted off-nominal thresholds or nominal ranges are transmitted because data transfer and processing performance saturation LCS 3.3.2 Corrupted off-nominal thresholds or nominal ranges are transmitted because hardware reading, writing, and memory failures causing loss of data integrity	Software quality assurance and testing. Hardware quality assurance and testing. Require independent verification of data integrity from end-to-end
UCA 3.4: Off-nominal thresholds and nominal ranges are set too early, causing a premature abort [H5]	LCS 3.4.1 Off-nominal thresholds and nominal ranges are set too early, causing a premature abort because the test sequence is incorrectly programmed	Software quality assurance and testing.

Unsafe Control Action	Loss Causal Scenarios	Mitigations
UCA 3.5: Off-nominal thresholds and nominal ranges are set too late, resulting in parameters exceeding thresholds with no abort [H2, H3, H4]	LCS 3.5.1 Off-nominal thresholds and nominal ranges are set too late, resulting in parameters exceeding thresholds with no abort because the test sequence is incorrectly programmed. LCS 3.5.2 Off-nominal thresholds and nominal ranges are set too late, resulting in parameters exceeding thresholds with no abort because of incorrect software process model LCS 3.5.3 Off-nominal thresholds and nominal ranges are set too late, resulting in parameters exceeding thresholds with no abort because of insufficient software control algorithm LCS 3.5.4 Off-nominal thresholds and nominal ranges are set too late, resulting in parameters exceeding thresholds with no abort because of stale data or data latency in transfer LCS 3.5.5 Off-nominal thresholds and nominal ranges are set too late, resulting in parameters exceeding thresholds with no abort because of IRIG timing mismatch between C2 rack and test article	Software quality assurance and testing. Independent verification of successful data transfer to test article Require independent timing validation and C2 rack condition monitoring
UCA 4.1: Gimbaling profile is not initiated [H5]	LCS 4.1.1 Gimbaling profile is not initiated because of an insufficient software control algorithm LCS 4.1.2 Gimbaling profile is not initiated because of a loss of connection to test article from C2 rack.	Software quality assurance and testing. Hardware quality assurance and testing. Redundant C2 rack capability and switchover
UCA 4.2: Gimbaling profile is initiated too early in the test [H5]	LCS 4.2.1 Gimbaling profile is initiated too early in the test because the test sequence is incorrectly programmed.	Software quality assurance and testing.
UCA 4.3 Gimbaling profile is initiated too late in the test [H5]	LCS 4.3.1 Gimbaling profile is initiated too late in the test because the test sequence is incorrectly programmed. LCS 4.3.2 Gimbaling profile is initiated too late in the test because insufficient software control algorithm	Software quality assurance and testing.
UCA 5.1: Engine cutoff monitoring not provided when commanded during main engine cutoff sequence [H3, H6]	LCS 5.1.1 Engine cutoff monitoring not provided when commanded during main engine cutoff sequence because of loss of signal to test article from improper connection.	Hardware quality assurance and testing. Redundant C2 rack capability and hard switchover in loss of communication situation Require independent manual shutoff capability in test control center

