

REPORT DOCUMENTATION PAGE					<i>Form Approved OMB No. 0704-0188</i>	
<small>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</small>						
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY)		2. REPORT TYPE			3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S)				5d. PROJECT NUMBER		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)					8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)					10. SPONSOR/MONITOR'S ACRONYM(S)	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT						
15. SUBJECT TERMS						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (Include area code)	



OPERATIONS RESEARCH CENTER

United States Military Academy
West Point, New York 10996

June 2015

Illuminating Tradespace Decisions Using Efficient Experimental Space-Filling Designs for the Engineered Resilient System Architecture

Prepared By

LTC Alex MacCalman, PhD

2LT Hyangshim Kwak

Ms. Mary McDonald

Mr. Steve Upton

CDT Coleman Grider

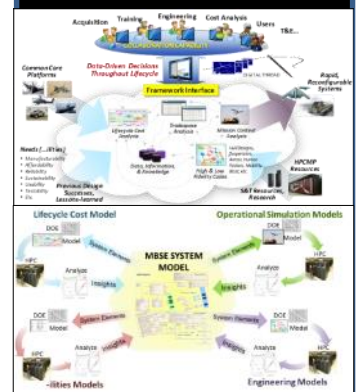
CDT Robert Hill

CDT Hunter Wood

LTC Paul Evangelista, PhD

Prepared For

US Army Engineered Research Development Center
Vicksburg, Mississippi



DSE-R-1501

Approved for public release; distribution is unlimited.

Contents

Executive Summary.....	5
1.Introduction	8
2.Technical Report Organization	9
3.Background	10
3.1. Process versus Data Driven Approach	10
3.2. Engineered Resilient System Architecture.....	12
4.Research Contributions	14
5.Representative Use Case Description	15
5.1. Agent-Based Model Overview	15
5.2. Simulation Scenarios.....	18
5.3. Model Inputs.....	19
5.4. Model Outputs.....	21
6.Model Based System Engineering Approach	22
6.1. System Modeling Language (SysML).....	23
6.2. External Model Integration	31
7.Building Statistical Metamodels using Simulation Experimental Designs.....	34
7.1. Statistical Design of Experiments Introduction.....	35
7.2. Understanding Complex Behavior	36
7.2.1. Design Drivers	38
7.2.2. Synergies/Interactions	39
7.2.3. Diminishing or Increasing Rates of Change.....	40
7.2.4. Identifying Thresholds with Partition Trees.....	41
7.3. Predicting Simulation Model Outputs	42
7.3.1. Stepwise Regression	42
7.3.2. Neural Nets	43
7.3.3. Boosted Trees	44
7.3.4. Bootstrap Forest	44
7.3.5. Model Comparison.....	45
7.4. Experimental Design Types	48
7.5. Correlation and space-filling design characteristics	49
7.6. Traditional and Space-Filling Designs.....	51

7.7. State-of-the-art space-filling designs	53
7.8. Use Case Experimental Design	54
8. Technical Requirements for High Performance Computing Clustering.....	54
8.1. Select System Model Element Design Variables	55
8.2. Select Analytical Models, Develop Baseline Scenarios, and Map Design Variables to Model Inputs	56
8.3. Create Experimental Design.....	57
8.4. Generate a Study File that Specifies which Model Input Parameters to Change	58
8.5. Generate Excursion Files for Each Experiment (row in the design matrix).	58
8.6. Execute HPC Simulation Runs	59
8.7. Post-Process Output Files	59
8.8. Perform Statistical Metamodeling	60
8.9. ERS Tradespace Visualization.....	60
9. Simulation Analysis and Tradespace Visualization	61
9.1. Exploratory Analysis	61
9.2. Dashboard Tradespace Visualization	70
9.2.1. Prediction Profiler Dashboard Component	71
9.2.2. Contour Profiler Dashboard Component.....	74
9.2.3. Monte Carlo Filtering Component	75
9.2.4. Viable Variant Exploration	76
10. Multiple Objective Decision Analysis.....	84
10.1. Qualitative Functional Objective Value Hierarchy.....	85
10.2. Quantitative Functional Objective Value Model	87
10.2.1. Natural Single-Dimensional Value Functions.....	88
10.2.2. Constructed Single Dimensional Value Function	89
10.2.3. Swing Weights for Value Measure Tradeoffs	91
10.2.4. Multi-Objective Value Function.....	93
10.3. Value and Cost Tradeoff Analysis.....	94
11. Conclusions and Future Work.....	102
11.1. MBSE Methodology Review.....	102
11.2. Technical Gap Bridges.....	104
11.3. Concluding Remarks	106

11.4. Future Research	106
References	108
Appendix A: Data Farming Instructional Manual	111
Appendix B: Design Creator Front-End User Manual	167
Appendix C: JMP Dashboard Building Instructions	174

Executive Summary

This technical report proposes an experimental design Model-Based System Engineering methodology that illuminates system design trade decisions in order to clearly identify key tradable variables and narrow the selection of viable system variants.

In today's complex environment, the DoD needs systems that are resilient to change and are effective across a wide variety of uncertain futures. The current Department of Defense (DoD) Acquisition lifecycle is a process driven approach that impedes the ability to rapidly develop resilient systems. The work-force is stove-piped, the data used to inform design decisions are lost in the process, and there is a lack of flexibility to adapt to changing requirements and mission contexts. Requirements are generally frozen early in the process making the difficult to change that result in inefficient use of time, resources and money. To address the resiliency challenge, the US Army Engineer Development Center (ERDC) is developing an Engineered Resilient System (ERS) Architecture that will leverage information technology to inform better manufacturing options during all stages of the lifecycle. The intent is to develop an open architecture that connects existing tools, information, and data in a common framework that is non-proprietary, platform agnostics, compatible with legacy systems, and can be shared among a wide variety of users. The ERS architecture provides the means to incorporate previous design successes, integrate models, generate the data needed to visualize the tradespace, and create a shared digital thread of design decisions accessible to a community of users throughout the system lifecycle.

