Application of MBSE to Risk-Informed Design Methods for Space Mission Applications

Rafael M. Perez

Master's Project

SYS 800: Special Problems in System Engineering

Fall 2013

Advisor: Dr. Jerry Jon Sellers

DEDICATION

I dedicate this research to God and my lovely wife, Yenie, for their unconditional love and support in my life and throughout my study.

ACKNOWLEDGEMENTS

I wish to express my deep gratitude to Dr. Jerry Jon Sellers, my research advisor, for his patient guidance, constant encouragement, and constructive critiques during the planning and conduct of this research. I wish to thank various people for their contribution to this project; Dr. Steven Dam, Mr. Robert Sperlazza and Mr. Chris Ritter, staff of SPEC Innovations, for their valuable technical support and recommendations in developing the Innoslate model; Dr. John Connolly, Dr. Robert Bayt, and Mr. Lee Graham, staff of NASA, for providing reference material on RID and the Altair Project; Mr. Randolph Rust, staff of NASA, for his valuable assistance in providing data from the Altair Project. I wish to offer special thanks to Mr. Scott Fisher, staff of Exelis Inc. and my department manager, for his support throughout my study. I would like to extend my thanks to Dr. Peter McQuade, Program Director, for supporting the Space Systems Engineering program at Stevens Institute of Technology and the approval of Dr. Sellers as my advisor. I would also like to extend my thanks to Mr. Vincent Lauria and Ms. Marian Plummer, staff of Exelis Inc., for their encouragement and recommendation to attend the program. Finally, I wish to thank my lovely wife, Yenie, for her incredible support and encouragement throughout my study.

ABSTRACT

This paper describes research into the application of Model-Based Systems Engineering (MBSE) tools and processes to Risk-Informed Design (RID). RID enables system risk analyses early in the lifecycle of spaceflight projects allowing designers to use risk as a design commodity and part of the overall trade space. RID uses a "minimum functionality" approach, whereby a minimal, single-string system design is first envisioned that only meets basic performance requirements without any regard to overall reliability or safety. Risk analyses are then used to apply informed design enhancements based on their contribution to risk reduction. A recent application of RID was the Altair Lunar Lander Project that was intended for human lunar exploration under NASA's Constellation Program. The Altair project's approach and results are reviewed and analyzed in this paper as a specific application of RID. In traditional projects, several tools such as Relex, Windchill or SAPHIRE, are used in parallel to apply risk informed design techniques. These analyses also traditionally occur later in the design cycle when changes are more difficult to implement. Safety and reliability analyses typically have no direct connection with the system architecture model, which accurately depicts the physical and functional constructs of a system, including the "ilities". The model is directly impacted by the results of the analyses. This creates a time-consuming iterative process of analyses and modification because of the need to integrate several tools and teams. To improve this process, the research described here investigated the use of a single, cloud-based MBSE CAD tool called Innoslate that integrates failure analysis into the system architecture model. The specific focus of the research was on the analysis of system failure events through the use of a system architecture-modeling tool and the establishment of an MBSE process that enables system engineers to make risk-informed system modifications during development. The conclusion of the research was that MBSE in general, and Innoslate specifically, is capable of providing an integrated, effective and quantitative means of developing a risk-informed system design using a minimum functionality baseline process. This can be applied to human and robotic spaceflight systems and other systems with similar complexity. The research demonstrated that random distributions could be added to failure probabilities in order to add "noise" to the results, a task that can be laborious, if not impossible, if performed using a calculator or spreadsheet. The research also demonstrated an end-to-end MBSE process that was applied to a basic system model and the Altair Project. Recommendations for future work conclude the paper.

TABLE OF CONTENTS

Application of MBSE to Risk-Informed Design Methods for Space Mission Applications

"Science is a way to teach how something gets to be known. In as much as anything can be known, because nothing is known absolutely. It's how to handle doubt and uncertainty. Science teaches us what the rules of evidence are. We mess with that at our peril." – Richard P. Feynman[1]

INTRODUCTION

Legacy and modern systems for human and robotic space missions are often comprised of many intricate parts that must work harmoniously to accomplish a mission. Standing at 184 feet and weighing over 4.5 million pounds at launch, NASA's Space Shuttle was composed of over 2.5 million parts, including the Orbiter, External Tank and Twin Solid Rocket Boosters.[2] Despite the arduous effort to avoid the loss of a mission or crewmember, 2 of the 5 operational orbiters were lost along with their fearless crew.

Engineers today are shouldered with the challenge and responsibility of preventing these accidents from occurring through robust and evolving engineering practices. Risk Management (RM) is an important function to perform throughout the lifecycle of any mission that has "repeatedly proven capable of uncovering design and operational weaknesses that had escaped even some of the best deterministic safety and engineering experts."[3] However, system engineers often lack the necessary tools to perform quantitative risk assessments early in a mission life cycle when the design is very flexible and adept to changes. The growing practice of Model-Based Systems Engineering (MBSE), an object-oriented method of system modeling, offers an opportunity for system engineers to implement risk assessments without the need of a

large design team or very advanced reliability software. This paper describes research into the application of MBSE tools and processes to Risk-Informed Design (RID).

RID enables system risk analyses early in the lifecycle of spaceflight projects allowing designers to use risk as a design commodity and part of the overall trade space. RID uses a “minimum functionality” approach, whereby a minimal, single-string system design is first envisioned that only meets basic performance requirements without any regard to overall reliability or safety. Risk analyses are then used to apply informed design enhancements based on their contribution to risk reduction. A recent application of RID was the Altair Lunar Lander Project that was intended for human lunar exploration under NASA’s Constellation Program. The Altair project’s approach and results are reviewed and analyzed in this paper as a specific application of RID.

Unfortunately, these analyses traditionally occur later in the design cycle when changes are more difficult to implement. Safety and reliability analyses typically have no direct connection with the system architecture model, which accurately depicts the physical and functional constructs of a system, including the “ilities”. The model is directly impacted by the results of the analyses. This creates a time-consuming iterative process of analyses and modification because of the need to integrate several tools and teams. To improve this process, this paper investigated the use of a single, cloud-based MBSE CAD tool called Innoslate that integrates failure analysis capabilities into the system architecture model. The specific focus of the research is on the analysis of system failure events through the use of a system architecture-modeling tool and the establishment of an MBSE process that enables system engineers to make risk-informed system modifications during design and development.

BACKGROUND REVIEW

This research paper is first and foremost about the leveraging of Model-Based System Engineering (MBSE) methods to enhance Risk-Informed Design (RID) methods. According to the International Council On Systems Engineering (INCOSE), MBSE is the "formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases."[4] It is an integrated, object-oriented and model-centric approach that enhances the system engineering experience to efficiently manage moderate to complex systems that is particularly useful for spaceflight programs. Traditional engineering methods use a document-centric approach in which the constructs of a system are defined by physical and/or electronic documentation. Many of these documents are products of a large variety of engineering CAD tools. Table 1 highlights some of the major differences between a document-centric and a model-centric approach adopted from a Vitech Corporation seminar on MBSE.[5]

MBSE tools can capture a wide range of system artifacts from system requirements and CONOPS to test plans and mission scenarios, including timelines. Engineers across multiple disciplines can use a single tool to track the development of a system. As long as the same system-modeling tool is used, engineers will have access to the latest system updates. This leaves less room for error and reduces risks. Although MBSE is a relatively new concept that is gaining more attention from the systems engineering community, RID has been in practice over many decades.

Document-Centric	Model-Centric
Documents are physically collected	Documents derived from a single model
Drawings are independent of each other	Views are consistent with each other
Diagrams exhibit static behavior	Diagrams exhibit executable and dynamic behavior
Data is stored across multiple locations and mediums	Data is stored in a linked repository
System views are stored	System views are dynamically generated
Process is ad hoc with inconsistent results	Process is repeatable with consistent results
Changes are manually propagated across all affected system items	Changes are automatically propagated across all affected system items

Table 1. Document-Centric Versus Model-Centric System Engineering[6]

RID is a conscious engineering effort to buy down risks on a system by making informed design trades during system design and throughout the lifecycle.[7] It assumes risk is a design commodity to be traded -- similar to mass, thrust, size or power -- rather than a result of the design.[8] Risk is often thought of in three components: scenarios, likelihoods and consequences.[9] Risk can also be broken down quantitatively into a probabilistic value as a success or failure criteria. For spaceflight missions, there is special interest in understanding the probability of the loss of crew (pLOC), loss of mission (pLOM) and/or loss of vehicle (pLOV). These probabilities are determined by assessing the probabilities of contributing failure events caused by the components that decompose a system and/or environmental factors such as Micrometeoroids & Orbital Debris (MMOD) or Galactic Cosmic Radiation (GCR).

The RID process aims to identify these risks early in the development cycle to make effective modifications to a system as it evolves into a deliverable product without blindly and unnecessarily applying risk reduction solutions (i.e. “make everything redundant and hope for the best”). However, the analysis tools, personnel and other resources required to apply RID are typically out of reach for system engineers during early development. In traditional projects, separate tools such as Relex, Windchill and SAPHIRE are used to perform risk assessments. The results of these analyses can sometimes have a dramatic impact on system design that may fail to effectively propagate to all affected components. MBSE CAD tools such as Innoslate can assist system engineers to expedite the RID process by integrating the failure analysis into the system architecture model, enabling early risk assessment and efficient tracking and updating of risks throughout the lifecycle.

Innoslate is a cloud-based MBSE CAD tool developed by Systems & Proposal Engineering Company (SPEC) Innovations led by Cynthia Mahugh-Dam and Dr. Steven Dam.[10] Innoslate implements a Lifecycle Modeling Language (LML) that captures the technical and programmatic constructs of a system. LML combines the logical constructs of SysML with the ontology of DoDAF Metamodel 2.0 (DM2).[11] The technical constructs can be broken up into model entities such as Actions, Assets, Artifacts, Characteristics and/or Statements. The programmatic constructs can also be broken down into entities such as Cost, Schedule, Risk and/or Time.

The fundamental power of this tool and most other MBSE tools, such as CORE or CRADLE, is the ability to create direct relationships within the entities of the model. This enables efficient tracking of system requirements and the automatic propagation of

changes made to the model. System engineers receive immediate feedback on changes implemented on the model.

Innoslate provides some unique capabilities when compared to other MBSE tools such as having an online data repository for artifacts, the ability to concurrently update the model and providing commentary feedback to specific entities in the model that can be viewed as a report. It is also platform independent, meaning that the tool can be used on a Mac or PC. Innoslate also offers readily available templates that can be imported into the model. These templates include DoDAF, CONOPS, WBS, JCIDS, MODAF and others. This research paper uses Innoslate as the primary tool to implement failure analysis on a basic system model with further extension to the Altair project.

MBSE RID PROCESS

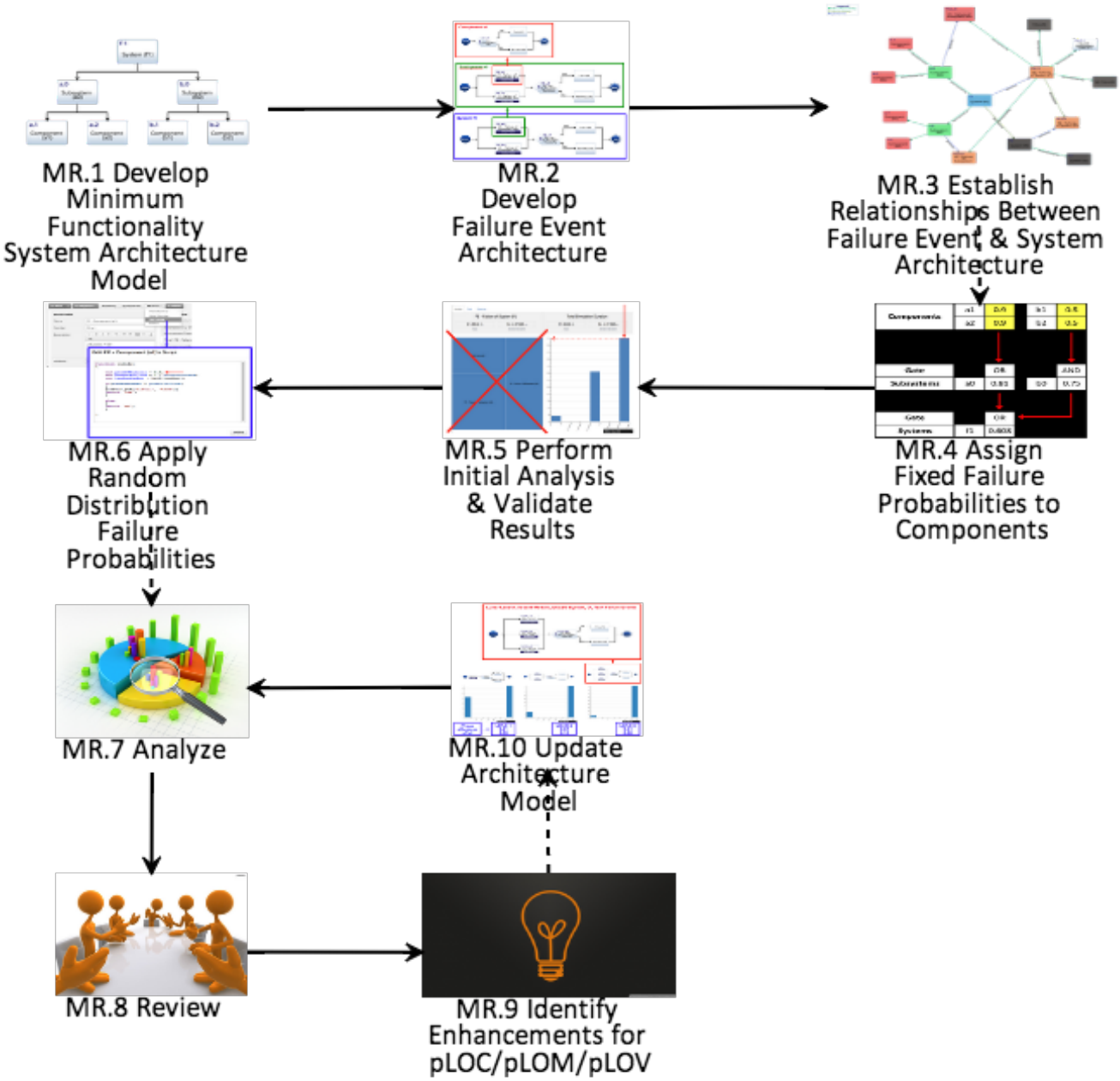


Figure 1. MBSE RID Process Flow Diagram

This section describes the process of applying MBSE to RID methods. It uses a minimum functionality RID approach and applies it to a basic generic system model developed using Innoslate. Each step in the process is described with general notes, assumptions and diagrams to guide the reader. Figure 1 depicts the MBSE RID process step-by-step and it is discussed in detail in this section.

MBSE RID Process:

1) Develop Minimum Functionality System Architecture Model
2) Develop Failure Event Architecture
3) Establish Relationships Between Failure Event & System Architecture
4) Assign Fixed Failure Probabilities to Components
5) Perform Initial Analysis & Validate Results
6) Apply Random Distribution Failure Probabilities
7) Analyze
8) Review
9) Identify Enhancements for pLOC/pLOM/pLOV
10) Update Architecture Model
11) Repeat Steps 7 to 10

Develop Minimum Functionality System Architecture Model

The MBSE RID process starts with the development of a system architecture model that includes the physical and functional constructs of a minimally functional system. For this research paper, the functional constructs are omitted for simplicity. Figure 2 depicts the physical architecture built in Innoslate. The model is generic and uses simple designations that can be very helpful later as the model gets more complex. The designations do not follow any particular standard. Innoslate has a built-in capability that can conveniently search for the designations when building diagrams or establishing relationships.

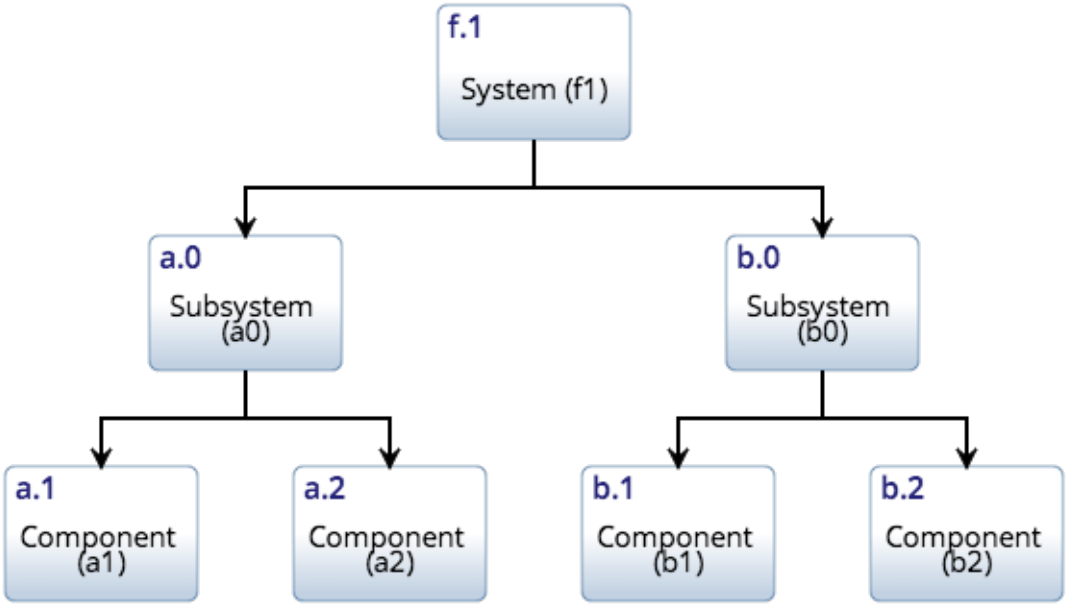


Figure 2. System f1 Hierarchy Chart

Develop Failure Event Architecture

After the system architecture model is built, the engineering team reviews the model to identify any potential failure events that the system may produce. For simplicity, the research paper identifies 1-for-1 failure events for each of the components and systems identified. For example, Component b1 performs a Failure Event of Component b1 and System f1 performs a Failure Event of System f1. Multiple failure events can be identified and captured by the model from a single component.