Unsafe Control Action	Loss Causal Scenarios	Mitigations
UCA 5.2: Engine cutoff monitoring provides false readings and aborts test [H5]	LCS 5.2.1 Engine cutoff monitoring provides false readings and aborts test because of incorrect software process model. LCS 5.2.2 Engine cutoff monitoring provides false readings and aborts test because of stale data or data latency in transfer.	Software quality assurance and testing. Hardware quality assurance and testing.
UCA 6.1: Safing profile not initiated after the hot fire test [H3, H5]	LCS 6.1.1 Safing profile not initiated after the hot fire test because the test sequence is incorrectly programmed. LCS 6.1.2 Safing profile not initiated after the hot fire test because a hardware connector fails in the C2 rack.	Software quality assurance and testing. Hardware quality assurance and testing.
UCA 6.2: Safing not provided while workers are present in the test area [H1]	LCS 6.2.1 Safing not provided while workers are present in the test area because of an incorrect software process model LCS 6.2.2 Safing not provided while workers are present in the test area because parameters are not at nominal levels due to insufficient software control algorithm LCS 6.2.1 Safing not provided while workers are present in the test area because of stale data or data latency provides false nominal indication	Software quality assurance and testing. Redundant C2 rack capability and hard switchover in stale data situation
UCA 6.3: Safing profile is initiated too early in the test causing an abort [H5]	LCS 6.3.1 Safing profile is initiated too early in the test causing an abort because the test sequence is incorrectly programmed.	Software quality assurance and testing.
UCA 6.4 Safing profile is initiated too late in the test [H3, H5]	LCS 6.4.1 Safing profile is initiated too late in the test causing an abort because the test sequence is incorrectly programmed. LCS 6.4.2 Safing profile is initiated too late in the test causing an abort because of insufficient software control algorithm. LCS 6.4.3 Safing profile is initiated too late in the test causing an abort because of stale data or data latency in transfer	Software quality assurance and testing. Redundant C2 rack capability and hard switchover in stale data situation Require independent manual safing capability in test control center
UCA 7.1: LOX and LH2 draining is not commenced when required by test profile resulting in unstable configuration [H2, H3, H4]	LCS 7.1.1 LOX and LH2 draining is not commenced when required by test profile resulting in unstable configuration because loss of communication with test article from connector not installed correctly LCS 7.1.2 LOX and LH2 draining is not commenced when required by test profile resulting in unstable configuration because insufficient software control algorithm.	Require independent manual draining command capability in test control center Software quality assurance and testing with actual test stand interface.
UCA 7.2: LOX and LH2 draining is commenced when other systems are not ready [H2, H3]	LCS 7.2.1 LOX and LH2 draining is commenced when other systems are not ready because the test sequence is incorrectly programmed. LCS 7.2.1 LOX and LH2 draining is commenced	Software quality assurance and testing.

Unsafe Control Action	Loss Causal Scenarios	Mitigations
	when other systems are not ready because of insufficient test control software process model	
UCA 8.1: Advance to shutdown command sequencing is not initiated when off-nominal redline values are reached or during loss of vehicle control [H2, H3, H5]	LCS 8.1.1 Advance to shutdown command sequencing is not initiated when off-nominal redline values are reached or during loss of vehicle control because of insufficient software process model LCS 8.1.2 Advance to shutdown command sequencing is not initiated when off-nominal redline values are reached or during loss of vehicle control because of insufficient software control algorithm LCS 8.1.3 Advance to shutdown command sequencing is not initiated when off-nominal redline values are reached or during loss of vehicle control because of hardware communication improperly connected to test article LCS 8.1.4 Advance to shutdown command sequencing is not initiated when off-nominal redline values are reached or during loss of vehicle control because of processing performance saturation prevents timely recognition of off-nominal redline condition LCS 8.1.5 Advance to shutdown command sequencing is not initiated when off-nominal redline values are reached or during loss of vehicle control because user apps and displays do not present redline situation adequately to warn test operator(s).	Redundant C2 rack capability and hard switchover in loss of communication situation Require independent manual shutoff capability in test control center Software quality assurance and testing. Hardware quality assurance and testing. User application and display standards for off-nominal redline monitoring and displays. Test operator training on off-nominal redlines recognition and applicable procedures.
UCA 8.2: Advance to shutdown procedure initiated while nominal values are still maintained [H5, H6]	LCS 8.2.1 Advance to shutdown procedure initiated while nominal values are still maintained because of insufficient software process model LCS 8.2.2 Advance to shutdown procedure initiated while nominal values are still maintained because of insufficient software control algorithm LCS 8.2.3 Advance to shutdown procedure initiated while nominal values are still maintained because user apps and displays do not present redline situation adequately to warn test operator(s).	Software quality assurance and testing. User application and display standards for off-nominal redline monitoring and displays. Test operator training on off-nominal redlines recognition and applicable procedures.

Unsafe Control Action	Loss Causal Scenarios	Mitigations
UCA 8.3: Advance to shutdown procedure is provided too late in preempting loss of vehicle or after an off-nominal redline condition is reached [H2, H3]	LCS 8.3.1 Advance to shutdown procedure is provided too late in preempting loss of vehicle or after an off-nominal redline condition is reached because hardware communication improperly connected to test article	Redundant C2 rack capability and hard switchover in loss of communication or stale data situation
	LCS 8.3.2 Advance to shutdown procedure is provided too late in preempting loss of vehicle or after an off-nominal redline condition is reached because of processing performance saturation prevents timely recognition of off-nominal redline condition	Require independent manual shutoff capability in test control center
	LCS 8.3.3 Advance to shutdown procedure is provided too late in preempting loss of vehicle or after an off-nominal redline condition is reached because user apps and displays do not provide adequate reaction time test operator(s).	Independent verification of monitored conditions and responses that require automated advance to shut down if test operator reaction time is less than TBD seconds required.