A system design involves hundreds of tradable variables that must be balanced in order to develop a viable system solution that meets the demands of the stakeholders and performs effectively. Understanding the key tradable variables that have the most influence on a system design problem is critical during the conceptual design of a system. A tradeoff is a compromise between objectives such that improving one requires that we degrade another. Tradeoff decisions are based on data, information, and knowledge acquired from simulation model outputs, developmental and operational testing, subject matter expertise, and legacy system architectures. Therefore, in order to effectively make quality decisions while minimizing the impact of requirement changes, DoD needs a data driven rather than a process driven approach to design new systems. The end result will be better informed decisions, faster engineering, less rework, and allow for a wider range of alternative solutions to progress through the phases without freezing the requirements too early.

An effective way to manage the design of a system is to use the Model-Based System Engineering (MBSE) approach. MBSE is a new paradigm that supports the specification, analysis, design, and verification of a complex system while using an integrated system model with a dedicated tool. MBSE is gaining popularity and is expected to become a common state of practice in the near future. The integrated system model effectively manages auditable records of a system design by defining a system element once to be used throughout the model. As a result, once a change is made to an element in the integrated system model, the dedicated tool will instantly identify how the change will impact the system. The MBSE approach uses a system modeling language (SysML) to express the structural and behavior elements of a system. A system's structure is defined as a set of structural blocks with value properties that define the system's configurations. The settings of the collection of value properties characterize each system alternative. A limitation of SysML is that they only provide static diagrams. In

order to conduct sophisticated engineering analysis, we must incorporate external models and simulations.

During the system lifecycle we use a variety of models and simulations that represent different domains; these domains include operational effectiveness, physical feasibility, life-cycle costing, manufacturability, reliability and many more. The inputs to these models and simulations represent the system value properties that define the alternative configurations. The value properties that have the highest impact on the model output performance are the system design drivers we are most interested in. Currently, there is a technical gap with regard to our ability to untangle the system design drivers when there is a high volume of multi-dimensional data. The general state-of-practice is to perform brute force simulation runs on a small set of baseline and excursions that do not effectively explore the system alternative design space. There is a lot of time, money, and resources devoted to building complicated simulation models and we do not use them to the maximum extent possible if we only compare a few excursions from the baseline. The most effective way to determine the system design drivers is to leverage the method of statistical experimental design. The field of design of experiments (DOE) allows the analyst to identify which model inputs effects the outputs of interest. DOE provides a number of benefits that can assist in the design of a system. We can clearly identify the model inputs that affect the output responses, identify interactions that may exist between model inputs, uncover detailed insight into the model's behavior, examine the modeling assumption implications, frame the questions when we do not know what to ask, challenge or confirm our expectation of directional model input effects and their relative importance, and uncover problems with simulation program logic.

In order to untangle the system design drivers across several different domain models, our methodology uses statistical metamodeling to approximate the simulations' behavior. A statistical metamodel is an empirical model developed from either observational or experimental design data that relates a set of inputs to an output. We build metamodels using a number of statistical methods that include stepwise regression, boosted trees, neural nets, and bootstrap forest. Generally, regression metamodels are excellent at describing individual model input impacts on model outputs while the other metamodeling methods are better able to predict effectiveness by interpolating in-between simulated points. Ultimately, the analyst must understand which methods to apply for either understanding or predicting model behavior.

To generate the data needed to fit a metamodel, we advocate using a new class of space-filling experimental designs known as the Nearly Orthogonal Latin hypercube and nearly balanced design. These designs are efficient, minimize the correlations between columns, can handle continuous, discrete, and categorical data, and effectively explore the interior of the experimental design region. These new designs allow us to determine the driving factors, detect interactions between input variables, identify points of diminishing or increasing rates of return, and find thresholds or change points in localized areas. These insights can be incorporated within the system integrated model as derived requirements or rationale for design decisions.

We create a dynamic dashboard using the collection of metamodels to help visualize multi-dimensional model output landscape using horizontal and vertical cross sections. These cross sections allow us to clearly identify the tradable variables and find viable system variants that met the desired capabilities across multiple viewpoints and are physically feasible. We can easily visualize the model output landscape with a surface plot when there are only three dimensions. Because there are often

several more dimensions in a systems design problem we developed a dashboard that visualizes horizontal and vertical cross sections of the multi-dimensional model output landscape. We use a contour profiler which is a two dimensional projection that shows a horizontal cross section of a model output landscape within the experimental design region. Visualizing the selected projections allows the user to interactively explore how multiple model outputs depend on two selected model inputs. The contour profiler allows us to set limits on the model outputs to help define infeasible and feasible regions; the shape of these shaded regions is dependent on the functional form of the multi-dimensional metamodel. We can also visualize the model output landscape using prediction profilers that show vertical cross sections. These vertical projections show each model input's impact of the model output.