Once the failure events have been identified, it is then placed into the model as an independent architecture, which is essentially a fault tree diagram. While the physical architecture entities are “Assets”, functional and failure event architecture entities are “Actions”. At the bottom of any failure event architecture are the failures produced by the components that comprise the system. In general, the failure of any system is determined by the failure its components. There are environmental factors to consider as well, but this research paper does not account for those factors. A SimScript program, similar to JavaScript, embedded into each of the logic entities generates the probabilities. The user embeds the SimScript. Appendix I provides more information on the details of the SimScript. Figure 3 depicts a sample of the failure event model for System f1 that is broken down to the bottom level where the failure probabilities are generated.

The Hierarchy Diagram view of the failure event architecture provides a better perspective of the relationships between each of the events; however, the views are too large to view in this document. The physical hierarchy chart in Figure 2 provides a

similar view, but does not include the logic gates. Appendix I provides information on how to access to Innoslate model if there is independent interest to view the entire model. Although the logic entities in Figure 3 show an “OR” description, the SimScript may actually implement an AND logic function instead. It is recommended to add a brief descriptive note on each of the logic entities to make the distinction (i.e. FG.OR, FG.AND) as shown in Figure 3.

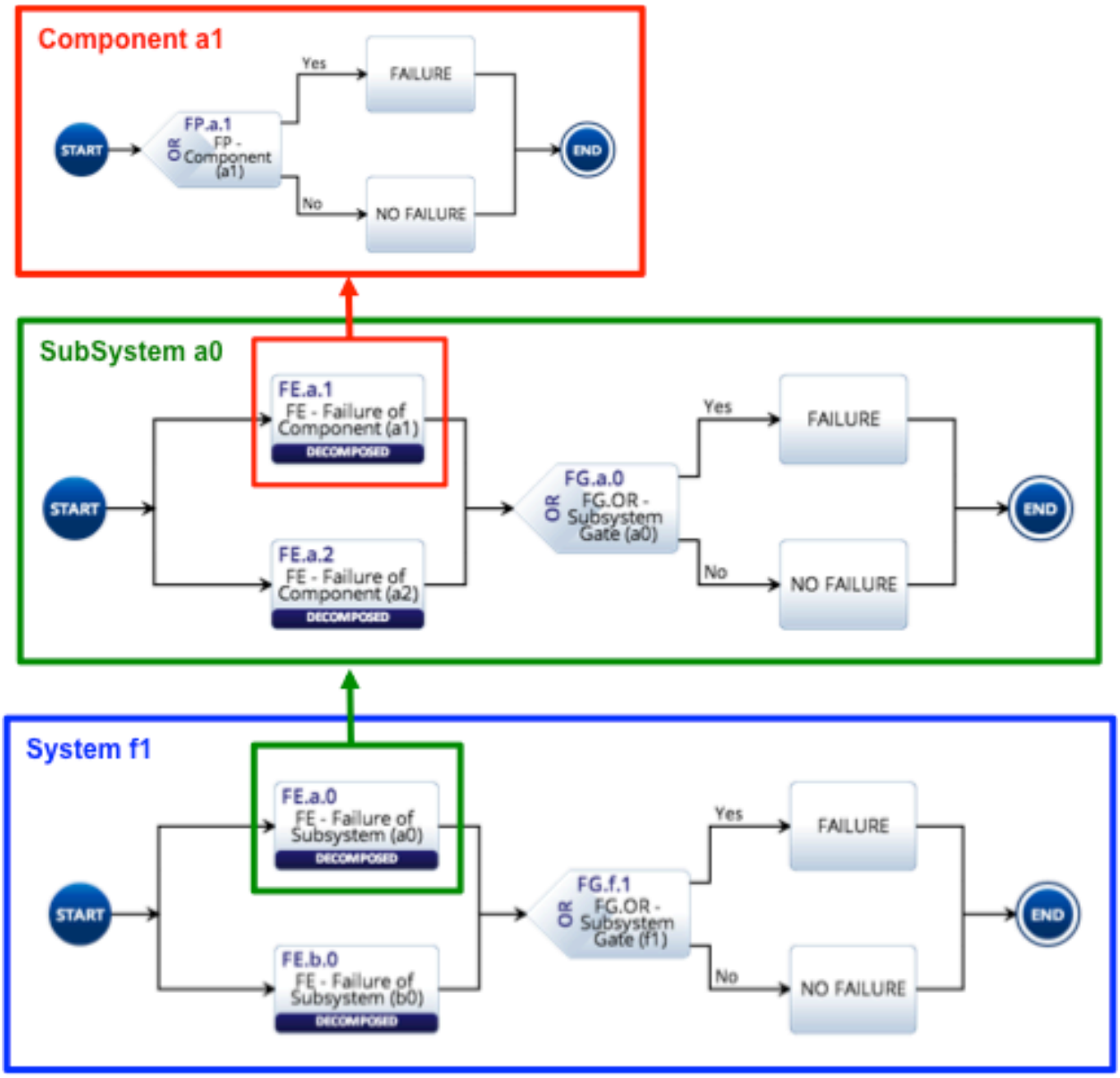


Figure 3. System f1 Failure Event Architecture Breakdown Action Diagram

Establish Relationships Between Failure Event & System Architecture

The failure event architecture was previously built separately from the physical model. Now relationships can be established between the two architectures by linking each component and system to its respective failure event. This is accomplished by going into each of the model assets and adding a link in the “performs Action” relationship. Figure 4 shows the “performs Action” entity location with a red arrow. This can be accomplished vice versa from the action entity, but the relationship is instead called “performed by”.

With relationships established, changes can be tracked and updated automatically without having to go into every affected entity in the model. An interesting view to use is the “spider diagram”. Figure 5 depicts the spider diagram for System f1 and includes a color legend for easier tracking. This figure shows how the model elements are related and traced to each other. It is broken down to 3 levels, but can be broken down further up to 9 levels. For complex architectures, high levels may become difficult to interpret.

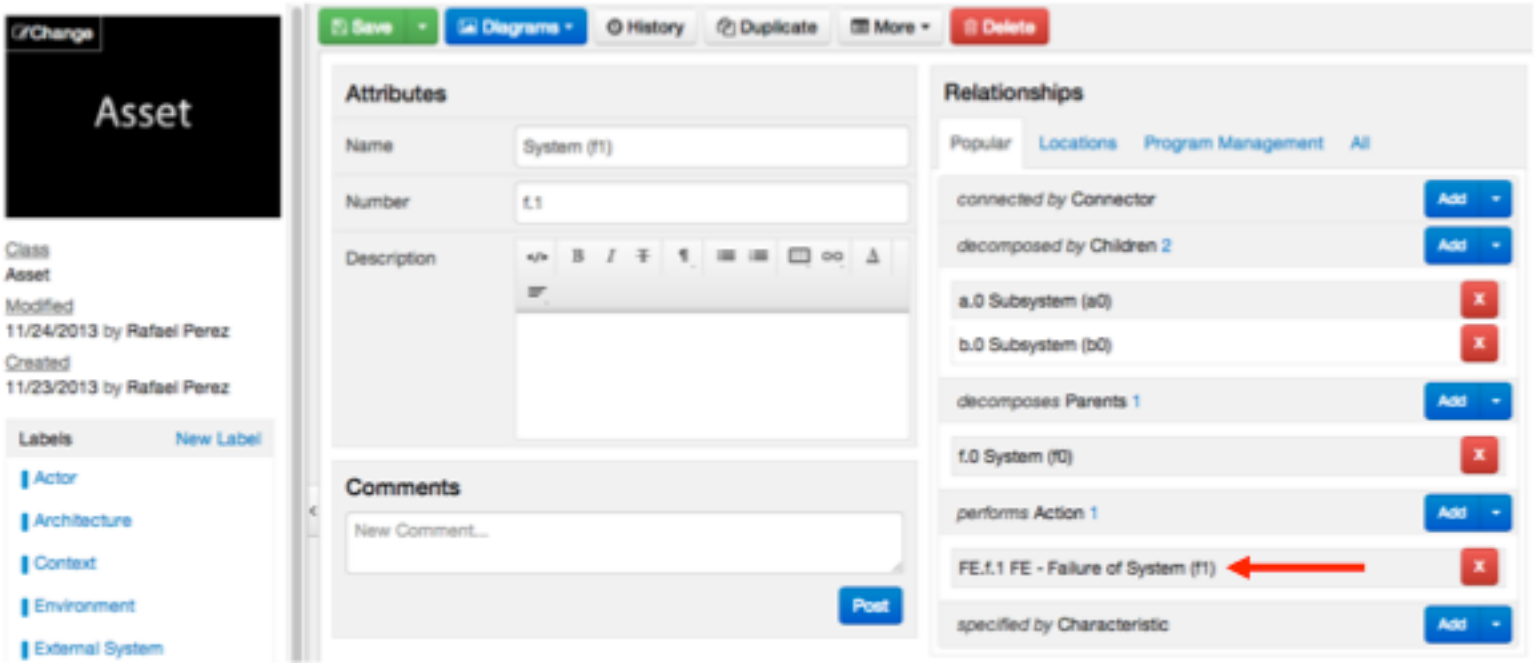


Figure 4. System f1 Asset Entity View

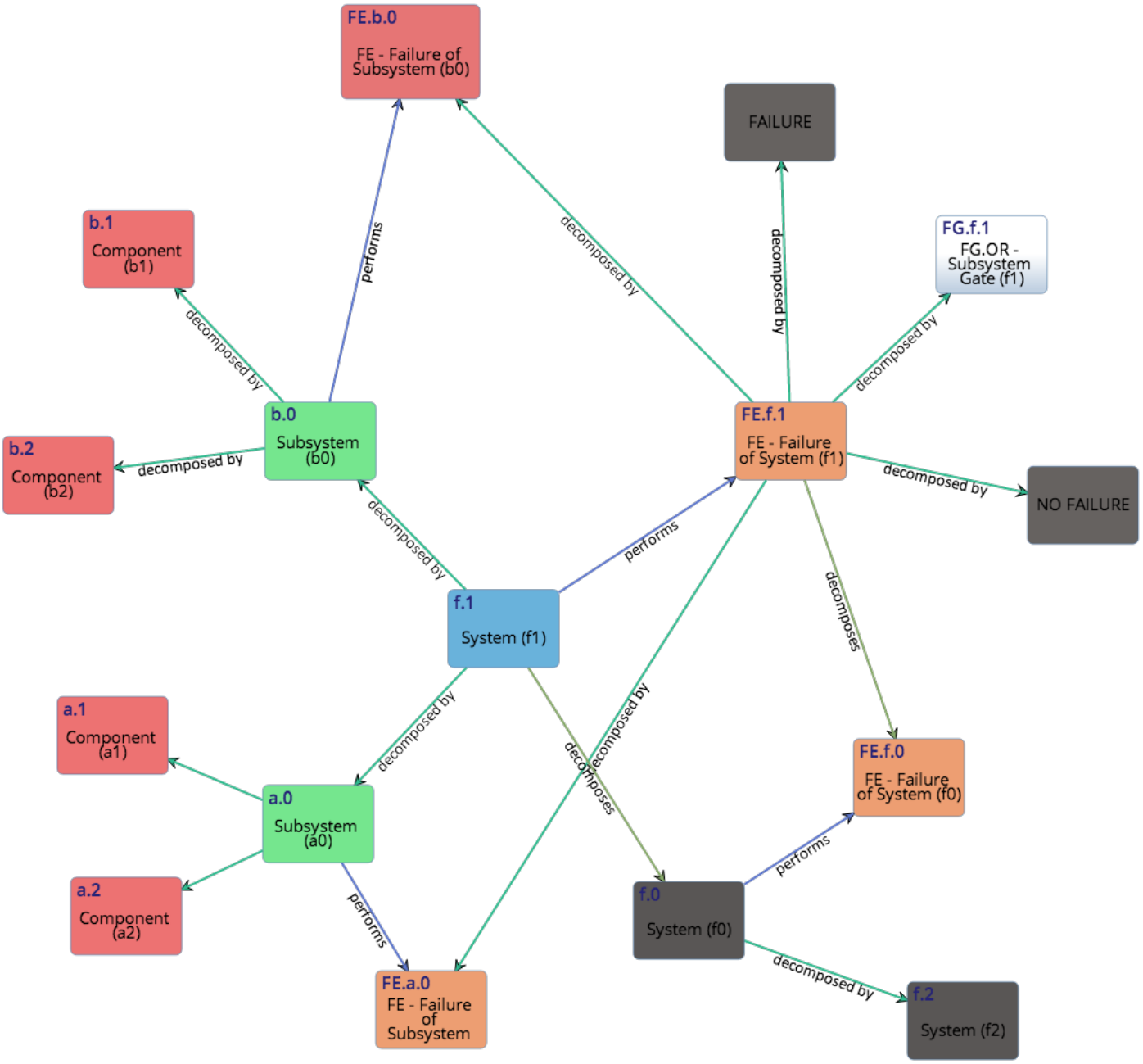


Figure 5. System f1 Traceability Spider Diagram

Legend
Blue = System
Green = Subsystem
Red = Component
Orange = Failure Event

Assign Fixed Failure Probabilities to Components

To ensure that the model is built correctly, it is recommended to first assign fixed failure probabilities to each of the components before adding any kind of “noise”, or random distribution, to the values. The probability values assigned to the components in this research paper are generic and arbitrary. There are several resources that can be used to determine a probability value such as “Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners”, “NASA Risk-Informed Decision Making Handbook” or “Human Spaceflight: Mission Analysis And Design”.[12,13,14] In general, for first-cut analysis, most failure probabilities are adopted from legacy or similar systems. Engineers also use reasonable judgment based on their personal experience. Determining failure probabilities is a process all in itself and it is not addressed in this research paper.

Figure 6 depicts the failure probability assignments for each of the components (Highlighted in Yellow) and the resulting system failure calculations performed in Microsoft Excel. Note that the numbers defined are actually probability of success values that are used to better compare results with the Innoslate analysis. To determine the actual failure probability, subtract the probability value from 1. For example, if pSuccess of f1 = 0.608 (Shown in Figure 6), then pFail of f1 = 1 – 0.608 = 0.392.

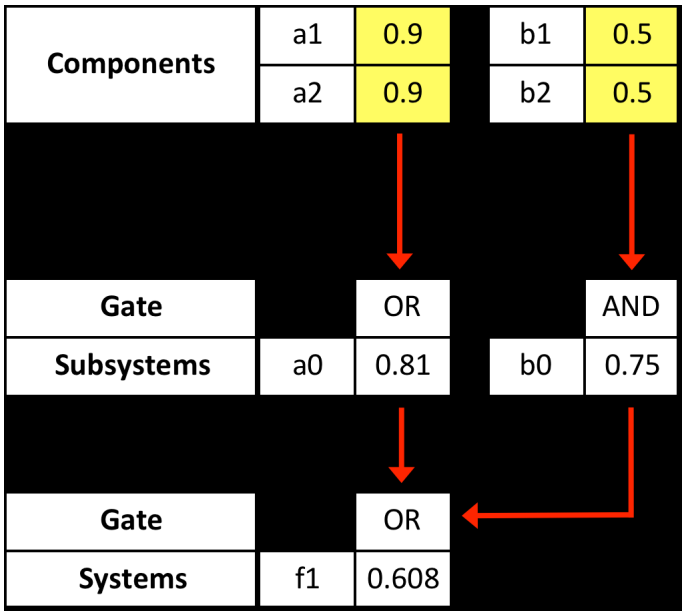


Figure 6. System f1 Failure Probability Assignment & Calculation

Perform Initial Analysis & Validate Results

The Excel calculation in Figure 6 should only be performed for simpler parts of the model to be able to validate the results from Innoslate. As the model becomes more complex and noise is introduced into the probabilities, it will be become a very laborious effort to work out the calculation. This paper continues to crunch this calculation for other parts of the process to show the reader that the manual calculation agrees with the Innoslate analysis.

For OR logic functions, the following equation is applied:

[x1 * x2 * x3 …]

For AND logic functions, the following equation is applied:

[1 – (1 – x1) * (1 – x2) * (1 – x3) …]

The SimScript uses the respective logic operators to implement the function:

[OR = ||] [AND = &&]

The next task is to perform the failure analysis in Innoslate. Using the Action Diagram for the System f1 Failure Event, a Monte Carlo Simulation can be performed in order to quantitatively assess the failure probability of System f1. Run the simulation and view the graphical results. Note that the Monte Carlo simulation function is not part of the free package and requires a Professional subscription plan. The Monte Carlo simulation runs 100 iterations maximum of the failure events and it is performed using cloud-based computing. When evaluating the results, look at the percentage of simulations that passed and failed in the bar graph on the right. During each iteration count, passing events add time to the simulation through the use of the “No Failure” entities while failure events add no time through the use of the “Failure” entities.