6.6 Deriving System Requirements and Mitigations

The causes of the losses in step 4 were used to determine mitigations to offset such scenarios from occurring. These include design considerations (such as hardening or redundancy), software programming considerations, procedural considerations (such as limiting how the system is used by the operators), and quality assurance checks. Combined with the system-level constraints from step 1 and the controller constraints from step 3, it is possible to derive additional system requirements and mitigation strategies to obtain a safer system as defined in the purpose in step 1. If used in conjunction with the design process, early on in the systems engineering "V," STPA can be iterated to further refine the design with safety in mind from the beginning.

6.7 Analysis and Results

STPA for the "ascent phase" of the analysis resulted in 83% of the total work time needed to complete a comparable "ascent phase" analysis using FMEA analysis. The STPA results are the same or meet a similar intent to those resolved in the FMEA with not gaps between the two methods. The recommended mitigation and constraints resulting from STPA are arguably more intuitive than those of the FMEA. This is likely due to the methodology of STPA which employs logical systems-based reasoning to explore the system's emergent properties and all possible causes as opposed to FMEA's failed-component approach which focuses mainly on hardware component failures.

This page has been intentionally left blank.

Chapter 7

7 Conclusion and Recommendations for Forward Work

This chapter provides concluding thoughts and summarizes the work of this thesis. Section 7.1 discusses the benefits of using STPA while Section 7.2 covers the challenges. Further, section 7.2 includes recommendations for forward work. Finally, Section 7.3 covers specific methods to integrate STPA into a systems engineering process in a complex aerospace design environment such as the rocket testing controller for SLS.

7.1 Benefits

STPA offers several benefits over traditional hazard methods. It requires little understanding of the underlying theoretical and mathematical foundation to be useful. It is further shown to require less time and resources to employ and can be integrated into the entire systems engineering design process. It is easier to incorporate from the beginning of the design process because it uses a top-down methodology rather than bottom-up methodology like FMEA. In addition, modern complexity is leading to important system properties (such as safety) not being related to the behavior of individual system components but rather to the interactions among the components, which STPA takes into account. Accidents can occur due to unsafe interactions among components that have not failed and, in fact, still satisfy their requirements. Therefore the goal of STPA is a more thorough systems-based method that can identify emergent properties that may be missed by traditional methods.

Traditional hazard analyses such as Failure Modes and Effects Analysis (FMEA) and Failure Modes and Effects Criticality Analysis (FMECA) are based on system decomposition that involve dividing the system into components. They assume that accidents are caused by component failure, calculate the probability of failure of each component separately, and later combine the analysis results into a system reliability figure, which is assumed to be a measure of safety or risk. They often only consider failures that can lead to a critical loss rather than all failures. These models assume that all components in the system are able to be defined stochastically for the probabilities or likelihood to be determined. The probability of each component failing is combined into chains of directly related physical or functional failure events. Unfortunately, software and humans do not satisfy this assumption. The assumptions underlying the decomposition-based approach to traditional hazard analysis are relatively true for the type of electromechanical systems built in less complex designs and still hold for certain properties in newer high-technology, software-intensive systems. However, the greatly increased complexity in our systems today (which is made possible primarily by the use of software) is creating systems where the approach is no longer as effective. It is much more difficult today to anticipate, understand, plan, and guard against all potential system behavior before operational use of our systems. Complexity is creating "unknowns" that cannot be identified by breaking the system behavior into chains of events [4].

7.2 Challenges and Forward Work

Additional forward work to consider includes automating STPA processes for steps 3 and 4, integrating STPA into the systems engineering design process from beginning to end, using prediction analysis for scheduling decisions impact, and additional means of increasing requirements clarity using natural language processing techniques.