The dashboard colors each vertical cross section such that green indicates a positive impact to the model output, red indicates a negative impact, white indicates a model output with a target value, and black indicates no impact. Additionally, there is a color gradient applied so the cells with a higher impact are darker and the cells with a lower impact are lighter. These colors allow us to clearly identify the key tradable variables; when there is a color contrast between green and red within a model input's vertical cross section, we must make a trade by accepting an improvement in one model output while accepting a degradation in another. Because the model input vertical cross sections are sorted from left to right based on their overall impact across all model outputs, we can clearly identify which model inputs have the highest impact on the design problem.

In addition to identifying the key tradable variables, our dashboard provides an optimization algorithm to find a solution that balances the established model output limits with a weighted desirability function; the desirability function normalizes the model output scale. When our solution does not meet one or more of the model output limits, we then can look to the vertical projections to identify the model inputs that have the highest impact on the unfeasible model outputs; changing these model inputs may result in a feasible solution. If we cannot change a model input to find a feasible solution we then must trade off infeasible model outputs limits in order to arrive at a viable system variant. We can arrive at a variety of viable system variants by setting different model output limits and weights to each of them.

Once we identify a reduced set of viable system variants, we then use multi-objective decision analysis (MODA) to help inform the design decision. We use an additive value model that incorporates a composite perspective of multiple stakeholders and competing objectives. We use the philosophy of value-focused thinking and the mathematics of MODA to arrive at a final design decision.

Currently, there is a technical gap with regard to our ability to untangle the system design drivers when there is a high volume of multi-dimensional data. This technical report outlines the procedural workflow of our proposed MBSE methodology. We address the technical gap by leveraging the methods of experimental design in order to clearly identify tradable variables and narrow the search for viable system variants.

1. Introduction

Our future operational environment will require the Department of Defense to acquire Engineered Resilient Systems (ERS) that will adapt to continuously changing demands. To address this challenge, the Engineer Research Development Center (ERDC) is developing an ERS Architecture that will construct and maintain a data thread of architectural decisions that will break the barriers between the operators, engineers, logisticians, acquisition experts and others allowing multiple communities of interest to collaborate during a system's lifecycle. Narrowing in on the key system drivers and critical trade decisions during Pre-Milestone A is a daunting task given the high number of design variables, system complexity, and uncertain future. First-order engineering models provide engineers insight into design feasibility but without analyzing system concepts within a mission context we have no way of understanding the effectiveness of a system design variant. In order to understand a system's operational effectiveness during the conceptual design phase we must rely on combat models and simulations; depending on the model fidelity, these simulations can take days or weeks to run. During a simulation study, the general state of practice is to do brute force simulation runs by leveraging High Performance Computer Clusters (HPC) to generate data for tradespace exploration. Despite recent breakthroughs in computation capabilities, a petaflop computer *cannot* effectively explore a high-dimensional tradespace study (Sanchez et al. 2009). For example, if we wanted to explore 100 factors at a high and low setting, assuming that a simulation runs as fast as a single operation, it would take 40 million years to complete one replication of the experiment. To overcome these challenges, we must leverage the statistical methods of experimental design.

The field of Design of Experiments (DOE) allows us to efficiently explore a high-dimensional tradespace problem in a feasible amount of time. Recent developments in DOE allow analysts to efficiently explore a large number of input factors; for example, we are now able to study 100 continuous factors with only 101 experiments (MacCalman 2013). After performing an efficient experiment, we can develop statistical metamodels, or mathematical equations that approximate the input output behavior of a simulation model; the metamodel then becomes a surrogate of the simulation. These surrogate metamodels allow the analyst to explore a wide range of input factors in order to identify the ones that drive system behavior and reveal the critical tradespace decisions. In addition, the metamodels provide the analyst a means to display a dynamic visualization dashboard that can illuminate the key trade decisions. Tradespace dashboards facilitate operational commanders and domain specific engineers to collectively make trade decisions by revealing the impact of system design configurations on multiple measures of effectiveness across different scenarios.

Recently, a new approach called Model-Based Systems Engineering (MBSE) is gaining popularity and is expected to become a common state of practice in the near future (Friedenthal et al. 2011). According to the International Council of Systems Engineering (INCOSE), MBSE is a methodology characterized by a collection of processes, methods, and tools used to support systems engineering design in a "model-based" context (INCOSE 2015). The System Modeling Language (SysML) is a visual language with a common semantic and notation standard that facilitates MBSE to support specification, analysis, design, and verification of a complex system. The practice of MBSE is emerging in different domains and uses. One of these areas examines the linkages of the SysML diagrams to external models. Because our primary interest is to leverage operational simulations that analyze system configurations in different mission

contexts, we need to explore the linkages between system elements within the SysML diagrams and the simulation model input parameters. By mapping SysML system component configurations to model input parameters, we can perform DOE to explore a high-dimensional problem to identify which input parameters affect the simulation output measures. Additionally, metamodeling dynamic dashboard exploration can influence design decisions and reveal where they satisfy the system engineering requirements.

ERDC has a number of technical thrust areas that support the development of the ERS Architecture; one of them is the tradespace analytics thrust area that seeks to increase the effective coverage of trade considerations within the ERS Architecture. The objective of our research is to derive an MBSE methodology that leverages DOE in order to illuminate the tradeoffs for a complicated system design problem. To demonstrate the methodology, we will use a notional representative use case involving new technologies that will enhance the Infantry Squad to overmatch current and future adversaries. The squad enhancement technology system provides us with a complex high-dimensional system design problem involving sensors, weapons, radios, body armor, exoskeletons, unmanned aerial vehicles, and robots that must operate in a variety of environments with several feasibility and life-cycle considerations that conflict with each other.