Therefore, simulations that passed will accumulate in the bar with the longest time on the extreme right of the bar graph.

Any failure in the event chain over the 100 iterations will show a shorter time on the left of the graph, meaning if the user were to add up the bars on the left, it would add up to the failure probability, while the extreme bar on the right adds up the success probability. Appendix II provides more information on Innoslate's Monte Carlo simulation and how it can be optimized to sum up the failure bars into a single bar. Figure 7 depicts Monte Carlo simulation graph generated by Innoslate. The graph on the left and the duration numbers on top can be ignored. The results of the Innoslate analysis show probability of success to be 0.60 (60%) while the results of the Excel spreadsheet in Figure 6 showed 0.608, an error of only 0.008. The number determined from the graph is "eyeballed" to the y-axis labels. Innoslate is working on an update to allow users to hover-over the bar to see an exact number as can be done with the graph on the left. Users will also notice that the analysis results will fluctuate up or down by a few percent. Unfortunately, there are not enough iteration counts to give an accurate result consistently. The simulation function is still experimental, but the Innoslate team is working on perfecting the simulation and providing more capability to the user community.

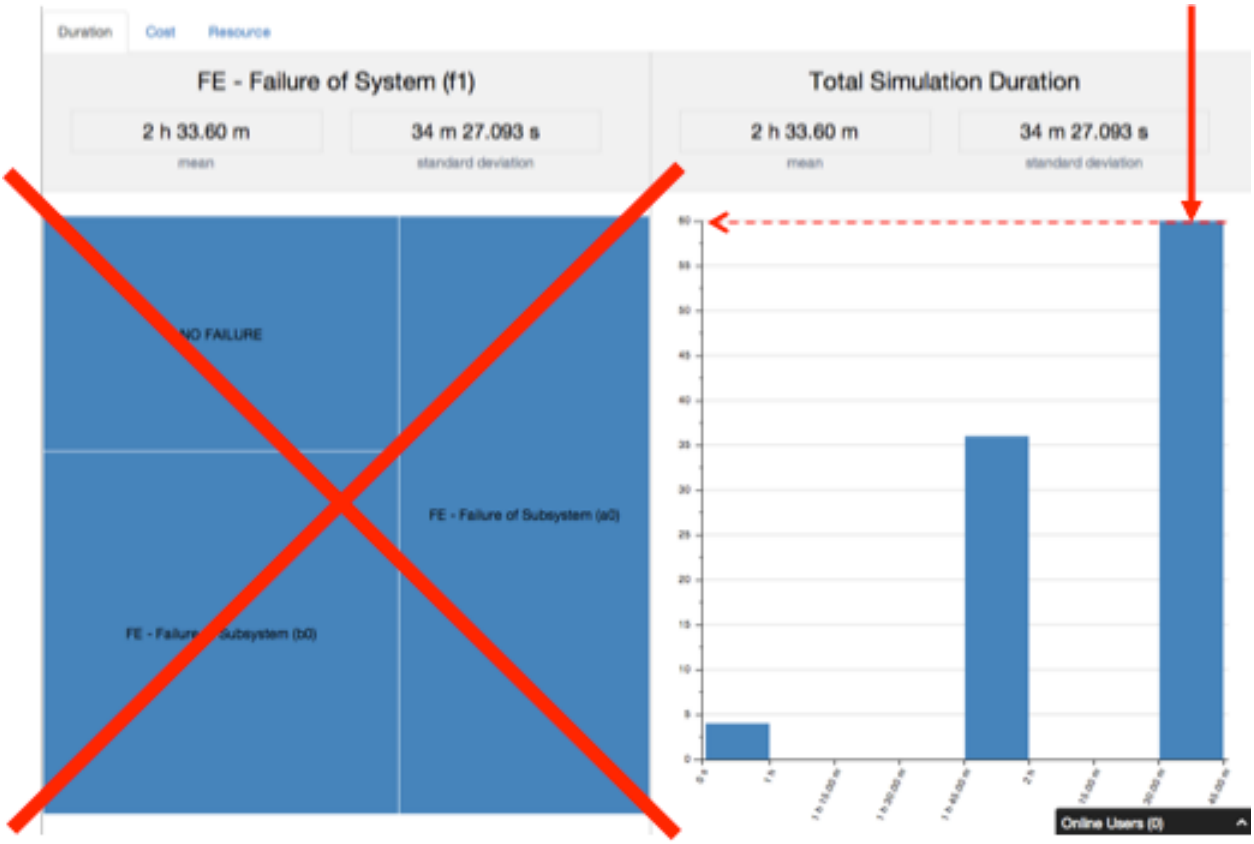


Figure 7. System f1 Innoslate Failure Analysis

Apply Random Distribution Failure Probabilities

Applying random distribution to failure probabilities is an advantageous capability offered by Innoslate through SimScripting. During system design, determining the exact failure probability of a component is often difficult to obtain and because of its statistical nature, can never be truly ascertained. There are space missions that have been short-lived and ones that have far exceeded its predicted lifetime. Therefore, variations need to be implemented on each of the assigned failure probabilities to "shake-up" the analysis and determine if the design still meets criteria. 'Math.random()' is a JavaScript function that returns a floating-point, pseudo-random, and non-uniform distribution number between the number 0 and 1 that is inclusive of the number 0, but exclusive of the number 1.[15,16] With this function, variations can be applied to each of the failure probabilities defined for the components. Figure 8 depicts the SimScript function for Component a1 with a red arrow where the function is applied. Note that there is another 'math.random' function in the script, but this is used to randomly decide if the component fails or passes while the 'probOfSuccess' (Red Arrow in Figure 8) sets the threshold. The component fails every time the random number is generated below 'probOfSuccess', in this case 0.9. Appendix I provides more information on the SimScript. Adding a 'math.random' function to the 'probOfSuccess' variable will vary the failure threshold at every iteration count of the Monte Carlo simulation. The threshold essentially becomes a moving target. The following distribution equation is implemented to 'probOfSuccess' for Component a1:

var probOfSuccess = Math.random()*(0.95-0.85)+0.85;

The distribution equation above will vary ‘probOfSuccess’ from 0.85 to 0.95, or ± 5% from 0.9. If ± 5% variation was implemented on all components of System f1 (i.e. Component a1, a2, b1, b2), the failure probability of System f1 should vary approximately ± 10%. This distribution is applied to the System f1 model in this research paper. Table 2 shows the results of 5 Monte Carlo simulation runs, which is equivalent to 500 iteration counts. It is interesting to note that although variation was shown in each run, over all the runs, the final average was approximately the fixed results from Figure 6 and 7. This is a contribution of the non-uniform distribution of the random function. Although the average eventually works itself out to the fixed value, the standard deviation becomes a valuable number to account for because it gives the user an indication of how much the failure probability varies from the mean.

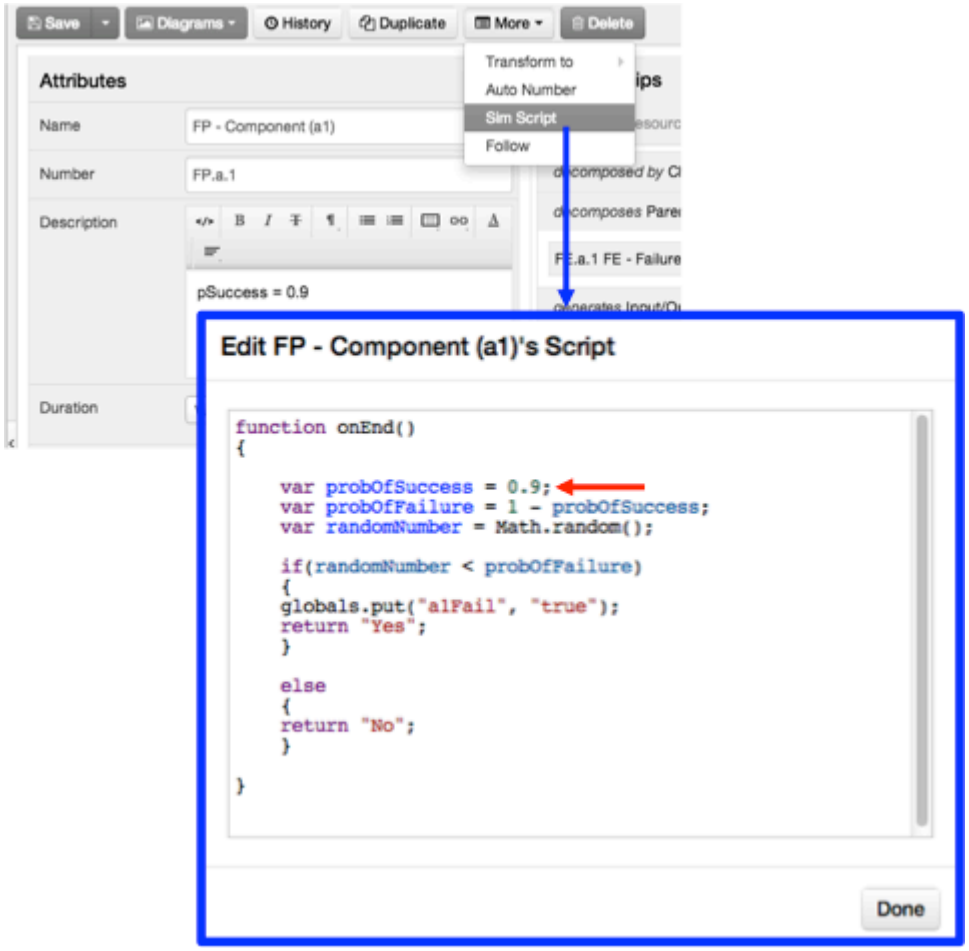


Figure 8. Component a1 Innoslate Failure Probability SimScript

System f1 Distribution Monte Carlo Run	Success Probability [probOfSuccess]	Failure Probability [1 - probOfSuccess]
1	0.57	0.43
2	0.67	0.33
3	0.56	0.44
4	0.62	0.38
5	0.59	0.41
Average	0.602	0.398
Standard Deviation	0.044384682	0.044384682

Table 2. System f1 Monte Carlo Runs With Distribution Applied

Analyze, Review, Identify Enhancements for pLOC/pLOM/pLOV & Update Architecture Model

The next step is an iterative cycle of analysis, review, identification and updates. The design team, subject matter experts and relevant stakeholders typically participate in this process. This research paper does not detail this process, but the documents identified in the "Assign Fixed Failure Probabilities to Components" section provide more information.[17,18] This research paper, however, continues the process by providing 2 arbitrary updates to the model shown in Figure 2 that can be applied to pLOC, pLOM or pLOV analysis scenarios. For simplicity, random distribution is not used for the updates and the distribution values identified in the previous section have been reset to fixed values.

Figure 9 shows the summary of the first set of updates for System f0 with the addition of Subsystem c0. The Excel calculation resulted in a probability of success of 0.731 (pFail = 0.269) while the Innoslate simulation resulted in 0.73 (pFail = 0.27). Figure 10 shows the summary of the second set of updates for System f0 with the addition of Subsystem d0. The Excel calculation resulted in a probability of success of 0.925 (pFail = 0.075) while the Innoslate simulation resulted in 0.93 (pFail = 0.07). Table 3 summarizes the Innoslate simulation results across the 3 phases shown in this section.

Figure 9 and 10 demonstrates that although the reliability of the system improved significantly (+0.33), it came at the sacrifice of additional complexity to the system that typically results higher in costs, longer schedules, higher mass, larger volume, more electrical power and/or extra resources. In the long run, a program saves money because this process is a conscious effort to add only the essential assets for a

successful mission, not a blind effort that adds complexity with insignificant impact to safety and reliability. The conscious effort also helps engineers identify and track the key drivers of system safety and reliability that can be addressed in a timely manner.

System f0 Phase	Success Probability [probOfSuccess]	Failure Probability [1 - probOfSuccess]
Baseline	0.6	0.4
Update 1	0.73	0.27
Update 2	0.93	0.07