Automating steps 3 and 4 is possible for certain design aspects of a repetitive nature. Employing cloud computing and enhanced software functionality can reduce repetitive tasks and make the STPA process even quicker. Further, by integrating STPA into the engineering process from beginning to end, emergent properties can be found and discovered early in the process to avoid expensive redesign or safety issues found in PRAs and FMEAs that often occur too late in the process. Implementing STPA across such a large organization poses a significant change management challenge that should not be taken lightly. Further, it poses challenges to the working relationship and mutual understanding between a government agency with specific standards and the private company as the prime contractor to what are acceptable methods of hazard analysis.

Monthly measures may be used to predict man-hours and schedule effects in order to build an effective prediction system using techniques like linear regression analysis alongside of queuing optimization models to minimize the total cost of the combined waiting cost (cost of schedule delays) and the service cost (cost of employees) as shown in Figure 41 [9].

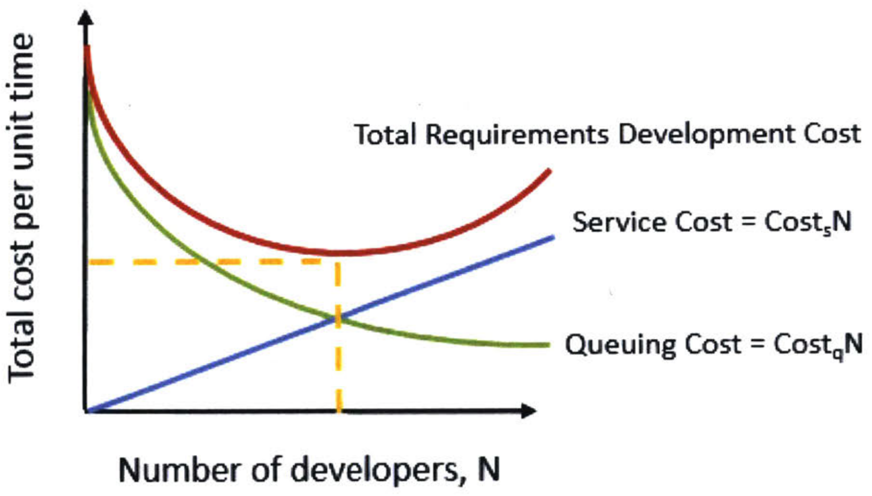


Figure 41. Using metrics to minimize schedule and cost impacts.

Additionally, the Lean Innovation Cycle offers the advantages of a prioritized task list that avoids making a big list of features and prevents scope-creep. The iterative nature allows a natural way to test if the program's assumptions were correct along the way.

Lastly, advances in natural language processing (NLP) may be incorporated into requirements databases to make sure requirements are written clearly and match overall program-wide system requirements to prevent inducing software defects based on inconsistent or potentially poorly-written requirements.

7.3 Integrating STPA into Your System Engineering Process

For larger organizations, major challenges, as mentioned earlier, involve changing the culture, changing the standard processes used, and training large numbers of people to use newer and different methods than they have in the past. Regarding training, experience shows that the most effective approach to introducing STPA is an interactive method with hands-on exercises used to reinforce the process. The organization should expect that employees with significant experience using traditional hazard analysis techniques may have the most difficulty learning or accepting STPA. For very large organizations in which thousands of people need to be trained, using a "train-the-trainer" approach may be desirable. The most effective way to produce STPA experts who can serve as future trainers and facilitators is to immerse candidates in real projects where STPA is actively used and even shadowing other facilitators.

STPA is the most useful when it is integrated into the design effort and not performed as an independent effort by a separate group. Therefore, for large projects, an STPA facilitator may be assigned as the expert whom will then break up the process into smaller components that can be done successfully by individuals or by small groups of experts. The facilitator may provide some initial STPA training at the start of the project, and should be able to answer any related questions that arise during the project. The facilitator can identify and schedule tasks that may be done in parallel by different groups. Eventually the facilitator can bring these components together into the overall analysis. Facilitators may also lead the meetings and manage the interactions among the team members. The facilitator generally reviews the results during the process and at the end of the process to ensure the method is being followed correctly and that no gaps occurred. At the conclusion of the project, the facilitator may compile the results into a final report and ensure the findings are provided to the appropriate functions as a means of providing traceability.