Our procedural demonstration starts with a defined system design problem that has an MBSE integrated model expressed in the SysML language. Simulations from various domains are selected to measure effectiveness and performance related to different aspects of the problem. An experimental design is performed for each model using a High Performance Computing Cluster (HPC). Once the output data is post-processed, the analyst performs exploratory analysis and fits surrogate metamodels that approximate the simulation model's behavior. These metamodels are then used to create a dynamic dashboard that allows the user to explore a high-dimensional complex problem to illuminate key tradable variables and identify a narrow set of viable system alternatives. The set of alternatives are then analyzed using multi-objective decision analysis (MODA) models to help inform design decisions when there are multiple stakeholders and competing objectives. Figure 1.1 is a visual depiction overview of the procedural workflow that each section in the technical report will expand on.

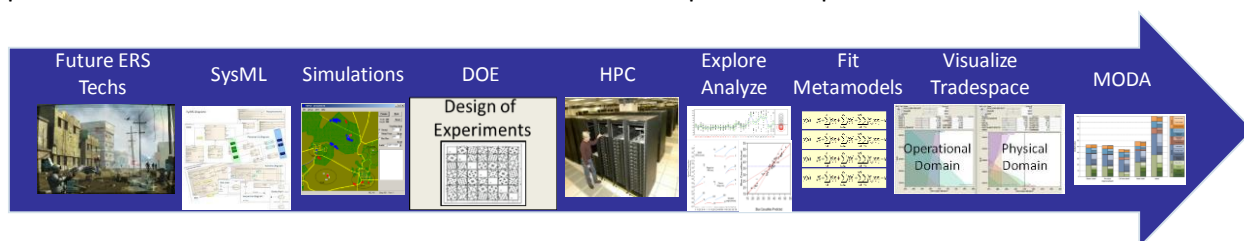


Figure 1.1 Procedural workflow of the proposed MBSE methodology.

2. Technical Report Organization

This technical report is organized into eleven sections with three appendices that outline the details of our proposed methodology. Section 3 provides a background of the Engineered Resilient System Architecture and describes the differences between a process-driven approach versus a data-driven approach. Section 4 discusses the major research contributions our methodology provides to fill the technical gaps identified by ERDC. In order to demonstrate our methodology, we introduce in Section 5

our representative use case and the operational simulation models we use to explore the mission context space. Section 6 discusses the benefits and limitations of the Model-Based Systems Engineering Approach and includes a discussion of the need to integrate external models. Section 7 describes how we can approximate simulation model behavior with statistical metamodels and use them to gain insights that can inform system design decisions; additionally, it provides an overview of the field of design of experiments and the types of designs used for experimental studies. Section 8 provides a functional flow description of the technical requirements necessary to facilitate our methodology. Section 9 discusses the types of analysis and insights we can gain from the experimental design study as well as introduce a visual dashboard that can identify the key tradable variables and narrow down the search for viable system variants. Once we arrive at our narrow set of viable variants, we then discuss multiple-objective decision analysis methods that assist stakeholders with alternative decisions using a functional objective value hierarchy model. Section 11 concludes the technical report and discusses future research endeavors. Appendix A provides a detailed step-by-step manual that describes how to perform data farming of the MANA simulation models on a cluster of computers. Appendix B is a user manual on how to use a custom experimental design creator that is especially suited for system design studies. Finally, Appendix C provides a user manual that contains the instructions needed to build our dashboard described in Section 9 in JMP™ 12.

3. Background

“A resilient system is trusted and effective out of the box in a wide range of contexts, easily adapted to others through reconfiguration or replacement, with graceful and detectable degradation of function” (Neches 2011). The need to engineer resilient systems is becoming more prevalent in today’s operational environment. The Department of Defense (DoD) will continue to rely on material solutions to address critical capabilities in response to our Nation’s threats, especially in an environment where our potential adversaries have global availability of rapidly developed technology. Engineering resilient system solutions that have a broad capability to perform in a wide variety of mission contexts against several potential enemies and uncertain futures is essential to protecting our Nation. In the FY13-17 Program Objective Memorandum, the Secretary of Defense designated Engineered Resilient Systems (ERS) as a Science and Technology (S&T) priority and tasked the Assistant Secretary of Defense for Research and Engineering to oversee the development and implementation of the ERS roadmap.

3.1.Process versus Data Driven Approach

Currently the acquisition community uses a process driven approach to develop capability requirements and mature technological solutions through multiple phases. Within the DoD community, the capabilities process is known as the Joint Capabilities Integration Development System (JCIDS). Figure 3.1 shows the interactions between the JCIDS process and the acquisition process phases.