Table 3. System f0 Innoslate Failure Analysis Summary

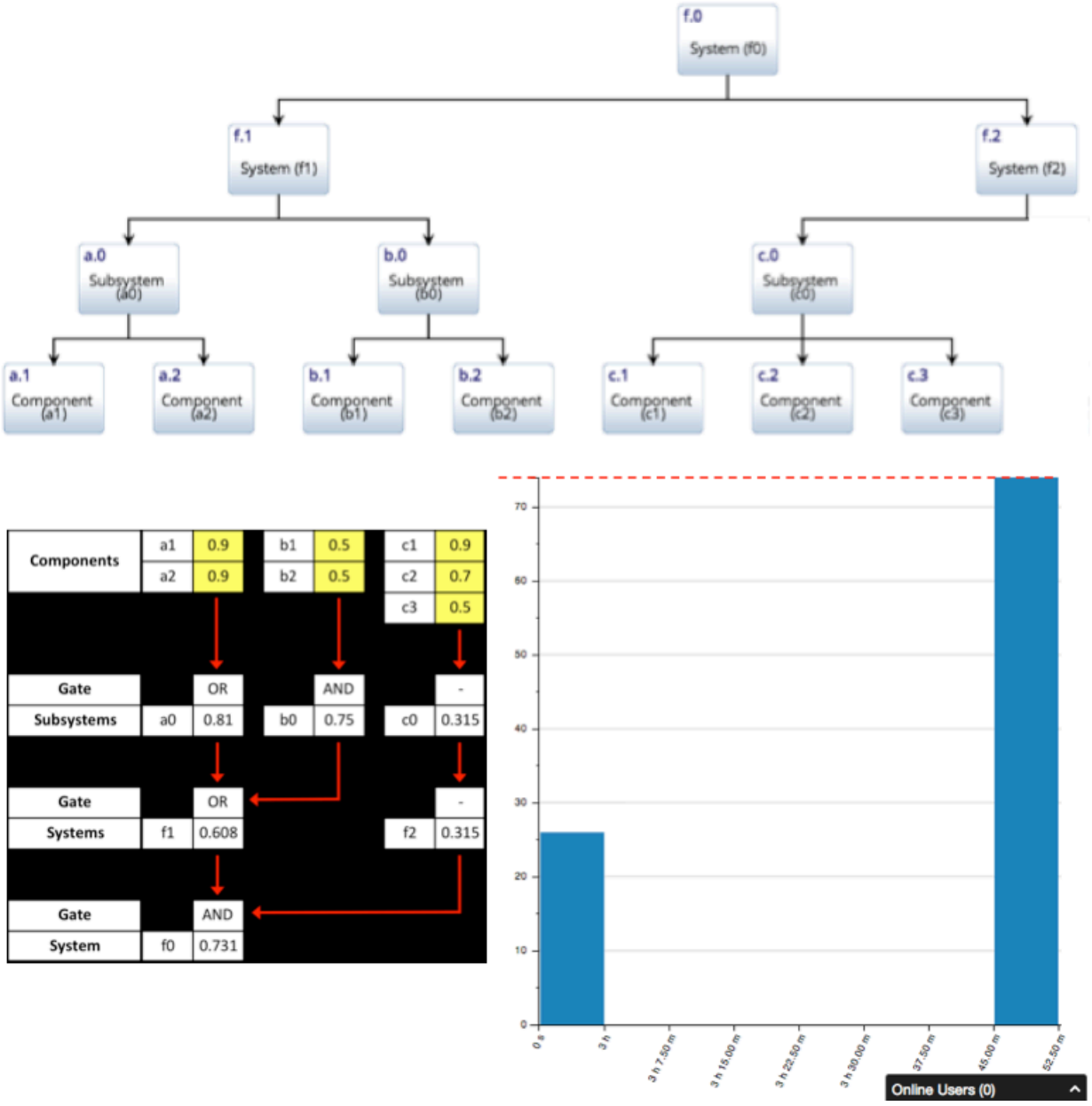


Figure 9. System f1 Update 1

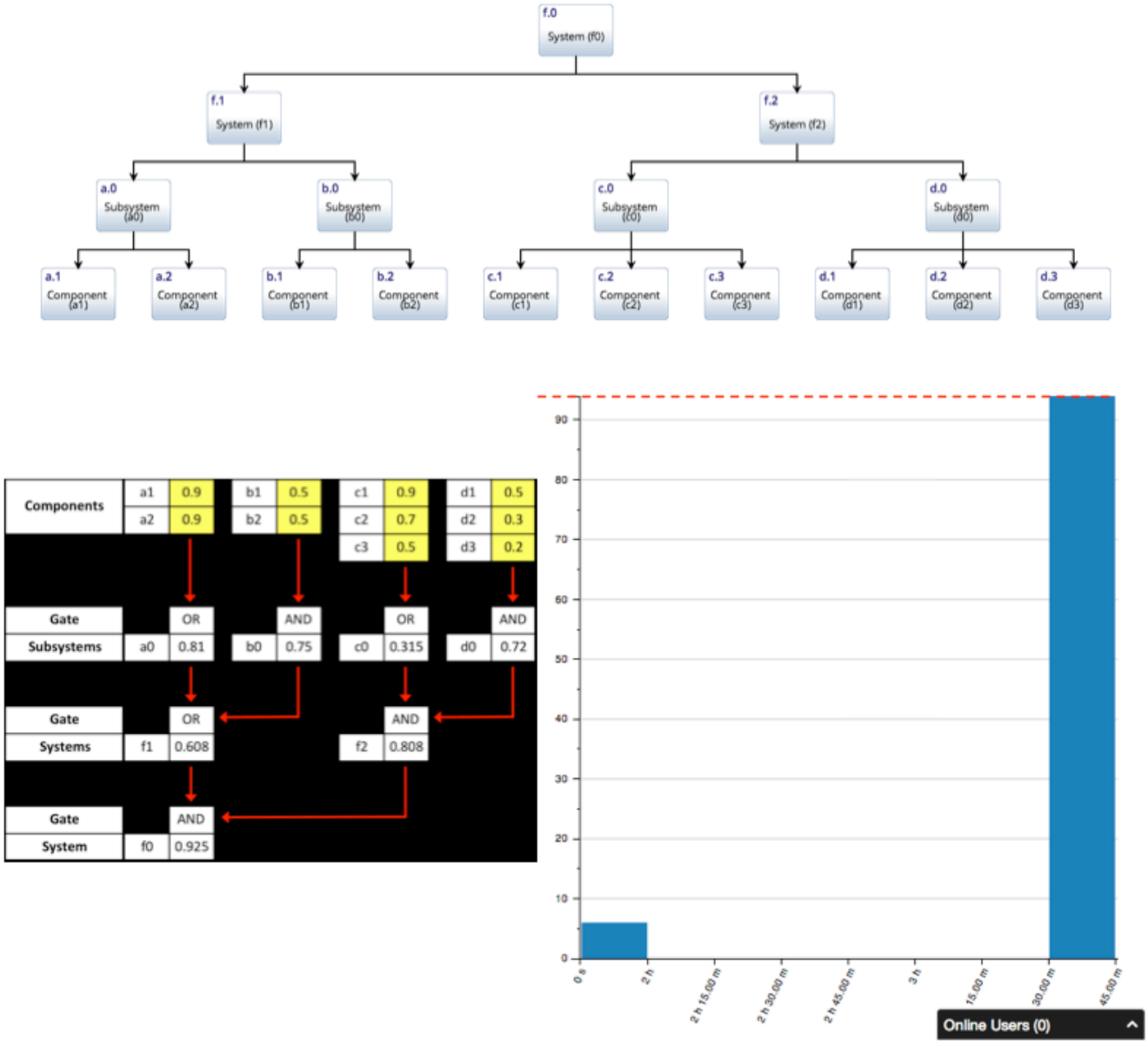


Figure 10. System f1 Update 2

ALTAIR DESIGN REFERENCE EXAMPLE

The Altair lunar lander was comprised of four major components, which were an Ascent Module (AM), a Descent Module (DM), an Airlock and an Ares V Earth Departure Stage/Altair Adapter (EDSA).[19] The Altair design implemented a RID process comprised of four phases called Lander Design Analysis Cycles (LDAC). LDAC-1, or the first phase, provided a "minimum functionality" baseline vehicle. This is a stripped down vehicle that performed only the very basic functions to accomplish the mission.[20] It did not account for any safety or reliability requirements for the mission or crew.

In LDAC-2, engineers essentially "bought back" crew safety to enhance pLOC primarily at the cost of mass. Added capabilities to the vehicle to improve pLOC included abort functions, redundant O_2 tanks, and an emergency communication system, among other capabilities.[21,22] Engineers were careful to include only capabilities that were necessary to significantly improve pLOC. There is a common intuition within engineers that adding redundancy is always the most effective method of improving safety and reliability in a system. However, a critical lesson learned during that Altair analyses was that "full redundancy was usually the most massive and frequently not the most effective option for improving LOC." [23] Because this is not always the case, it challenges engineers to come up with creative solutions that do not use redundancy. For example, an abort function is not a redundant item; however, it greatly enhances the safety of a system. Another example is adjusting the flight trajectory to one that is safer for humans, which is also a non-redundant item.

In LDAC-3, engineers bought back mission reliability to enhance pLOM, also primarily at the cost of mass. Added capabilities to the vehicle to improve pLOC included manual circuit breakers for the power distribution unit, changing the DM and Airlock primary structures to composites, and adding another O_2 tank, among other capabilities.[24] The design continued to LDAC-4 with very small increments of enhancements with natural design maturation, but nothing more was documented after LDAC-4 as program activity eventually halted due to the cancellation of Constellation.

As discussed earlier in the Develop Failure Event Architecture section, at the bottom of any failure event architecture are the failures produced by the components that comprise the system. The results of the failure analysis are only as good as the probabilities assigned to each of the components, whether it is a fixed value or a distribution, therefore, “garbage in equals garbage out”. Appendix IV provides an example of how probabilities were assigned to the components of the Altair lunar lander system, courtesy of Mr. Randolph Rust at NASA. Due to technical restrictions, the failure rates are generic and do not represent actual data, however, these numbers are representative of the probability numbers typically associated with space systems and their components.

The “minimum functional” design philosophy was new to large-scale NASA human spaceflight projects.[25] Although incomplete, Altair provided very useful insight to crew vehicle design that can be applied to future human spaceflight projects. Appendix III depicts a summary of the Altair lunar lander design changes as it evolved through the analysis cycles taken from a design lessons paper developed by the Altair design team. In addition to the RID process, Altair also implemented MBSE to model the system and

its activities. According to Dr. John Connolly and Mr. Randolph Rust at NASA, the team used CRADLE as the system architecture-modeling tool; however, the failure analysis for pLOM was performed on a separate tool called SAPHIRE (Version 7-26), which is controlled by the Nuclear Regulatory Commission and Idaho National Laboratory, while the analysis for pLOC was performed using a Microsoft Excel document developed by Valador Corporation. Fault trees were created using common applications such as Microsoft Word and PowerPoint. This is a prime example of how this research paper can enhance RID with MBSE by consolidating the system modeling and failure analysis effort into a single tool.

Unfortunately, due to the complexity of the Altair project and the limited information and time available, this research paper could not develop a complete and accurate model of the Altair lunar lander using the MBSE method described in this research paper to replicate the failure probabilities shown in Appendix III and make an “apples-to-apples” comparison. Instead, this research paper shows a brief example of how failure events in the Altair Lunar Lander are modeled and simulated through the 3 LDAC phases and how it contributes to pLOC and pLOM.

Figure 11 shows a sample physical architecture of the lunar lander that starts from the 4 major, top-level systems down to the O_2 tanks of the ECLSS System in the Ascent Module. O_2 Tank 1 is a minimum functionality item needed to keep the crew alive and accomplish the mission; therefore, it is included in the LDAC-1 analysis and folds into the pLOC and pLOM failure analysis. The O_2 Tanks 2 and 3 are added successively at each phase of the design cycle as long as the design team deems it

necessary through peer reviews. In this case, the design team deemed the additional O_2 tanks necessary according to the LDAC assumptions shown in Appendix III.

For simplicity, the success probability for each of the O_2 tanks is arbitrarily set to 0.6 (pFail = 0.4). Since it takes all 3 tanks to fail in order for the ECLSS System to fail, an AND logic is implemented. Figure 12 summarizes the Innoslate failure analysis implemented across the 3 design phases. Note the failure rate improvement as it progresses through the phases, but at the cost of adding new elements to the architecture resulting in higher in costs, higher mass and larger volume to say the least. Since there are numerous failure events required to determine pLOC and pLOM, it is not addressed in this research paper.

The failure of the O2 tanks in this scenario contributes to both pLOC and pLOM because not only would the crew not have breathable air that would suffocate them (pLOC), the lunar lander system would not have the required human inputs to complete the mission (pLOM). Determining pLOC and pLOM typically requires 2 separate failure event architectures, or fault trees; however, many of the failure events overlap such as in the case of the O_2 tanks. Innoslate has no issue reusing failure event entities for multiple architectures, therefore saving engineering design time. However, failure events within a single architecture must be unique because of the unique variables set within the SimScripts (i.e. a0Fail, b0Fail, f2Fail, etc.). This section demonstrated how the MBSE RID process described in this paper is applied to a specific human spaceflight project.

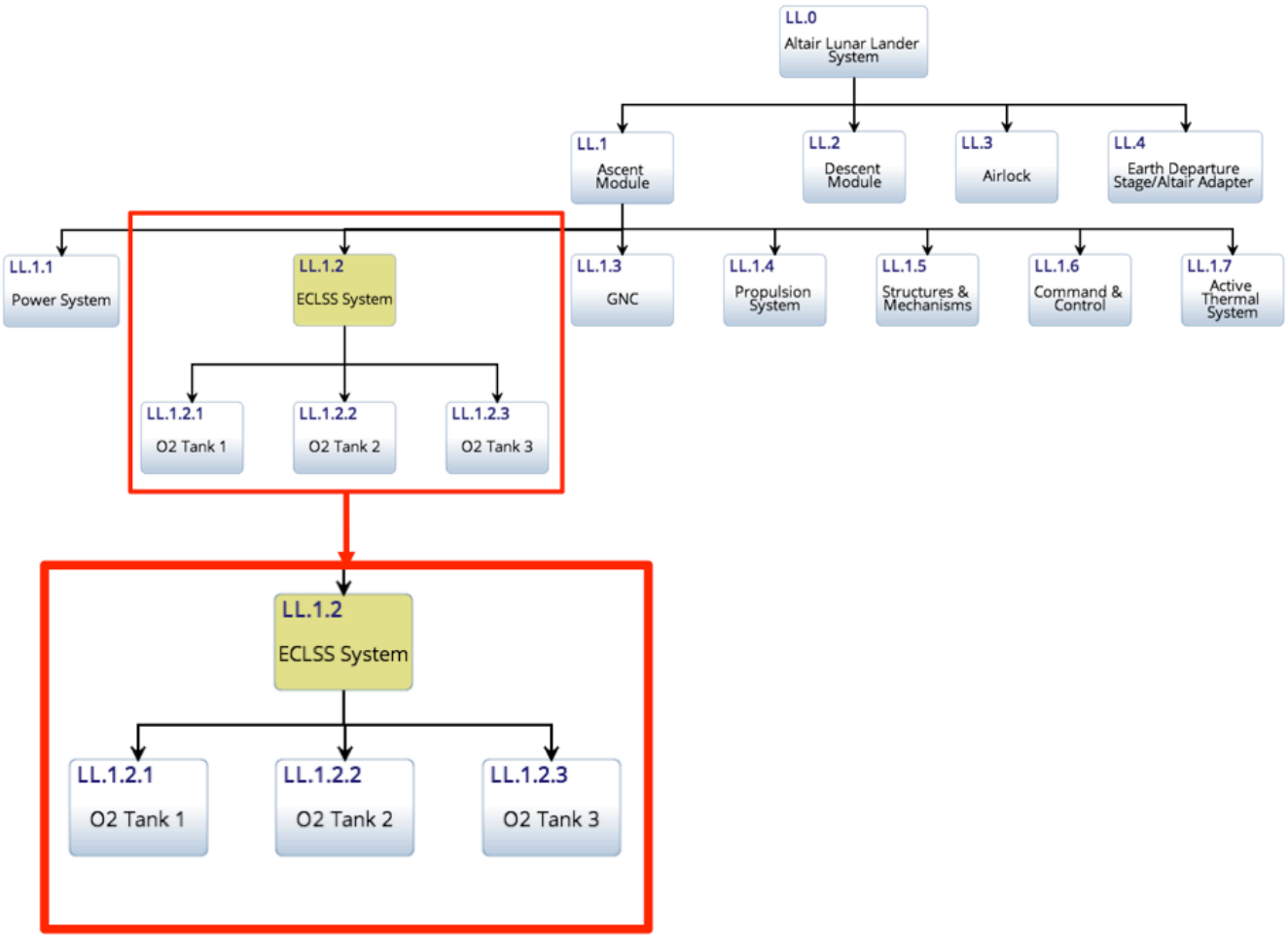


Figure 11. Altair Lunar Lander Sample Physical Architecture

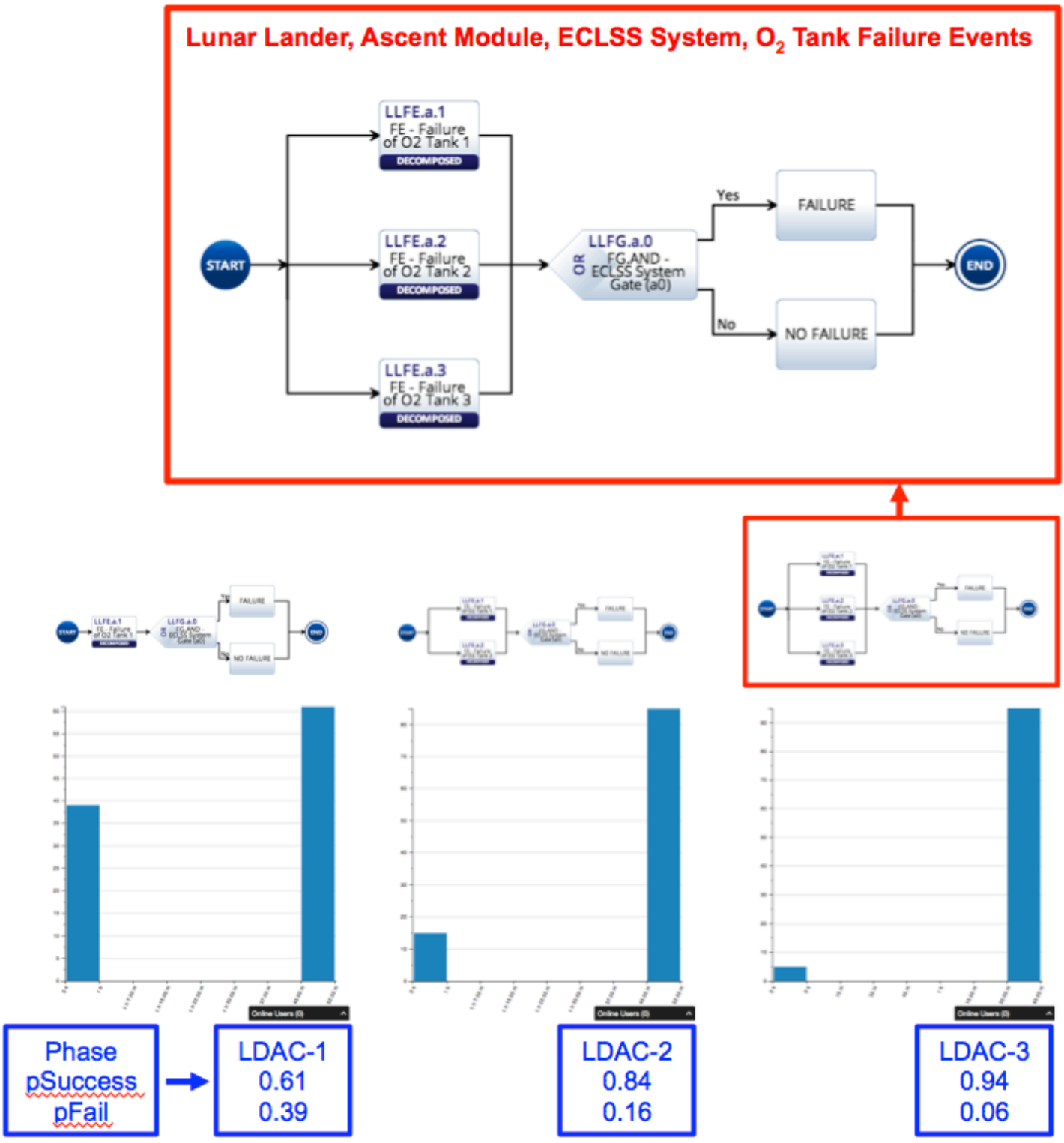


Figure 11. Altair Lunar Lander O_2 Tank Failure Events

LESSONS-LEARNED

This section highlights some of the lessons-learned in conducting this research and offers suggestions. In general, many of the lessons-learned were documented in the MBSE RID Process section. There were many unknowns at the beginning of the research that were underestimated and that needed to be investigated and worked out. The list below is not an exhaustive list of lessons-learned, but focuses on the important ones.

1. As the system and failure event architecture were being developed, reference designations were being assigned for each of the entities in the "number" field. It was straightforward at first to just assign designations in some kind of logical order (i.e. 'a.0', 'a.1', 'a.2'), but when changes needed to be made, it was a difficult task to make changes without ruining the logical order set, especially when it's somewhere in the middle of the order. Establish a reference designation strategy prior to model development. The strategy should be logical and flexible to allow for changes throughout the process. This can be implemented for any category of entities (i.e. failure events – FE.0, failure probability generators – FP.0, failure gates – FG.0, components – C.0, etc.).

2. In the sea of entities that can be created in the MBSE CAD tool, it became burdensome to sort the entities of interest for a particular modeling task and keep track of it. Establish a labeling strategy prior to model development. For small-scale models, using labels may not provide significant value, but for mid- to large-scale models, it becomes very useful, almost necessary, to efficiently manage the model. Innoslate offers a labeling feature that filter entities of a specific group determined by the user. This feature may not be offered in all MBSE CAD tools, but it is one to look for and use to improve the model development experience.