During the initial design process, the early control structure model can be used to help identify the areas of expertise that will be needed. Those with background or experience in testing are often excellent choices for an STPA team. Testers can be quite knowledgeable and able to find problems that may have been overlooked because their job involves constantly questioning the assumptions and claims that have been made. They rarely assume that requirements, specifications, or design decisions are completely correct, an excellent characteristic for STPA team members. Initial skepticism by those deeply involved in a project design is quickly overcome when they find UCAs or scenarios that had not been previously identified.

With regard to cost and return on investment, STPA is very efficient. Typically, the first use is most costly because of learning costs, but the cost quickly drops off as experience is gained. With regard to the time required, Figure 42 shows the relative amount of time spent by all STPA team members on a recent industry project. For most industry STPA projects, most of the time is spent learning how the system to be analyzed works (which would need to be done for any method that identifies hazard causes), and actually not spent learning STPA or performing STPA itself. Roughly a quarter of the time is spent finding answers to "what if" technical questions about the system that were not considered previously. A relatively small amount of time is typically spent actually learning STPA and following the process [4].

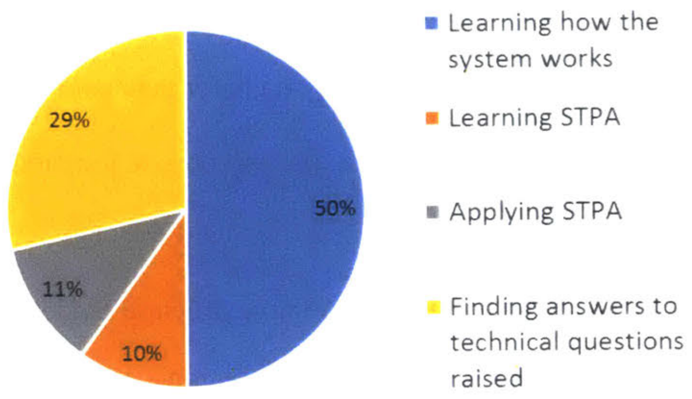


Figure 42. Relative amount of time spent on different tasks [4].

Most of the time is spent exploring real concerns in the system that can immediately be used to drive decisions and provide insight about how to make improvements. Organizations in past use cases reported that STPA requires about 2 or 3 orders of magnitude less time and resources than the more traditional hazard analysis techniques and produces more complete results [4].

List of Acronyms

APU	Auxiliary Power Unit
BOS	Boston's Logan International Airport
C	Controller Constraint
C2	Command and Control
CAST	Casual Analysis Using Systems Theory
CDR	Critical Design Review
ConOps	Concept of the Operation
CS	Core Stage
ECP	Engineering Change Process
ECU	Engine Control Unit
EM	Exploration Mission
EPS	Electrical Power Systems
ET	External Tank
ETA	Event Tree Analysis
EUS	Exploration Upper Stage
FHA	Fault Hazard Analysis
FIFO	First-In-First-Out
FMEA	Failure Mode and Effects Analysis
FMECA	Failure Modes and Effects Criticality Analysis
FTA	Fault Tree Analysis
GEV	Generalized Extreme Value
GUI	Graphical User Interface

HAZOP	Hazard and Operability Analysis
IPT	Integrated Product Team
IRR	Internal Requirements Revision
ISS	International Space Station
ITAR	International Traffic in Arms Regulations
JAL	Japan Airlines
KLOC	Thousand Lines Of Code
KSC	Kennedy Space Center
LCS	Loss Causal Scenario
LEO	Low Earth Orbit
LH2	Liquid Hydrogen
LO2	Liquid Oxygen
MAF	Michoud Assembly Facility
MLOC	Million Lines of Code
MPS	Main Propulsion System
MPTA	Main Propulsion Test Article
MSFC	Marshall Space Flight Center
NASA	National Aeronautics and Space Administration
NTSB	National Transportation Safety Board
OV	Orbiter Vehicle
PDR	Preliminary Design Review
PHA	Preliminary Hazard Analysis
PRA	Probabilistic Risk Assessment

RS-25D	Rocket System 25 (Space Shuttle Main Engine)
SC	Stage Controller
SLS	Space Launch System
SME	Subject Matter Expert
SpaceX	Space Exploration Technologies Corporation
SRB	Solid Rocket Boosters
SSC	Stennis Space Center
STPA	Systems-Theoretic Process Analysis
STAMP	Systems-Theoretic Accident Modeling and Processes
STE	Special Test Equipment
STS	Space Transportation System
S&MA	Safety & Mission Assurance
TAK	Takamatsu Airport
TCC	Test Commit Criteria
TPS	Thermal Protection System
TVC	Thrust Vector Control
UADs	User Apps and Displays
UCA	Unsafe Control Actions
WIP	Work in Progress

Bibliography

[1] B. Mumgaard, Z. Hartwig, B. Sorbom and D. Brunner, "MIT Plasma Science and Fusion Center SPARC Mission Charts," 14 January 2016. [Online]. Available: http://library.psfc.mit.edu/catalog/online_pubs/iap/iap2016/mumgaard.pdf. [Accessed 26 June 2018].