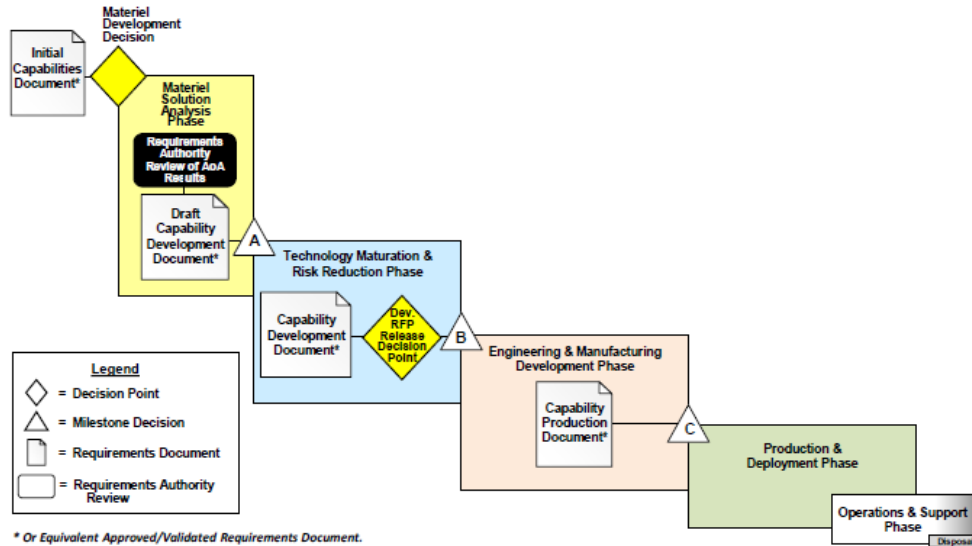


Figure 3.1. Interaction between the DoD JCIDS and Acquisition process (retrieved from <http://acqnotes.com/acqnote/acquisitions/jcids-overview>).

The current process driven approach impedes DoD's ability to rapidly develop systems that are resilient to uncertain futures. The work-force is stove-piped, the data used to inform design decisions are lost in the process, and there is a lack of flexibility to adapt to changing requirements and mission contexts. Requirements are generally frozen early in the process making them difficult to change resulting in inefficient use of time, resources and money. Figure 3.2 shows in red the implications of a requirements change downstream of the process and the need for the types of data to redesign and rework the system.

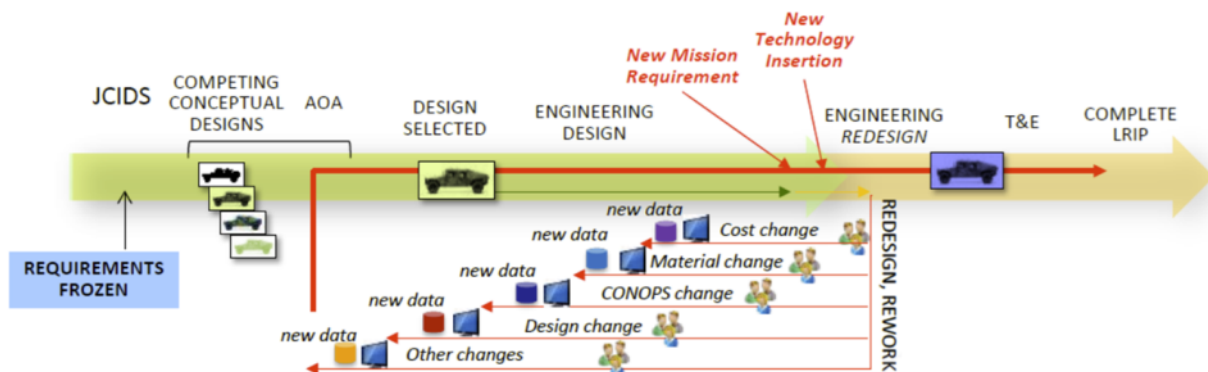


Figure 3.2. Illustration of the redesign and rework resulting in a requirements change downstream of the process.

Ensuring that systems are resilient to changing environments poses significant challenges to the Acquisition community. These challenges involve multiple competing objectives involving several different domains. The types of domains include operational effectiveness, physical feasibility, life-cycle costing, manufacturability, reliability and many more. During the design of a system there are several key stakeholders involved with design decisions that all have different perspectives across each domain. Some of the stakeholders include program managers, operational users, modelers, system certifiers, testers, concept design engineers, detailed design domain engineers, and cost estimators to name a

few. All of these multiple competing objectives and stakeholder needs require tradeoffs that balance a wide range of different domain considerations when designing a system.

A tradeoff is a compromise between objectives such that improving one requires that we degrade another. A classic example is the tradeoff between force protection and maneuverability. Increasing the force protection of a vehicle may require additional armor that weighs down the platform making it less maneuverable. In order to increase force protection we must tradeoff our ability to move fast over various terrains due to the increased load required. A system design involves hundreds of tradable variables that must be balanced in order to develop a viable system solution that meets the demands of the stakeholders and performs effectively. Understanding the key tradable variables that have the most influence on a system design problem is critical during the conceptual design of a system. Tradeoff decisions are based on data, information, and knowledge. Therefore, in order to effectively make quality decisions while minimizing the impact of requirement changes, DoD needs a data driven rather than a process driven approach to design new systems. The end result will be better informed decisions, faster engineering, less rework, and allow for a wider range of alternative solutions to progress through the phases without freezing the requirements too early. Figure 3.3 shows a data driven approach that begins with millions of possible designs and leverages conceptual modeling to configure systems before analyzing alternatives. This approach allows the designers to easily redesign and rework alternatives with the conceptual model, data, information, and knowledge that is available.