3. It was sometimes taken for granted that the failure event pieces in the model were setup with the correct configuration, connections and SimScripts. Results were therefore shown with error. Perform manual calculations of small parts of the failure event architecture to validate correct operation. This is a sanity check to ensure that the SimScripts and connections are working properly. Calculations can be performed with a portable calculator, but it is recommended to use Microsoft Excel, or any spreadsheet application, to perform the calculations.

4. One of the main items investigated in this research was the use of the MBSE CAD tool, Innoslate. The tool provided many great features as mentioned in the Background Review section. However, Innoslate still has room for improvement. It is a relatively new program that released its second version over the Summer 2013. The Monte Carlo simulation was considered experimental at the time of this research. Appendix I provides more details on the use of Innoslate and notes some of the minor issues experienced. As with any tool, there is a learning curve to account for, therefore, plan accordingly for some training time. However, engineers that are savvy with CORE or CRADLE will likely have a much smaller learning curve since much of the modeling concept and language are very similar. The SPEC Innovation technical team provides great support in answering questions, addressing issues and providing training opportunities (i.e. lunch & learn).

5. A critical lesson-learned taken from the Altair project was that full redundancy is often not the most effective option for improving failure probabilities. It is quite instinctive for engineers to simply add redundancy to improve safety and reliability. No direct examples were shown in this research paper, but this is a valuable note for engineers to take away and consider for future projects.

RECOMMENDATIONS

The following are recommendations for potential future research topics that can be taken up academically or professionally. The specific focus and limited timeline of this research paper prevented further investigation of these topics.

1. Non-Redundant Methods of Improving Safety & Reliability for Space Systems
    a. The Altair project has shown that full redundancy is not always the answer for enhancing safety and reliability within a system. Some non-redundancy options were noted such as abort capabilities and flight trajectory alternatives. Implementing non-redundancy sometimes requires creativity and ingenuity. The focus of this investigation could be on developing non-redundant options for space systems by collecting data from legacy and current systems, identifying common issues and solutions, identifying rare cases and creating novel approaches. Engineers can benefit by using the research as a reference guide that can be directly applied to a problem or as an idea kickstarter. This investigation seeks to answer a fundamental question, “What non-redundant alternatives for space systems can engineers implement to improve safety and reliability?”

2. Application of Dynamic Logic To System Architecture Modeling
    a. This research paper only covered static logic (i.e. AND, OR), which can execute series and parallel configurations in a system. There are other configurations such as Standby, Cross-linked, On-Demand and more exotic configurations that cannot be implemented using static logic, but rather using dynamic logic, which can account for temporal aspects of failure events. One example of dynamic logic is the use of Priority AND (PAND) gates, where the order in which the inputs occur matter. This research paper provides a couple of references on dynamic logic.[26,27] The focus of this investigation could be on developing SimScript or JavaScript algorithms that can implement dynamic logic and then integrate it into an MBSE CAD tool to execute time-based failure events such as Standby or Cross-linked. This can enhance the MBSE RID process by having the ability to simulate other types of failure events.

3. MBSE RID Process Failure Probability Analysis Extension Into MTBF Analysis
    a. This research paper focused on the quantitative analysis of failure events based on the failure rates of system components, but it does not translate any of the analysis for use in MTBF analysis. MTBF is simply a reciprocal of failure rate and it represents a ratio of total operating time to total number of failures within that time.[28] It is very often used in industry alongside failure rates. Modern MBSE CAD tools have been slowly incorporating Reliability, Availability and Maintainability (RAM) capabilities that can perform this analysis and perhaps bridge this gap. The focus of this investigation could be on developing algorithms or leveraging MBSE CAD features to translate failure rates found in this research paper into MTBF values for use as an alternative method to failure analysis.

4. Application of Alternative Random Distribution Types
    a. As discussed earlier, the ‘math.random()’ JavaScript function implements a pseudo-random and non-uniform distribution. It was difficult to track down references on the exact nature of the ‘math.random()’ function, but it would seem to implement a flat (i.e. even, 50/50) distribution, or at least very close to it, based on the results found in this paper. It would likely be of great interest to engineers to be able to implement different types of distribution such as Gaussian, Uniform or Logarithmic. The focus of this investigation could be on developing algorithms to implement several types of random distribution to failure rates in MBSE CAD tools.

CONCLUSION

The research demonstrated an end-to-end MBSE RID process that was applied to a basic system model and a sample of the Altair lunar lander system. The process streamlines the effort of system architecture modeling and failure analysis that offer system engineers a cost-effective advantage of conducting RID early in the design cycle. Once the model is developed during the design phase, it becomes an iterative process of review and updates that extends throughout the entire lifecycle of a system. MBSE CAD tools enhance the experience of executing the process.

The research also demonstrated that MBSE in general, and Innoslate specifically, is capable of providing an integrated, effective and quantitative means of developing a risk-informed system design using a minimum functionality baseline process. This can be applied to human and robotic spaceflight systems and other systems with similar complexity. As with any engineering analysis tool, engineers should never use the results of an analysis as the sole justification to make a decision, however, the results can be used as a focal point in technical discussions. The Altair RID process also exercised this philosophy. Although Innoslate was the prime MBSE tool used, there are other tools that may provide similar functionality.

Furthermore, the research demonstrated that random distributions could be added to failure probabilities in order to add “noise” to the results, a task that can be laborious, if not impossible, if performed using a portable calculator or spreadsheet. Because of the statistical nature of failure events, adding distributions to component failure rates help ensure that systems still meet criteria over a determined variation.

Prior to adding distributions, the model should be validated first by using fixed values to ensure that the model is setup correctly.

With space systems continuously evolving and becoming more complex every day, engineers are struck with the need to continuously develop tools to address the challenging environment and provide effective solutions to mitigate technical and programmatic risks. Tools and processes are only as effective as the people who use it; therefore, engineers must continuously seek professional development not only for personal growth, but also for the growth of the state-of-the-industry. In a climate of shrinking budgets and increasing technical demands, the MBSE RID Process described in this research proposes a feasible alternative to risk management during system design, development and deployment.

ENDNOTES

[1] Kate Gartside, *The Challenger Disaster,* Directed by James Hawes, Performed by William Hurt, Science Channel, 2013.

[2] Boeing, "Space Shuttle:Backgrounder," *Boeing,* June 2011, http://www.boeing.com/assets/pdf/defense-space/space/hsfe_shuttle/docs/shuttle_overview.pdf (accessed December 03, 2013).

[3] NASA, "Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners," *NASA,* December 2011, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20120001369_2012001000.pdf (accessed December 03, 2013): 1-1.

[4] INCOSE, "Systems Engineering Vision 2020," *INCOSE.* September 2007, http://www.incose.org/ProductsPubs/pdf/SEVision2020_20071003_v2_03.pdf (accessed December 03, 2013): 15.

[5] Vitech Corporation, "Model-Based Systems Engineering," *INCOSE,* September 2011, http://www.incose.org/chesapek/Docs/2011/Presentations/2011_09_17_Model-Based%20SystemsEngineeringPublicSlides.pdf (accessed December 03, 2013): 37.

[6] Ibid.

[7] George Deckert, "Risk Informed Design as Part of the Systems Engineering Process," *NASA,* October 2010, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100037185_2010038869.pdf (accessed December 03, 2013), 2.

[8] Ibid, 3

[9] NASA, "NASA Risk-Informed Decision Making Handbook," *NASA,* April 2010, http://www.hq.nasa.gov/office/codeq/doctree/NASA_SP2010576.pdf (accessed December 03, 2013): 11.

[10] Systems and Proposal Engineering Company, *Leadership Team,* 2013, http://www.specinnovations.com/leadership-team (accessed December 03, 2013).

[11] Steven H. Dam, *Lifecycle Modeling – Application to Architecture Development,* October 2011, http://www.dtic.mil/ndia/2011system/13130_DamThursday.pdf (accessed December 03, 2013): 7.

[12] NASA, "Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners," *NASA,* December 2011, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20120001369_2012001000.pdf (accessed December 03, 2013).

[13] NASA, "NASA Risk-Informed Decision Making Handbook," *NASA,* April 2010, http://www.hq.nasa.gov/office/codeq/doctree/NASA_SP2010576.pdf (accessed December 03, 2013).

[14] Heydorn, Richard P., and Jan W. Railsback, "Safety of Crewed Spaceflight," In *Human Spaceflight: Mission Analysis And Design*, by Wiley J. Larson, Linda K. Pranke, John Connolly and Robert Giffen, The McCraw-Hill Companies, 2000: 193.

[15] Mozilla Developer Network, *Math.random(),* 2013, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random (accessed December 03, 2013).

[16] W3Schools, *JavaScript random() Method,* 2013, http://www.w3schools.com/jsref/jsref_random.asp (accessed December 03, 2013).

[17] NASA, "Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners," *NASA,* December 2011, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20120001369_2012001000.pdf (accessed December 03, 2013).

[18] NASA, "NASA Risk-Informed Decision Making Handbook," *NASA,* April 2010, http://www.hq.nasa.gov/office/codeq/doctree/NASA_SP2010576.pdf (accessed December 03, 2013).

[19] Lauri Hansen, and John Connolly, *The Altair Design And Minimum Functionality Approach,* NASA, International Astronautical Congress, 2008: 2.

[20] Ibid, 3.

[21] Ibid, 4.

[22] John Connolly, Robert L. Bayt, and James H. McMichael, *Human Planetary Spacecraft Design Lessons,* NASA, International Astronautical Congress, 2010: 2.

[23] Lauri Hansen, and John Connolly, *The Altair Design And Minimum Functionality Approach,* NASA, International Astronautical Congress, 2008: 7.

[24] John Connolly, Robert L. Bayt, and James H. McMichael, *Human Planetary Spacecraft Design Lessons,* NASA, International Astronautical Congress, 2010: 2.

[25] Lauri Hansen, and John Connolly, *The Altair Design And Minimum Functionality Approach,* NASA, International Astronautical Congress, 2008: 4.

[26] Jianwen Xiang, Fumio Machida, Kumiko Tadano, Kazuo Yanoo, Wei Sun, and Yoshiharu Maeno, "A Static Analysis of Dynamic Fault Trees with Priority-AND Gates," *NEC,* 2013, http://www.nec.com/en/global/rd/labs/lasd/image/ladc2013-jianwen.pdf.

[27] Jun Ni, Wencheng Tang, and Yan Xing, "A Simple Algebra for Fault Tree Analysis of Static and Dynamic Systems," *IEEE,* October 2013, http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6632939&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel7%2F24%2F4378406%2F06632939.pdf%3Farnumber%3D6632939 (accessed December 03, 2013).

[28] Ken Neubeck, *Practical Reliability Analysis,* Upper Saddel River, NJ: Pearson Education, 2004: 2, 3.

LIST OF REFERENCES

Boeing. "Space Shuttle:Backgrounder." *Boeing.* June 2011. http://www.boeing.com/assets/pdf/defense-space/space/hsfe_shuttle/docs/shuttle_overview.pdf (accessed December 03, 2013).

Connolly, John, Robert L. Bayt, and James H. McMichael. *Human Planetary Spacecraft Design Lessons.* NASA, International Astronautical Congress, 2010.

Dam, Steven H. *Lifecycle Modeling – Application to Architecture Development.* October 2011. http://www.dtic.mil/ndia/2011system/13130_DamThursday.pdf (accessed December 03, 2013).

Deckert, George. "Risk Informed Design as Part of the Systems Engineering Process." *NASA.* October 2010. http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100037185_2010038869.pdf (accessed December 03, 2013).

Gartside, Kate. *The Challenger Disaster.* Directed by James Hawes. Performed by William Hurt. Science Channel, 2013.

Hansen, Lauri, and John Connolly. *The Altair Design And Minimum Functionality Approach.* NASA, International Astronautical Congress, 2008.

Heydorn, Richard P., and Jan W. Railsback. "Safety of Crewed Spaceflight." In *Human Spaceflight: Mission Analysis And Design*, by Wiley J. Larson, Linda K. Pranke, John Connolly and Robert Giffen. The McCraw-Hill Companies, 2000.

INCOSE. "Systems Engineering Vision 2020." *INCOSE.* September 2007. http://www.incose.org/ProductsPubs/pdf/SEVision2020_20071003_v2_03.pdf (accessed December 03, 2013).

Mozilla Developer Network. *Math.random().* 2013. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random (accessed December 03, 2013).

Neubeck, Ken. *Practical Reliability Analysis.* Upper Saddel River, NJ: Pearson Education, 2004.

NASA. "NASA Risk-Informed Decision Making Handbook." *NASA.* April 2010. http://www.hq.nasa.gov/office/codeq/doctree/NASA_SP2010576.pdf (accessed December 03, 2013).

NASA. "Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners." *NASA.* December 2011. http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20120001369_2012001000.pdf (accessed December 03, 2013).

Ni, Jun, Wencheng Tang, and Yan Xing. "A Simple Algebra for Fault Tree Analysis of Static and Dynamic Systems." *IEEE.* October 2013. http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6632939&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel7%2F24%2F4378406%2F06632939.pdf%3Farnumber%3D6632939 (accessed December 03, 2013).

Systems and Proposal Engineering Company. *Leadership Team.* 2013. http://www.specinnovations.com/leadership-team (accessed December 03, 2013).

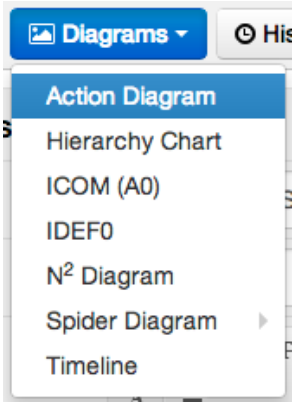
Vitech Corporation. "Model-Based Systems Engineering." *INCOSE.* September 2011. http://www.incose.org/chesapek/Docs/2011/Presentations/2011_09_17_Model-Based%20SystemsEngineeringPublicSlides.pdf (accessed December 03, 2013).

W3Schools. *JavaScript random() Method.* 2013. http://www.w3schools.com/jsref/jsref_random.asp (accessed December 03, 2013).

Xiang, Jianwen, Fumio Machida, Kumiko Tadano, Kazuo Yanoo, Wei Sun, and Yoshiharu Maeno. "A Static Analysis of Dynamic Fault Trees with Priority-AND Gates." *NEC.* 2013. http://www.nec.com/en/global/rd/labs/lasd/image/ladc2013-jianwen.pdf.

APPENDIX I: INNOSLATE MODELING

- Figure 12 depicts a graphic overview of how to develop a physical architecture using “Assets” in Innoslate. In the same manner, a functional architecture can be developed, however, “Action” entities are used instead of “Assets”.
- Figure 13 & 14 show a “Database” view of all the generic failure event entities developed in this research paper, which comprises the entire failure event architecture.
- Figure 15 depicts a breakdown structure of the System f0 failure architecture down to Component a1 in an “Action Diagram” view (Similar to Figure 3). When creating the ‘OR’ action entities that implement SimScript routines, it is best to create it in the “Action Diagram” view (See graphic below).

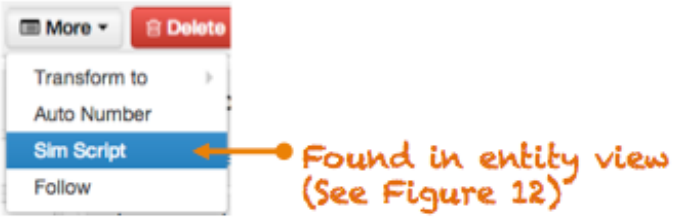


- Figure 16 represent “Failure” and “No Failure” time path entities. These are important to include in every failure architecture model where a logic gate exists. While a failure adds no time to the simulation, a no-failure adds time (1 hour). Simulation iterations that fully pass will accumulate as the longest time in the Monte Carlo bar graph.

APPENDIX I: INNOSLATE MODELING

Innoslate Modeling Figure Notes

- Figure 17 depicts the Failure Event Architecture Hierarchy View for System f0. Although difficult to view in this document, it can provide a general sense of the structure.
- Table 4 & 5 show all the SimScript details for the failure gate and probability entities that require it.