[2] T. L. Hamlin, E. Thigpen, J. Kahn and Y. Lo, "Shuttle Risk Progression: Use of the Shuttle Probabilistic Risk Assessment (PRA) to Show Reliability Growth," 2011. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20110004917.pdf. [Accessed 2 June 2018].

[3] National Aeronautics and Space Administration , "Space Launch System Fact Sheet," [Online]. Available: https://www.nasa.gov/sites/default/files/atoms/files/sls_fact_sheet_06122018.pdf. [Accessed 20 August 2018].

[4] N. G. Leveson and J. P. Thomas, "STPA Handbook," March 2018. [Online]. Available: http://psas.scripts.mit.edu/home/. [Accessed 1 April 2018].

[5] National Aeronautics and Space Administration, Systems Engineering Handbook, NASA/SP-2007-6105 Rev1, Washington, DC: NASA, 2007.

[6] The Boeing Company, "Boeing: Space Launch System," [Online]. Available: https://www.boeing.com/space/space-launch-system/. [Accessed 20 August 2018].

[7] L. Dean, "B-2 Test Stand on Schedule for Testing Space Launch System Core," NASA, 19 October 2016. [Online]. Available: https://www.nasa.gov/feature/b-2-test-stand-on-schedule-for-testing-space-launch-system-core. [Accessed 22 December 2018].

[8] National Chung Hsing University, Department of Mechanical Engineering, "Computer Integrated Manufacturing, Ch. 3 Concurrent Engineering," 10 March 2009. [Online]. Available: http://www.me.nchu.edu.tw/lab/CIM/www/courses/ Computer%20Integrated%20Manufacturing/ Chapter3%20-Concurrent%20Engineering.pdf. [Accessed 27 June 2018].

[9] A. Loconsole, "A Correlational Study on Four Measures of Requirements Volatility," Sweden, 2008.

[10] G. Stark, P. Oman, A. Skillcorn and R. Ameele, "An Examination of the Effects of Requirements Changes on Software Maintenance Releases," *Journal of Software Maintenance,* vol. 11, no. 5, pp. 239-309, 1999.

[11] A. Loconsole, "Definition and Validation of Requirements Management Measures," Department of Computing Science, Umeå, Sweden, 2008.

[12] C. Woody, R. Ellison and W. Nichols, "Predicting Software Assurance Using Quality and Reliability Measures," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2104.

[13] B. Olson and D. Swenson, "Overtime Effects on Project Team Effectiveness," in *Proceedings of the 44th Annual Midwest Instruction and Computing Symposium (MICS),* Duluth, MN, 2011.

[14] J. Little, S. Graves, D. (eds.) Chhajed and T. Lowe, "Chapter 5 Little's Law," in *Building Intuition: Insights From Basic Operations Management Models and Principles*, Springer Science+ Business Media, LLC, 2008, pp. 81-100.

[15] N. G. Leveson, Engineering a Safer World, Cambridge, MA: MIT Press, 2012.

[16] JPL Special Review Board, "Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions," 22 March 200. [Online]. Available: https://spaceflight.nasa.gov/spacenews/releases/2000/mpl/mpl_report_1.pdf. [Accessed 21 August 2018].

[17] National Transportation Safety Board (NTSB), "Auxiliary Power Unit Battery Fire Japan Airlines Boeing 787-8, JA829J," [Online]. Available: https://www.ntsb.gov/investigations/AccidentReports/Pages/AIR1401.aspx. [Accessed 21 August 2018].

[18] A. Jenkinson, "The frequency distribution of the annual maximum (or minimum) values of meteorological elements.," *Quarterly Journal of the Royal Meteorological Society,* no. 87, pp. 158-171, 1955.