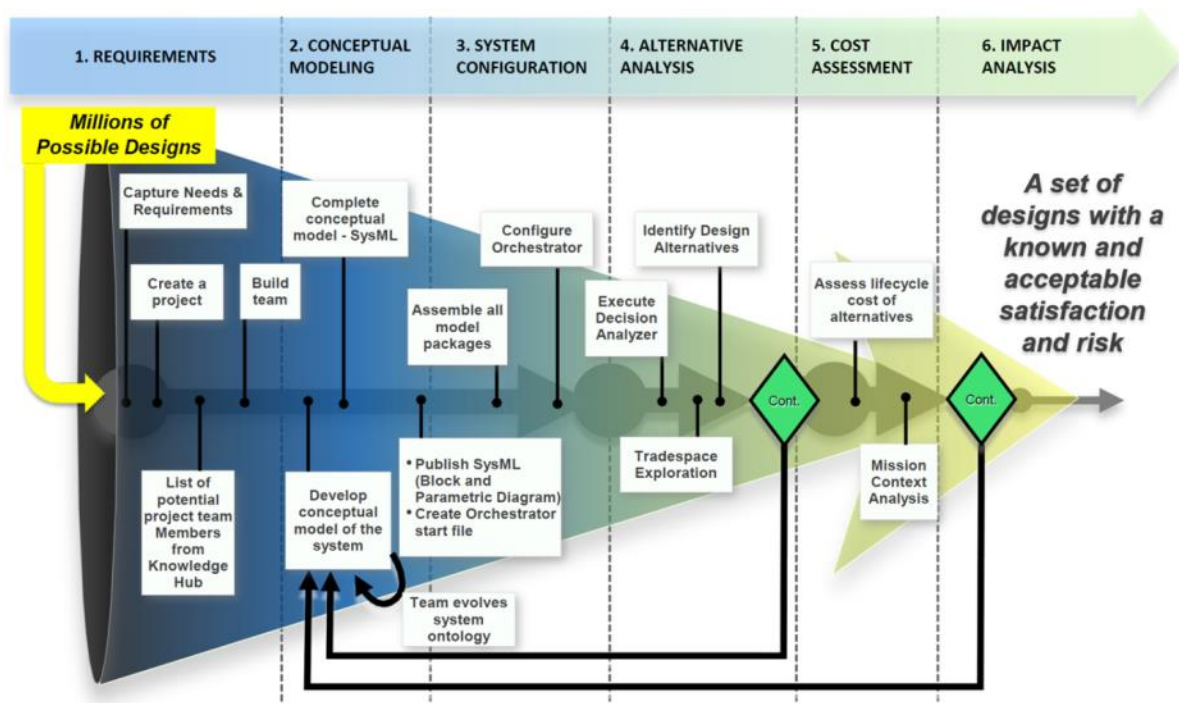


Figure 3.3. Data driven approach.

3.2.Engineered Resilient System Architecture

With today's increased computational power and availability we are now able to explore data and collaborate effectively across several different domains with multiple stakeholders. In response to the Assistant Secretary of Defense for Research and Engineering priority to engineer resilient systems, the

US Army Engineer Research and Development Center (ERDC) is developing an Architecture that leverages information technology to help Acquisition teams make better informed design decisions throughout the system lifecycle. The intent is to develop an open architecture that connects existing tools, information, and data in a common framework that is non-proprietary, platform agnostics, compatible with legacy systems, and can be shared among a wide variety of users. Figure 3.4 is an Operational View -1 Diagram that depicts the ERS architecture as a computational cloud that integrates multiple domain models, simulations, and data to develop a digital thread accessible to several communities of interest for collaboration during the system design.

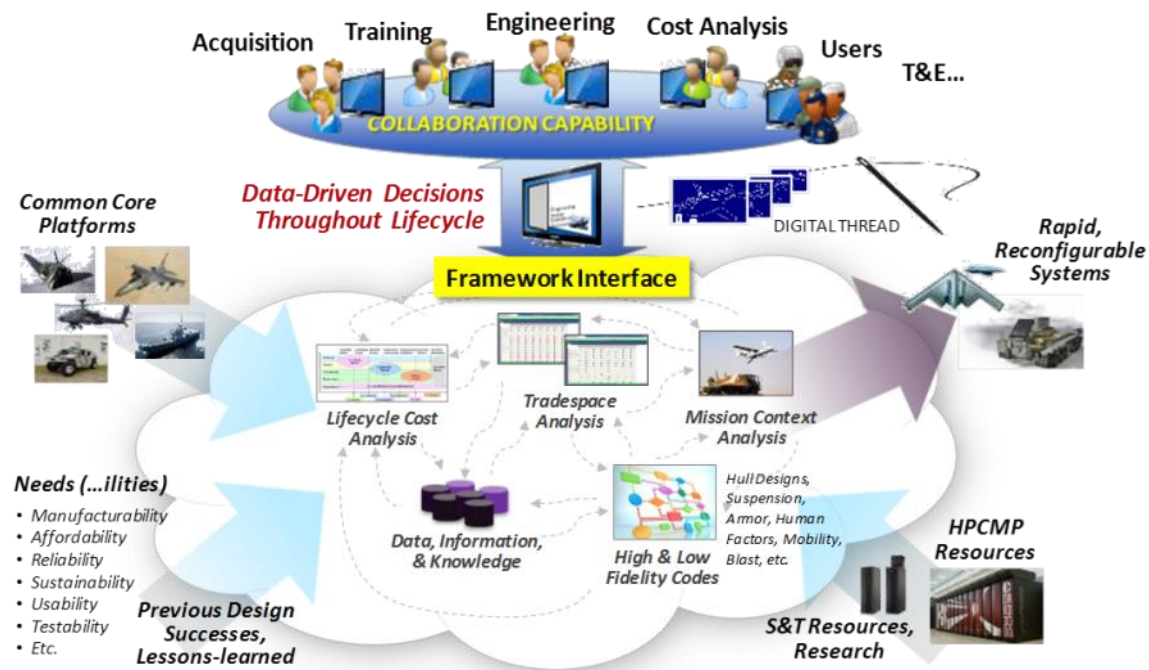


Figure 3.4. ERS Architecture Operational View – 1 Diagram.