  - User-Defined Variables (In Green):
    - var probOfSuccess = #.#; --- Set Success Rate
    - globals.put("refdesFail", "true"); --- Set Reference Designation Variable
    - globals.get("refdesFail") --- Retrieves Variable Set
    - var probOfSuccess = Math.random()*(max-min)+min; --- Set Random Distribution with Minimum and Maximum Range Values
    - **Warning! Unless the user is savvy with JavaScript, it is recommended to leave all other lines as-is unless adding more global variables or updating the logic operators**
  - AND Logic Operator = &&
  - OR Logic Operator = ||
  - AND & OR logic operators can be combined within a single script, but may be difficult to track externally within the model since scripts are buried within the entities

APPENDIX I: INNOSLATE MODELING

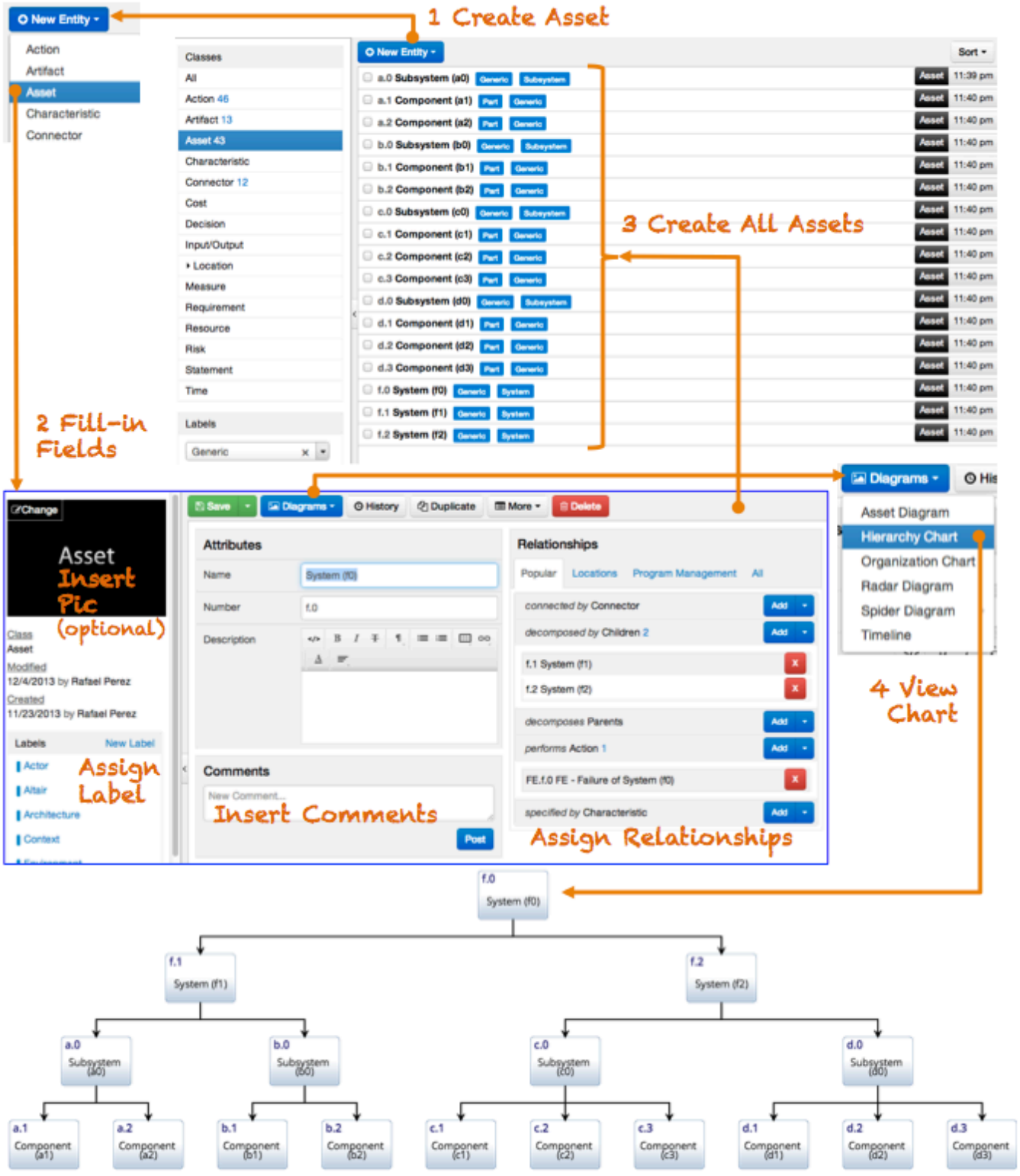


Figure 12. Developing Assets Using Innoslate

APPENDIX I: INNOSLATE MODELING

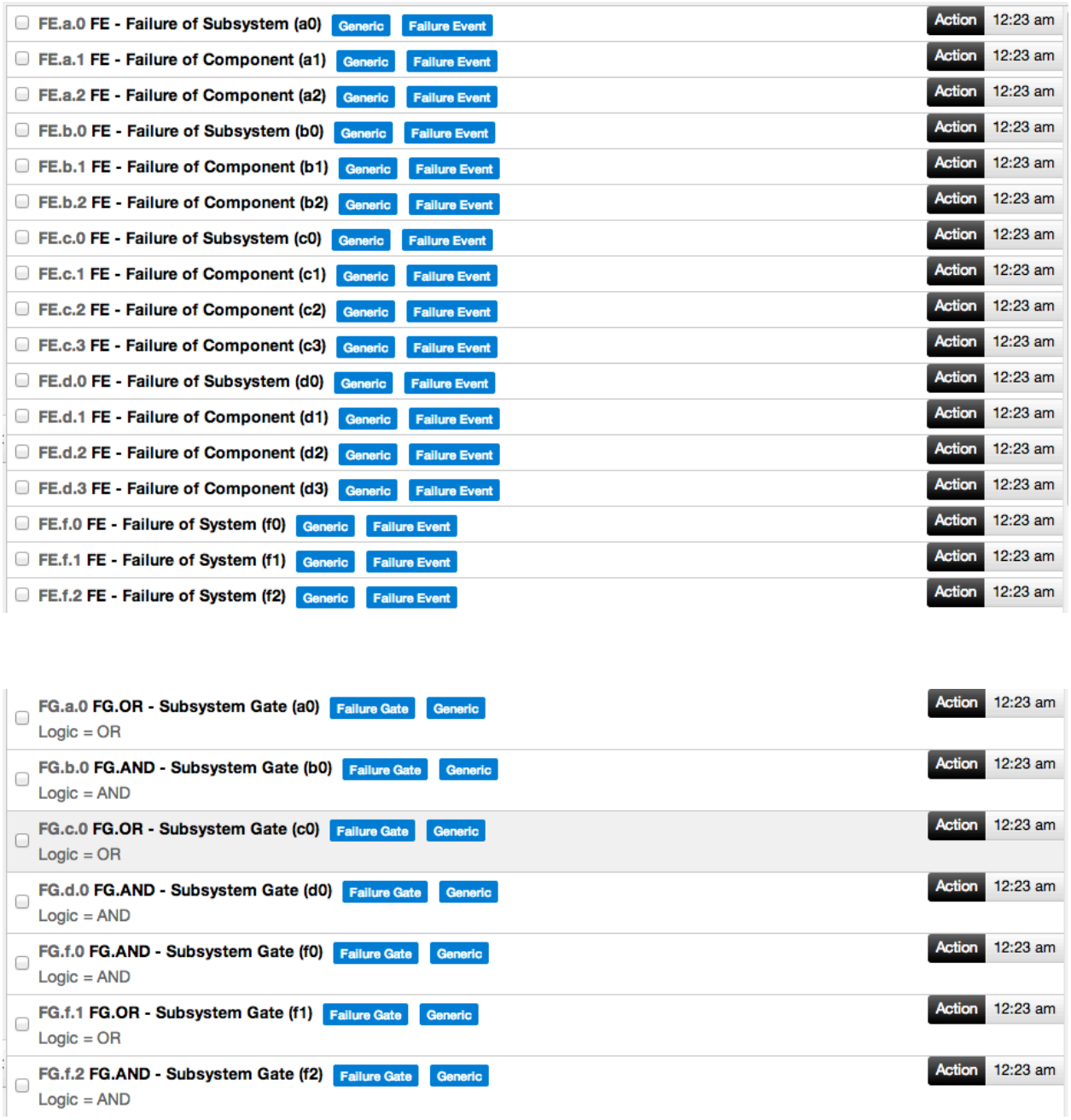


Figure 13. Innoslate Failure Event & Gate Entities

APPENDIX I: INNOSLATE MODELING

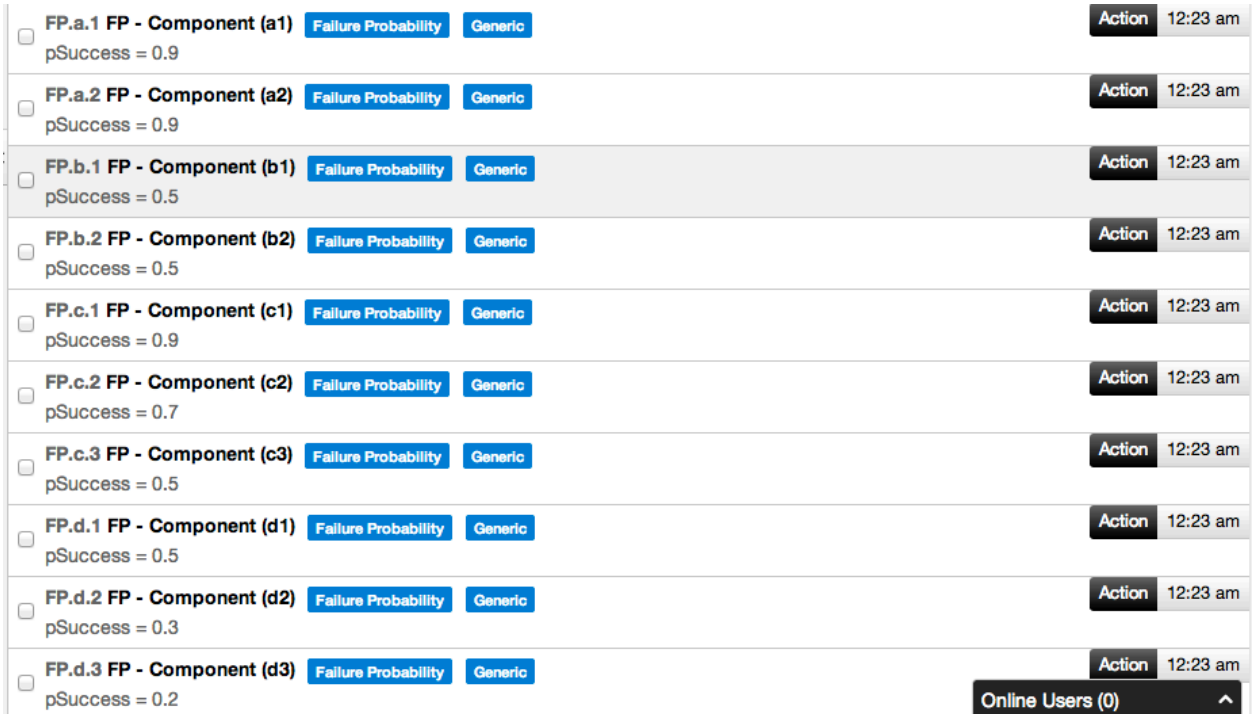


Figure 14. Innoslate Failure Probability Entities

APPENDIX I: INNOSLATE MODELING

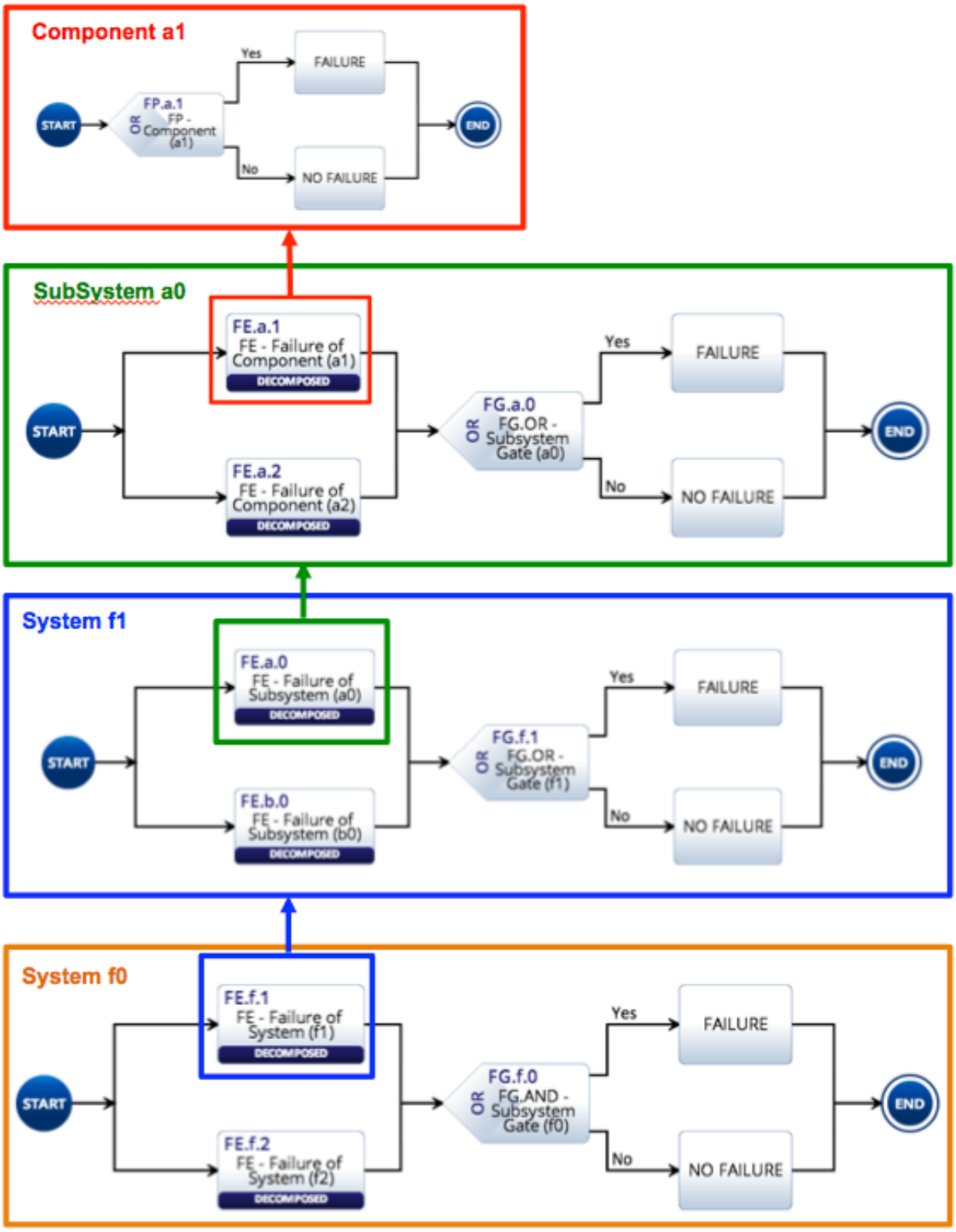


Figure 15. Developing Failure Architecture Using Innoslate

APPENDIX I: INNOSLATE MODELING

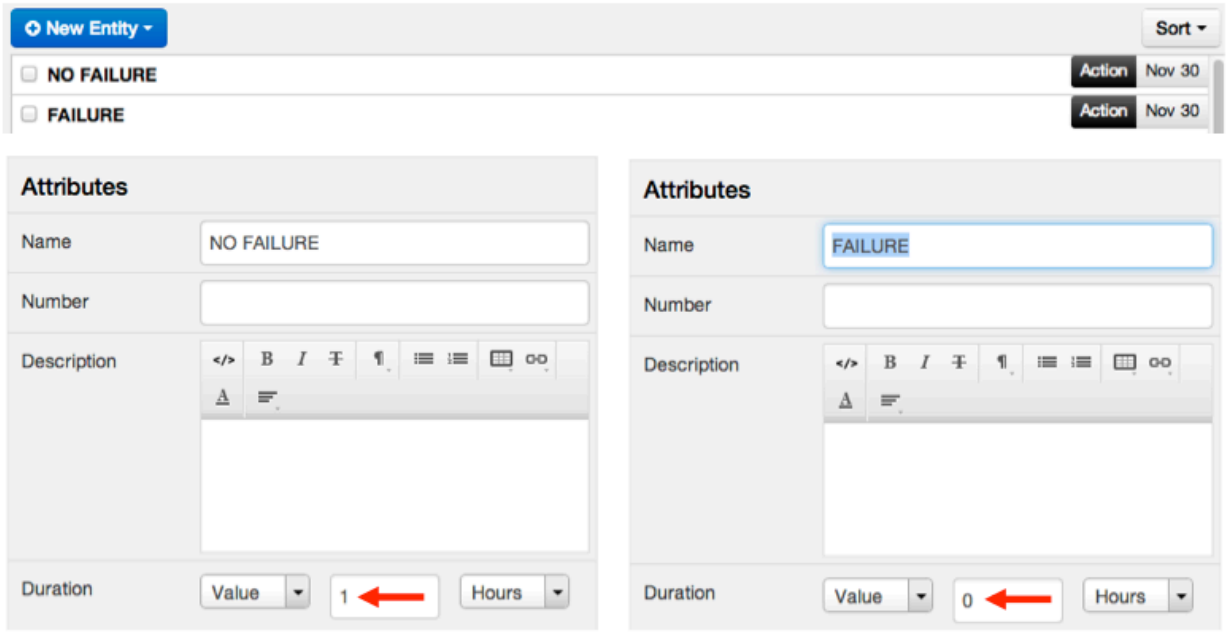


Figure 16. Failure/No-Failure Time Path Entities

APPENDIX I: INNOSLATE MODELING

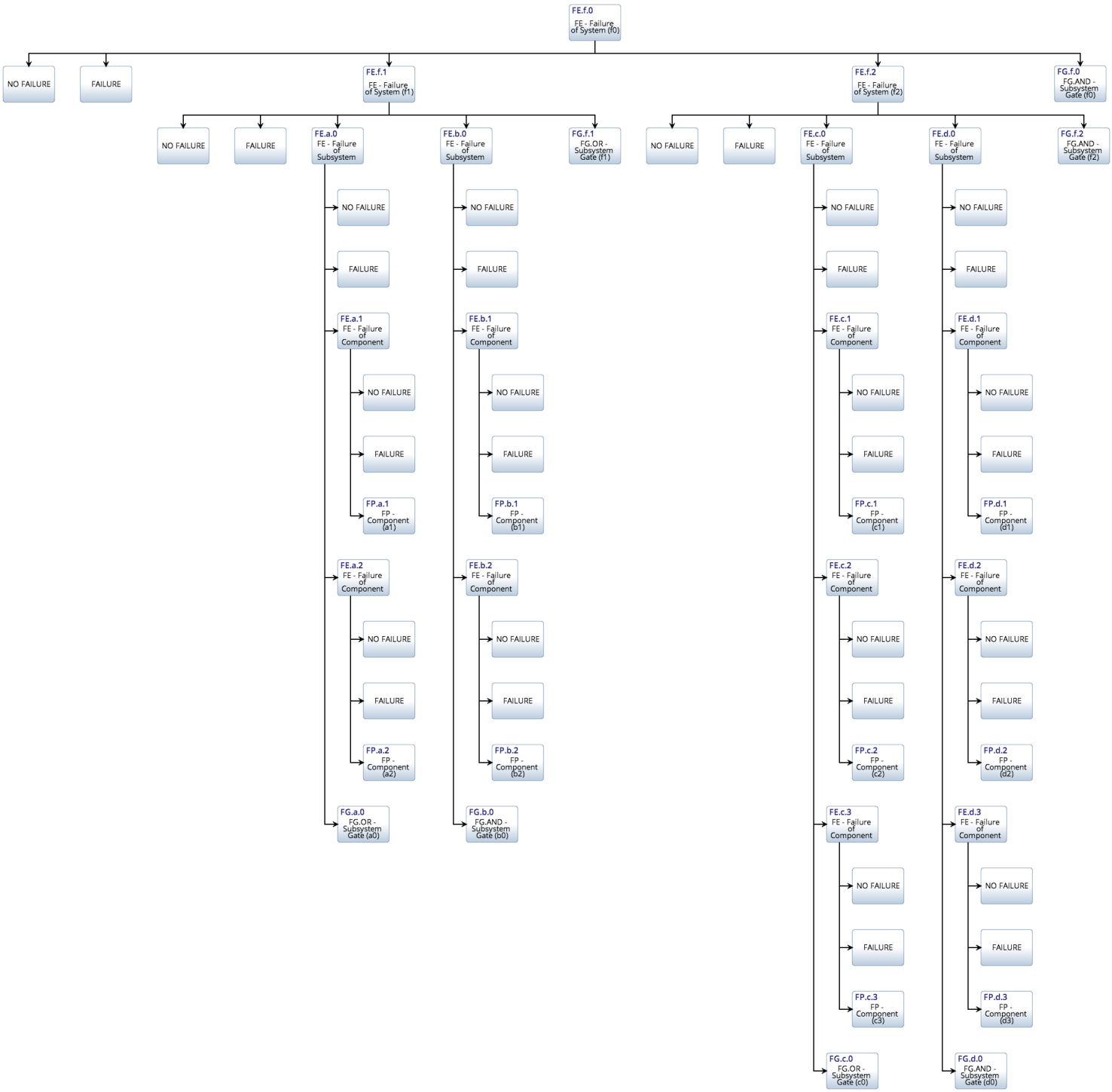


Figure 17. System f0 Failure Event Architecture Hierarchy View

APPENDIX I: INNOSLATE MODELING

FG.OR - Subsystem Gate (a0)	FG.AND - Subsystem Gate (b0)

function onEnd()
{
 	
if(globals.get("a1Fail") ||
globals.get("a2Fail"))
 	
{
globals.put("a0Fail", "true");
return "Yes";
}
 	
else
{
return "No";
}
 	
}
	function onEnd()
	{
	
	if(globals.get("b1Fail") &&
	globals.get("b2Fail"))
	
	{
	globals.put("b0Fail", "true");
	return "Yes";
	}
	
	else
	{
	return "No";
	}
	
	}

FG.OR - Subsystem Gate (c0)	FG.AND - Subsystem Gate (d0)

function onEnd()
{
 	
if(globals.get("c1Fail") ||
globals.get("c2Fail") ||
globals.get("c3Fail"))
 	
{
globals.put("c0Fail", "true");
return "Yes";
}
 	
else
{
return "No";
}
 	
}
	function onEnd()
	{
	
	if(globals.get("d1Fail") &&
	globals.get("d2Fail") &&
	globals.get("d3Fail"))
	
	{
	globals.put("d0Fail", "true");
	return "Yes";
	}
	
	else
	{
	return "No";
	}
	
	}

Table 4A. Failure Gate Entity SimScript Summary A

APPENDIX I: INNOSLATE MODELING

FG.OR - Subsystem Gate (f1)	FG.AND - Subsystem Gate (f2)	FG.AND - Subsystem Gate (f0)

function onEnd()
{
 	
if(globals.get("a0Fail") ||
globals.get("b0Fail"))
 	
{
globals.put("f1Fail", "true");
return "Yes";
}
 	
else
{
return "No";
}
 	
}
	function onEnd()
	{
	
	if(globals.get("c0Fail") &&
	globals.get("d0Fail"))
	
	{
	globals.put("f2Fail", "true");
	return "Yes";
	}
	
	else
	{
	return "No";
	}
	
	}
		function onEnd()
		{
		
		if(globals.get("f1Fail") &&
		globals.get("f2Fail"))
		
		{
		globals.put("f0Fail", "true");
		return "Yes";
		}
		
		else
		{
		return "No";
		}
		
		}

Table 4B. Failure Gate Entity SimScript Summary B

APPENDIX I: INNOSLATE MODELING

FP - Component (a1)	FP - Component (a2)

function onEnd()
{
 	
var probOfSuccess = 0.9;
var probOfFailure = 1 - probOfSuccess;
var randomNumber = Math.random();
 	
if(randomNumber < probOfFailure)
{
globals.put("a1Fail", "true");
return "Yes";
}
 	
else
{
return "No";
}
 	
}
	function onEnd()
	{
	
	var probOfSuccess = 0.9;
	var probOfFailure = 1 - probOfSuccess;
	var randomNumber = Math.random();
	
	if(randomNumber < probOfFailure)
	{
	globals.put("a2Fail", "true");
	return "Yes";
	}
	
	else
	{
	return "No";
	}
	
	}

FP - Component (b1)	FP - Component (b2)

function onEnd()
{
 	
var probOfSuccess = 0.5;
var probOfFailure = 1 - probOfSuccess;
var randomNumber = Math.random();
 	
if(randomNumber < probOfFailure)
{
globals.put("b1Fail", "true");
return "Yes";
}
 	
else
{
return "No";
}
 	
}
	function onEnd()
	{
	
	var probOfSuccess = 0.5;
	var probOfFailure = 1 - probOfSuccess;
	var randomNumber = Math.random();
	
	if(randomNumber < probOfFailure)
	{
	globals.put("b2Fail", "true");
	return "Yes";
	}
	
	else
	{
	return "No";
	}
	
	}

Table 5A. Component Failure Probability Entity SimScript Summary A

APPENDIX I: INNOSLATE MODELING

FP - Component (c1)	FP - Component (c2)	FP - Component (c3)

function onEnd()
{
 	
var probOfSuccess = 0.9;
var probOfFailure = 1 - probOfSuccess;
var randomNumber = Math.random();
 	
if(randomNumber < probOfFailure)
{
globals.put("c1Fail", "true");
return "Yes";
}
 	
else
{
return "No";
}
 	
}
	function onEnd()
	{
	
	var probOfSuccess = 0.7;
	var probOfFailure = 1 - probOfSuccess;
	var randomNumber = Math.random();
	
	if(randomNumber < probOfFailure)
	{
	globals.put("c2Fail", "true");
	return "Yes";
	}
	
	else
	{
	return "No";
	}
	
	}
		function onEnd()
		{
		
		var probOfSuccess = 0.5;
		var probOfFailure = 1 - probOfSuccess;
		var randomNumber = Math.random();
		
		if(randomNumber < probOfFailure)
		{
		globals.put("c3Fail", "true");
		return "Yes";
		}
		
		else
		{
		return "No";
		}
		
		}

FP - Component (d1)	FP - Component (d2)	FP - Component (d3)

function onEnd()
{
 	
var probOfSuccess = 0.5;
var probOfFailure = 1 - probOfSuccess;
var randomNumber = Math.random();
 	
if(randomNumber < probOfFailure)
{
globals.put("d1Fail", "true");
return "Yes";
}
 	
else
{
return "No";
}
 	
}
	function onEnd()
	{
	
	var probOfSuccess = 0.3;
	var probOfFailure = 1 - probOfSuccess;
	var randomNumber = Math.random();
	
	if(randomNumber < probOfFailure)
	{
	globals.put("d2Fail", "true");
	return "Yes";
	}
	
	else
	{
	return "No";
	}
	
	}
		function onEnd()
		{
		
		var probOfSuccess = 0.2;
		var probOfFailure = 1 - probOfSuccess;
		var randomNumber = Math.random();
		
		if(randomNumber < probOfFailure)
		{
		globals.put("d3Fail", "true");
		return "Yes";
		}
		
		else
		{
		return "No";
		}
		
		}

Table 5B. Component Failure Probability Entity SimScript Summary B

APPENDIX I: INNOSLATE MODELING

Innoslate Modeling General Notes

1. Unable to duplicate an action entity within the same class (i.e. duplicating an OR action entity does not duplicate to an OR entity, but rather a basic action entity) or to change the action entity type
2. For security reasons, Innoslate denies read and write access to its program scripts. Users only have access to global variable sets. There are no limits to how many variables can be set and are reset after every simulation.
3. Innoslate limits to 200 entities in a model. Be sure to clear out un-needed simulation runs to keep the model clean and free.
4. Avoid using the same failure entity twice in the same architecture chain. An anomaly was observed when a Monte Carlo simulation was performed and an infinite loop was detected. This will force the user to eventually delete and recreate the entity, but not for its related entities. The loop did not happen for every case.
5. Minor issues have been observed with the type of browser used, but this is typically resolved either with an update to the browser by the user or the Innoslate software by the SPEC team
6. Although Innoslate uses cloud-computing, slower performance was noticed after running many Monte Carlo simulations from the personal computer processor and hard drive storage also gets eaten away

APPENDIX I: INNOSLATE MODELING

Innoslate Links

- https://innoslate.com/
- https://innoslate.com/features/tour/overview/
- https://innoslate.com/help/
- https://innoslate.com/wp-content/uploads/2012/08/Using-Innoslate-for-Operations-and-Support-OS.pdf

JavaScript Links

- http://www.w3schools.com/js/

For Read-Only access to the system architecture model described in this paper, please sign-up for an Innoslate account at https://innoslate.com/ and send an e-mail to the author at rmperez88@gmail.com. Be sure to include the e-mail used in the Innoslate account.

APPENDIX II: INNOSLATE MONTE CARLO SIMULATION

Innoslate Monte Carlo Simulation Figure Notes

- Figure 18 depicts a graphic overview of how to run Monte Carlo simulations in Innoslate. The computation time varies depending on the complexity of the model and it is performed in the Cloud. The bar graph at the end summarizes the iterations that failed or succeeded over 100 counts. Each success iteration accumulates in the single, extreme right bar, while each failure iteration accumulates on any of the bars on the left. Remember that the “Failure” entities add no time to the simulation while the “No Failure” entities add time (In this model, +1 Hour). The results in Figure 18 shows that there were 92% (0.92) success iterations over 100 passes, while there were 8% (0.08) failure iterations over 100 passes. These values represent the failure rate of that particular system.

- Figure 19 shows how to retrieve Monte Carlo simulation artifacts from the model after the simulation has been closed out. Every simulation performed in Innoslate automatically creates an artifact in the model. By default, artifacts automatically get assigned a name with a time stamp (i.e. “Monte Carlo 2013-12-06 05:12:15 AM”). It is recommended to rename the artifact to something more logical and flexible to allow for easier tracking and updating as the model grows.

APPENDIX II: INNOSLATE MONTE CARLO SIMULATION

- Figure 20 shows an optional way of consolidating the failure bars of the Monte Carlo simulations. For the generic model in this research paper, not many additional failure bars were generated, however, as the model grows, much more failure bars are likely to show up. Each failure bar represents a different time in the sequence of failure events when the system failed, therefore, longer failure event chains will show more failure bars and the graph may become difficult to read. Adding a time spacer element with a long duration to **only** the last "No Failure" entity in the failure event sequence will push the success bar to a more extreme time category. This results in the consolidation of the smaller failure time categories to the left because the graph will not support very large increments.