Inputs to the ERS Architecture are common core platform information from previously designed successful systems. These inputs constitute the data, information, and knowledge that contain the functional, logical, and physical architectural concepts from several different domains. These domains include physics-based high and low fidelity codes, life-cycle costs, mission contexts, and all the life-cycle consideration needs (-ilities). Models and simulations are used to explore several system designs with respect to each of the domains the models represent. The ERS Architecture uses High Performance Computing Resources to run simulation experiments to generate the data needed to create a tradespace environment that highlights the key tradable variables within a design decision. The data and tradeoff environment is available through a portal for the wider community to collaborate on design decisions throughout the system lifecycle.

Understanding the linkages between the conceptual architectural information, the simulations that model different domains, and the data that is generated to create the tradeoff environment is an ongoing research effort. In order to enable these linkages ERDC has organized the ERS Architecture development into the following five technical thrust areas:

1. **System representation and conceptual modeling.** The development of the functional, logical, and physical architectures that model the system's element structure, behavior, relations between elements and their interoperability. The approach that enables this trust area is the Model Based System Engineering (MBSE) approach which is the subject of Section 6 of this report. The MBSE approach reduces the impact of requirement changes and allows for alternatives to be kept longer and explored deeper.
2. **Characterizing changing operational contexts.** Constructing, verifying, and validating simulation model scenarios that represent a wide range of mission contexts. Provides a deeper understanding of the warfighter needs and allows designers to refine the operational context of changing mission requirements.
3. **Cross-Domain coupling.** The coupling between the simulation models that will enable designers to better understand the linkages between the domains each model represents. Model designers must understand the model interchange requirements and the impact of fidelity differences between models.
4. **Data driven tradespace exploration and analysis.** The generation of the tradespace environment that allows for the exploration of a multi-dimensional design problem that illuminates the key tradable variables and helps narrow down the viable system variants.
5. **Collaborative design and decision support.** Enables well-informed, low-overhead discussion, analysis, and assessment among engineers and decision makers. Provides collaborative analysis of engineering issues and impacts.

ERDC has established the following envisioned end state over the next few years:

- Improved engineering and design capabilities: more environmental and mission context, more alternatives developed, evaluated and maintained, better trades: managing interactions, choices, and consequences.
- Improved systems: highly effective: better performance, greater mission effectiveness, easier to adapt reconfigure or replace, confidence in graceful degradation of function.
- Improved engineering process: fewer rework cycles, faster cycle completion, better managed requirements shifts.

4. Research Contributions

Brute force simulation experiments that focus on a limited set of system alternatives do not allow Systems Engineers to effectively explore a wide variety of design alternatives early in the conceptual design. Incorporating DOE methods within a MBSE design methodology allows for the exploration of a wider range of alternatives. In addition, surrogate metamodels that approximate a simulation's behavior provide valuable insights into the design problem by identifying the most critical drivers for key measures and form the basis to build a dynamic dashboard that can explore a high-dimensional design problem. Our proposed MBSE methodology addresses several technical tradespace analytic gaps identified by ERDC. Table 4.1 summarizes the technical gaps and the research contributions that address each of these gaps. In Section 11, we conclude with a more detailed explanation of how we address these technical gaps by proposing our MBSE methodology.

Table 4.1. Research contributions to tradespace gap mitigations.

Area	Gap	USMA DSE ORCEN Contributions
Tradespace Analytics	<ul style="list-style-type: none"> Tradespace analyses produce static rather than dynamic visualization of alternatives The tradespace is incompletely defined and not guaranteed to be bounded Computational tools limit exploration to a few variables at a time 	<ul style="list-style-type: none"> Introduce experimental designs suited for complex simulation models that allow the analyst to explore a high-dimensional problem Illuminate the key tradable variables with a dynamic dashboard created in JMP that combines statistical analytic artifacts with visualization features
Decompose Tradespace	<ul style="list-style-type: none"> Decomposition of the system is based on static requirements Design-driving variables are unknown Tools and methods for conceptualizing design alternatives overlook or prematurely eliminate feasible designs, inadvertently limiting early trades Insufficient data available to populate a tradespace for new designs 	<ul style="list-style-type: none"> Demonstrate an MBSE methodology that maps SysML elements to experimental design factors and incorporates analytical insights as derived requirements Consider a wide range of design alternatives by utilizing the methods of experimental design Demonstrate statistical methods that identify the most significant design drivers, the nature of their impact, and the synergies that exist between them
Tradespace Search	<ul style="list-style-type: none"> Methods for searching tradespaces are slow due to: <ul style="list-style-type: none"> increasing complexity a growing number of variables high-fidelity models for accurate results Thus regions of feasible design space may be missed 	<ul style="list-style-type: none"> Leverage state-of-the-art space-filling experimental designs to explore the entire design space Perform efficient experiments of complex models with a large number of input variables Demonstrate how to identify a narrow set of viable system alternatives that effectively span the design region
Evaluating Tradespace Results	<ul style="list-style-type: none"> Despite massive amounts of data, decisions are still difficult to make due to competing stakeholder requirements (perspectives) System attribute weighting factors are often subjective, making it challenging to build a consensus Trade results are often biased towards performance metrics only 	<ul style="list-style-type: none"> Integrate the philosophy of value focused thinking and the mathematics of multi-objective decision analysis to incorporate multiple stakeholder viewpoints Utilize swing weight matrices to simultaneously incorporate subjective importance and objective levels of capability impacts

5. Representative Use Case Description

To demonstrate our MBSE methodology we chose a representative use case that involves an opportunity to invest in new technologies that will increase the capabilities of the Infantry Squad. The system is the collection of integrated technologies that enhance the squad's effectiveness (sensors, weapons, exoskeletons, radios, UAVs, robots, body armor). This use case provides a lot of opportunities to highlight tradeoffs across multiple types of costs, performance, schedule, and risk considerations. It also allows for a wide variety of solutions/alternatives that are comprised of different combinations of the seven system components. In order to illustrate our methodology clearly, we use two versions of our squad enhancement use case. We refer to the first one as the large squad problem that includes 38 model inputs and over 40 outputs. Our second version is referred as the small squad problem that only includes 4 inputs and 6 outputs. During our technical report, we interchange between these use cases while demonstrating our methodology.