APPENDIX II: INNOSLATE MONTE CARLO SIMULATION

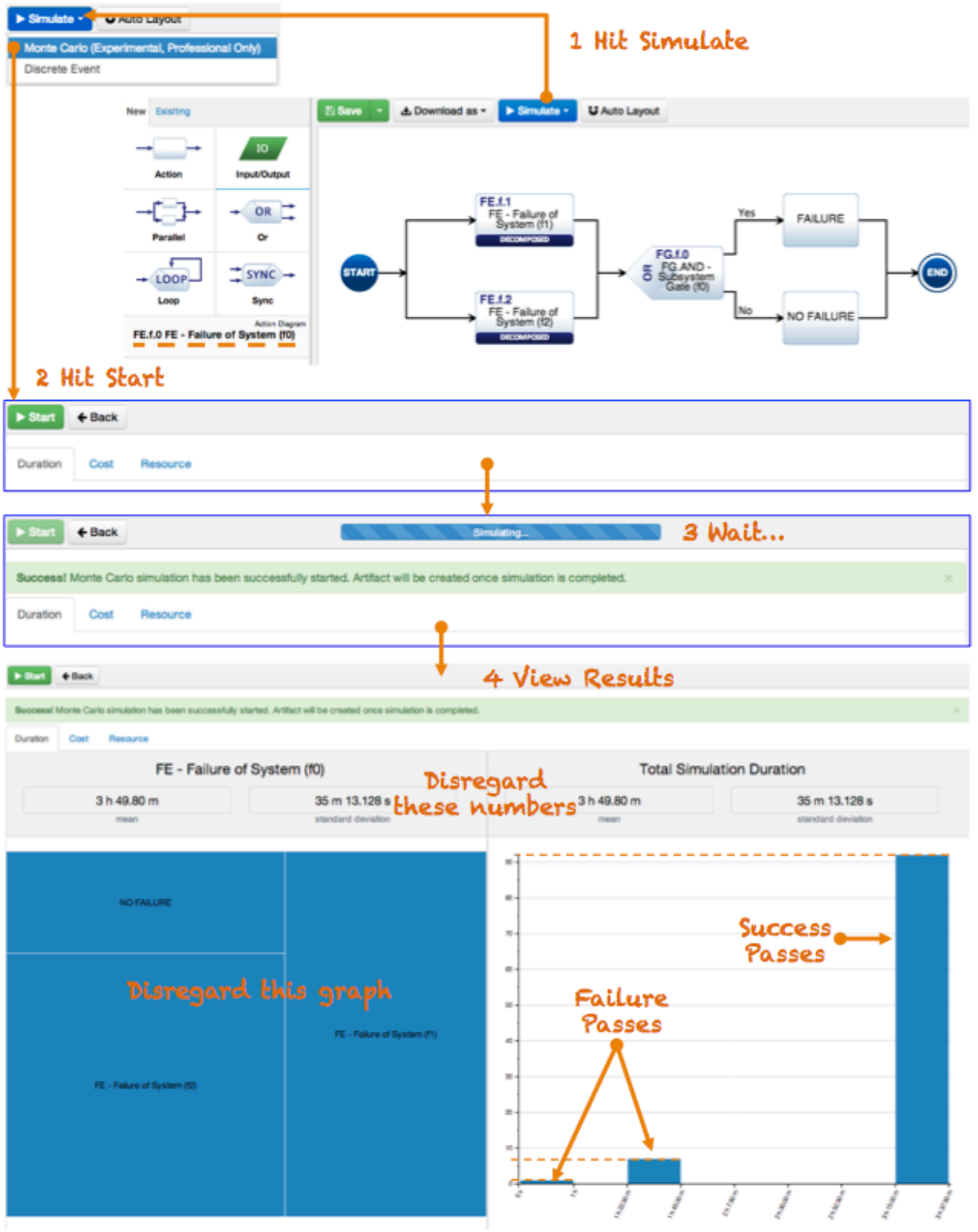


Figure 18. Running Monte Carlo Simulations Using Innoslate

APPENDIX II: INNOSLATE MONTE CARLO SIMULATION

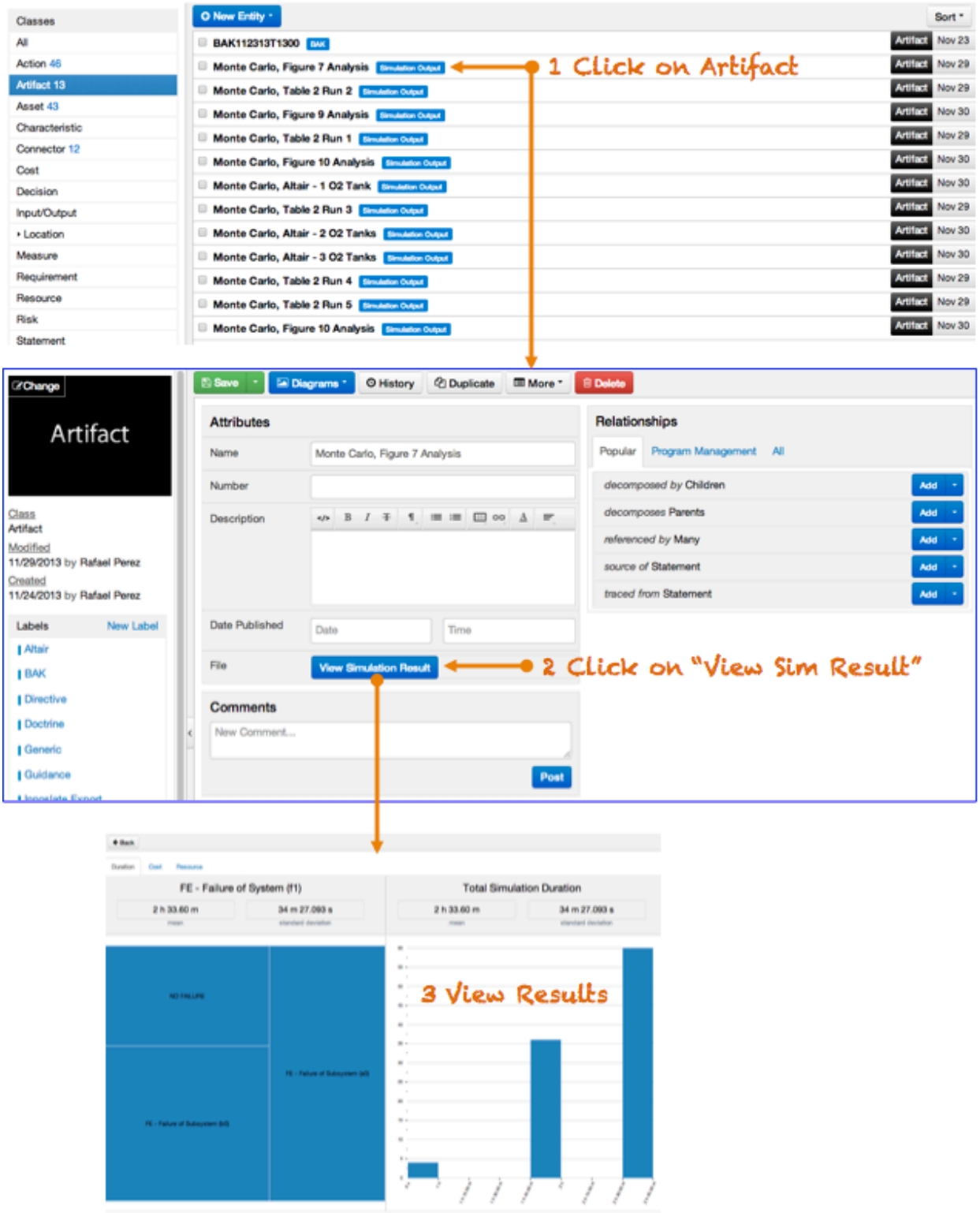


Figure 19. Retrieving Monte Carlo Simulation Artifacts

APPENDIX II: INNOSLATE MONTE CARLO SIMULATION

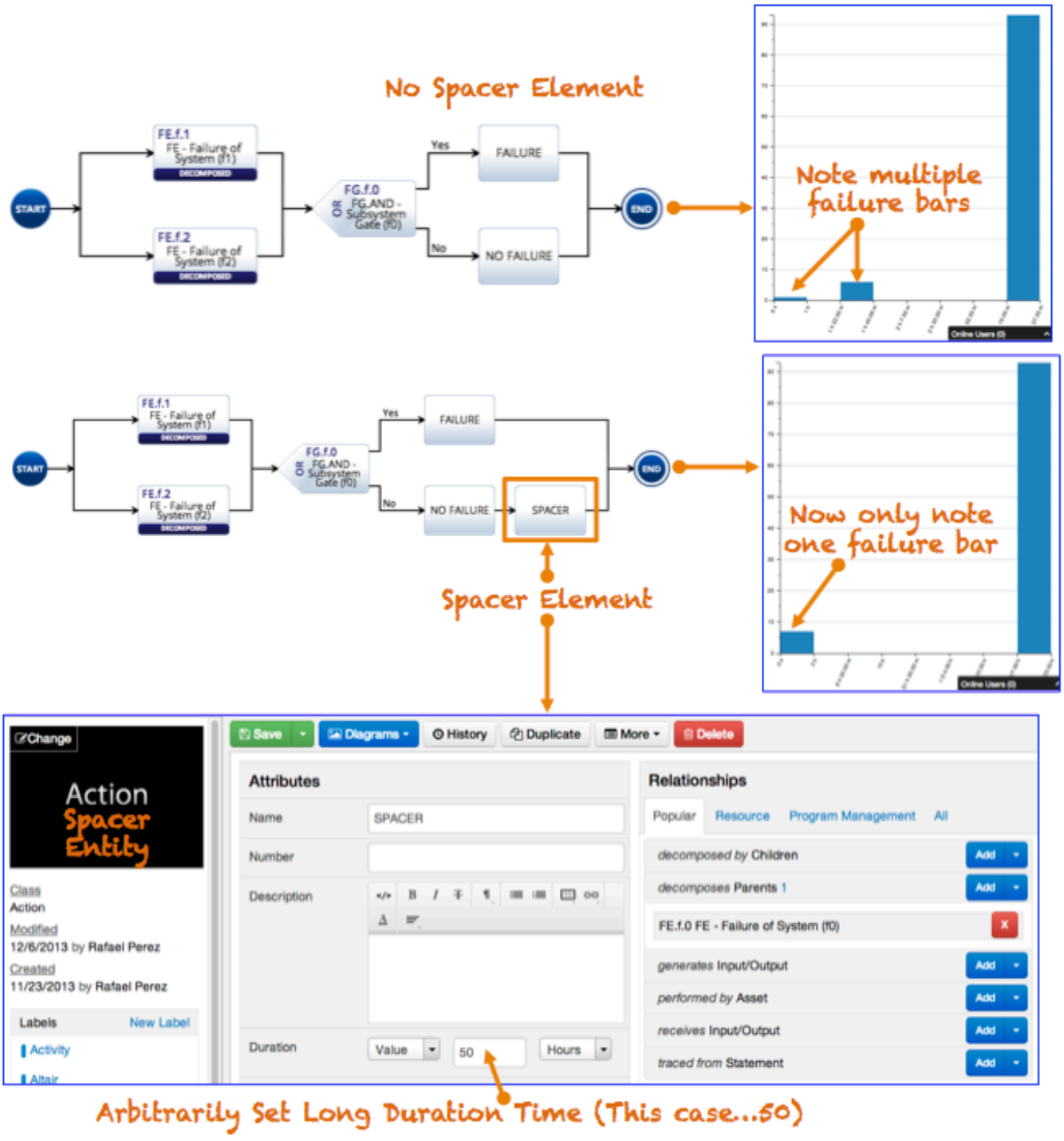


Figure 20. System f0 Spacer Time Element Comparison

APPENDIX II: INNOSLATE MONTE CARLO SIMULATION

Innoslate Monte Carlo Simulation General Notes

- Innoslate limits the iteration count to 100 maximum. They are looking to provide more flexible iteration options for Monte Carlo simulations. For now, it's experimental.
- Innoslate Monte Carlo bar graphs that do not provide exact accumulation numbers requiring the user to follow the top edge of the bar to the axis. They are looking to provide a hover-over feature to allow users to view the exact numbers in the next version.
- When viewing the bar graph for the Monte Carlo analysis, the chat box icon sometimes blocks the x-axis marking depending on the size and resolution of the computer display. Smaller displays will tend to block the axis. Users can zoom out using the browser function to clear it away.
- Users can upload any type of electronic file into "Artifacts". Innoslate will hold it as data repository item. Artifacts are treated as any other entity in the model, but with the additional upload feature, therefore, users can create artifacts just like other entities and also create relationships to artifacts, a key benefit. Back-up project files for Innoslate can also be held in artifacts, which comes as a '.xml' file.

APPENDIX II: INNOSLATE MONTE CARLO SIMULATION

- When running Monte Carlo simulations, the simulation may sometimes seem to hang-up in the process. If the user waited for several minutes after the simulation was declared “Success!” with no response and the simulation icon below is still active, then there may be a chance that the simulation already completed and generated an artifact. Figure 19 shows how to retrieve artifacts. Keep in mind that complex models will require more time to simulate. If the user decides to look for the artifact during a simulation, then the simulation will stop and if not completed, it will not generate an artifact. The user will be forced to run the simulation again.

APPENDIX III: ALTAIR LUNAR LANDER DESIGN CHANGES THROUGH THE ANALYSIS CYCLES

	DAC-1	DAC-2	DAC-3	RAC-1&2	DAC-4
Theme:	♦ Minimally functional vehicle	♦ Improve LOC risk posture	♦ Improve LOM risk posture	♦ Mature requirements and assess gaps in Lander design	♦ Requirement incorporation and maturation
Major Upgrades, Additions, Findings:	♦ N/A	♦ Added abort capabilities ♦ Updated descent trajectory / delta v ♦ Redundant suit loop compressor, O2 sensors, and O2 tanks ♦ Selected redundancy within fuel cell stack, battery, and PDU ♦ Emergency communications system ♦ Second IMU, star tracker, and b/u radar electronics ♦ Valving, plumbing, wiring upgrades to all sub-systems	♦ Third flight computer ♦ Third IMU, docking camera backup to star tracker, and IMU-less (manual) mode ♦ Manual circuit breakers in PDUs ♦ Added 3rd life support O2 tank ♦ Valving, plumbing, wiring upgrades to all sub-systems ♦ Changed DM and AL primary structures to composites ♦ Changed to autogenous pressurization on DM	♦ Reviewed and matured the Altair-allocated CARD requirements, C3I requirements, and HSIR requirements ♦ Improved fidelity and expanded the T/O list ♦ Matured 6 major variants of the DM structure and tank configurations	♦ Re-close vehicle design based upon RAC1&2 requirement acceptances ♦ Distributed avionics replacing centralized avionics ♦ Landing loads assessment and resizing
Expected Vehicle Mass:	45,000 kg (no T/O assessments)	45,002 kg (no T/O assessments) (p0804-D)	45,524 kg (no T/O assessments) (p0905-A)	45,720 kg Base 7,558 kg Threats (p0905-C)	44, 900 kg Base 4,400 kg Threats (p1006-D)
LOC:	1 in 6	1 in 196	1 in 256	1 in 277	In work
LOM:	Not assessed	1 in 4	1 in 22	1 in 22	In work

Notes:

- Taken from the following source:

  John Connolly, Robert L. Bayt, and James H. McMichael, *Human Planetary Spacecraft Design Lessons,* NASA, International Astronautical Congress, 2010: 2.

APPENDIX IV: ALTAIR LUNAR LANDER GENERIC FAILURE RATES

Name	Rate	Failure Type
AirInlet	3.00E-06	Rate
AirPressTransd	7.00E-06	Rate
AM-Fire	3.00E-06	Rate
AmmoniaTank	1.00E-05	Rate
Antenna Switch	3.00E-05	Demand
AtmoRevitalization	4.00E-06	Rate
AudioInterface	4.00E-06	Rate
Battery	1.00E-05	Rate
CEVPyro	2.00E-06	Demand
CheckValve	1.00E-05	Rate
CockpitKeypad	3.00E-06	Demand
ColdPlate	3.00E-07	Rate
CommHeadset	8.00E-07	Rate
CondensStorTank	7.00E-06	Rate
ControlPanelPress	9.00E-06	Rate
Coupler	2.00E-06	Rate
DepressRepressPump	3.00E-06	Rate
Desiccant	1.00E-05	Rate
DiffPressSensor	8.00E-07	Rate
Diplexer	1.00E-08	Rate
Dock	3.00E-03	Demand
EmergencyLight	9.00E-07	Rate
EngineCutoffSens	4.00E-04	Demand
EthernetHub	2.00E-05	Rate
ExternalLight	3.00E-05	Rate
ExternalVideoSwitch	3.00E-05	Rate
FilterRCS	1.00E-07	Rate
FlowMeter	1.00E-06	Rate
FluidCheckValve	3.00E-07	Rate
FuelCell	5.00E-05	Rate
GasLiqRegulator	1.00E-06	Rate
GasTankO2	1.00E-07	Rate
GasTrap	2.00E-06	Rate
Heater	1.00E-06	Rate
HeatExchanger	1.00E-06	Rate
HeatPipe	4.00E-06	Rate
HighLoadHeatExc	4.00E-06	Rate
IMU	5.00E-06	Rate
InterfaceBox	4.00E-07	Rate
InterfaceBoxI	6.00E-06	Rate
InternalESSMDM	3.00E-05	Rate
InternalWireHarness	2.00E-07	Rate
IsoValveRCS	2.00E-06	Rate
ISSFilter	5.00E-09	Rate
ISSITCSPump	6.00E-06	Rate
ISSTempSensor	3.00E-08	Rate
KuBandTransc	3.00E-05	Rate
Landing	2.00E-04	Demand
LCDDispCont	5.00E-07	Rate
LCGPump	6.00E-06	Rate
LIDAR	2.00E-05	Rate
LineExternal	2.00E-09	Rate
LineInternal	1.00E-09	Rate
LiqLevelSensor	8.00E-07	Rate
LiqMotorIso	5.00E-07	Rate

Name	Rate	Failure Type
Low Noise Amp	1.00E-06	Rate
ManualValve	1.00E-08	Rate
MDM16Card	6.00E-05	Rate
MDMPowerSupply	1.00E-07	Rate
MDMPowerSupplyI	5.00E-07	Rate
MixValve3Way	1.00E-05	Rate
MMAM-1	3.00E-07	Rate
MMDM-1	3.00E-07	Rate
MMDODDM-1	3.00E-07	Rate
MMHM-1	3.00E-07	Rate
MMODAM-1	3.00E-07	Rate
MMODHM-1	3.00E-07	Rate
OMSEngine	1.00E-05	Demand
OMSGimbal	3.00E-08	Demand
Orifice	1.00E-06	Rate
OverBoardVent	1.00E-08	Rate
ParabAntenna	4.00E-07	Rate
PassiveRadiator	4.00E-08	Rate
PressRegulator	3.00E-06	Rate
PressTransducer	2.00E-06	Rate
PressureGage	1.00E-05	Rate
PumpFlowCont	5.00E-06	Rate
PumpModule	2.00E-05	Rate
PumpPackage	3.00E-05	Rate
PVRadiator	7.00E-06	Rate
Pwr Amp	8.00E-06	Rate
RackFlowCont	1.00E-05	Rate
Radio Filter	1.00E-07	Rate
RCSReliefValve	1.00E-04	Demand
RCSThruster	5.00E-04	Demand
RelayContact	4.00E-08	Rate
RelayContactI	1.00E-06	Rate
ReliefValve	1.00E-06	Rate
RL10mission	7.00E-03	Demand
RL10Run	4.00E-06	Rate
RL10Start	3.00E-04	Demand
RotatHandCont	6.00E-07	Rate
SmokeDetector	1.00E-06	Rate
Software-1	1.00E-04	Demand
SolenoidRelay	2.00E-05	Rate
SolenoidRelayDemand	5.00E-04	Demand
StarTracker	1.00E-05	Rate
Star Tracker Mech	4.00E-05	Rate
Structure	0.00E+00	
TankHeater	5.00E-06	Rate
TempSensor	2.00E-08	Rate
TransHandCont	7.00E-07	Rate
Transponder	8.00E-06	Rate
UHFAntenna	1.00E-07	Rate
UHFRadio	2.00E-05	Rate
ValveSolenoid	3.00E-06	Rate
VolitileOrganicAnaly	3.00E-05	Rate
Waste Baggies	1.00E-06	Rate
Waste management collector can	1.00E-06	Rate
Water storage tank	1.00E-06	Rate

Notes:

- Rate values are arbitrary and unitless
- Table provided courtesy of Mr. Randolph Rust at NASA

ACRONYMS

AM	Ascent Module
CAD	Computer-Aided Design
CONOPS	Concept of Operations
DAC	Design Analysis Cycle
DM	Descent Module
DM2	DoDAF Metamodel 2.0
DoDAF	Department of Defense Architecture Framework
DTIC	Defense Technical Information Center
EDSA	Earth Departure Stage/Altair Adapter
ECLSS	Environmental Control and Life Support System
FMECA	Failure Modes and Effects Criticality Analyses
FE	Failure Event
FG	Failure Gate
FM	Failure Mode
FP	Failure Probability
GCR	Galactic Cosmic Radiation
IAC	International Astronautical Congress
INCOSE	International Council on Systems Engineering
JCIDS	Joint Capabilities Integration and Development System
LDAC	Lander Design Analysis Cycle
LL	Lunar Lander
LML	Lifecycle Modeling Language
MBSE	Model-Based Systems Engineering
MMOD	Micrometeoroids & Orbital Debris (MMOD)
MODAF	Ministry of Defence Architecture Framework
NASA	National Aeronautics and Space Administration
NEC	Nippon Electric Company

pLOC	Probability of Loss of Crew
pLOM	Probability of Loss of Mission
pLOV	Probability of Loss of Vehicle
pLOS	Probability of Loss of System
PRA	Probabilistic Risk Assessment
RAC	Requirements Analysis Cycles
RAM	Reliability, Availability & Maintainability
RID	Risk-Informed Design
RIDM	Risk-Informed Decision Making
RM	Risk Management
SAPHIRE	Systems Analysis Programs for Hands-on Integrated Reliability Evaluations
SPEC	Systems and Proposal Engineering Company
SysML	System Modeling Language
WBS	Work Breakdown Structure