5.1. Agent-Based Model Overview

In order to evaluate the system in an operational context, we use an agent-based simulation called Map Aware Nonuniform Automata (MANA). MANA is a stochastic, agent-based, time-stepped

simulation modeling environment, developed by the New Zealand Defense Technology Agency (McIntosh et al. 2007). MANA is a low-resolution simulation of combat, intended to “capture only enough physics as is necessary.” For example, range-probability pairs are used to capture sensor, weapon and communication device effectiveness, vice attempting to explicitly simulate the physics involved. MANA has been used for numerous studies, including many Masters theses, at the Naval Postgraduate School, see <https://harvest.nps.edu/>. Additionally, we are aware of its use in studies at the RAND Corporation, Office of Naval Research, Marine Corps Operations Analysis Division, Marine Corps Warfighting Lab, and Army G-8.

The basic entity in MANA is an agent, which can be made to represent a soldier, group of soldiers, or major combat platform or vehicle, as desired. Agents can see, shoot, move, communicate according to the properties given to them. These entities (agents) interact with each other, as well as the environment in which they operate, and make decisions based on their movement goals and their situational awareness. A “squad” is the MANA term for a group of agents who share the same physical and behavioral properties. The squad can have any number of member agents, so depending upon how the user has set it up, it may translate directly to an entire Infantry Squad, or to homogeneous members of an Infantry Squad, e.g. Grenadiers.

Agents attempt to see other agents on the battlefield with their organic sensors. If an agent is able to classify another agent with one or more of its sensors then it knows whether the sensed agent is a Friend, Neutral, or Enemy. Depending on the properties of its sensors, it’s possible that that an agent could detect another agent but not classify it, in which case the detected agent is treated as an “unknown.” An agent must be classified as enemy in order to be targeted with a weapon. Agents within the same squad post their contacts to a Squad Situational Awareness (SA) Map. Additionally, agents can pass contact information to other agent squads, using user-defined communication links, which can have properties such as reliability, capacity, latency, etc. Contacts that are passed to a squad over a communication link are posted to the Squad Inorganic Situational Awareness Map. Agents can be made to act upon contact information, for a user-specified period of time, from their Squad SA Map, their Inorganic SA Map, or both. Figure 5.1 contains a screen shot of the Graphical User Interface (GUI) for the Defense scenario.

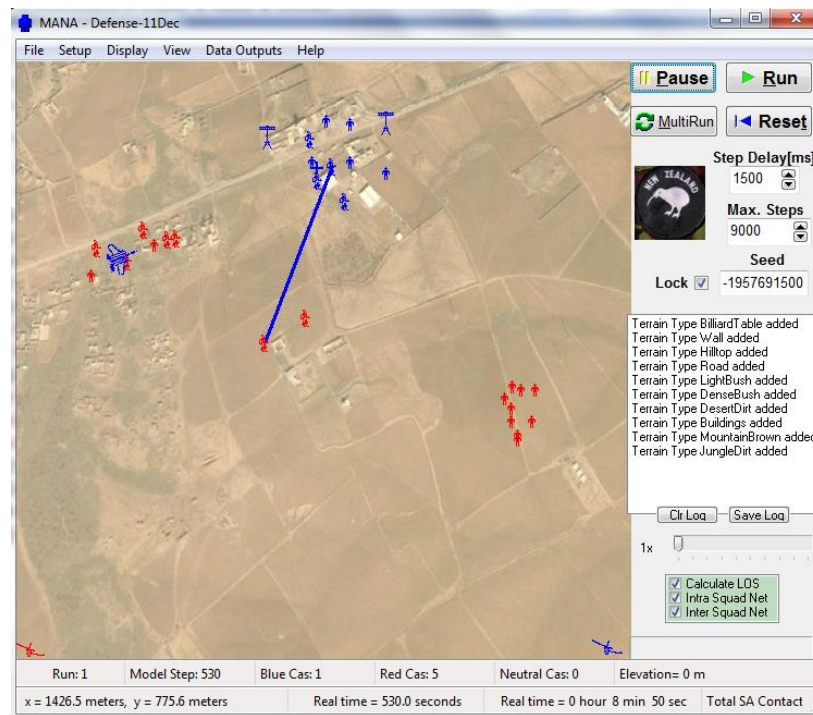


Figure 5.1: Screen Shot of the Defense Scenario

In MANA, the user has the option to create a terrain map, where each pixel color determines which type of terrain it represents. Each type of terrain is given properties of Going, Cover, and Concealment which defines how trafficable the terrain is, how much protection it provides, and how much concealment from view it provides, respectively. MANA comes with a few default terrain types (such as road, dense brush, hill, etc.) but the user can also define their own. Figure 5.2 contains a screen shot of the Scenario Map Editor, displaying the Terrain Map.

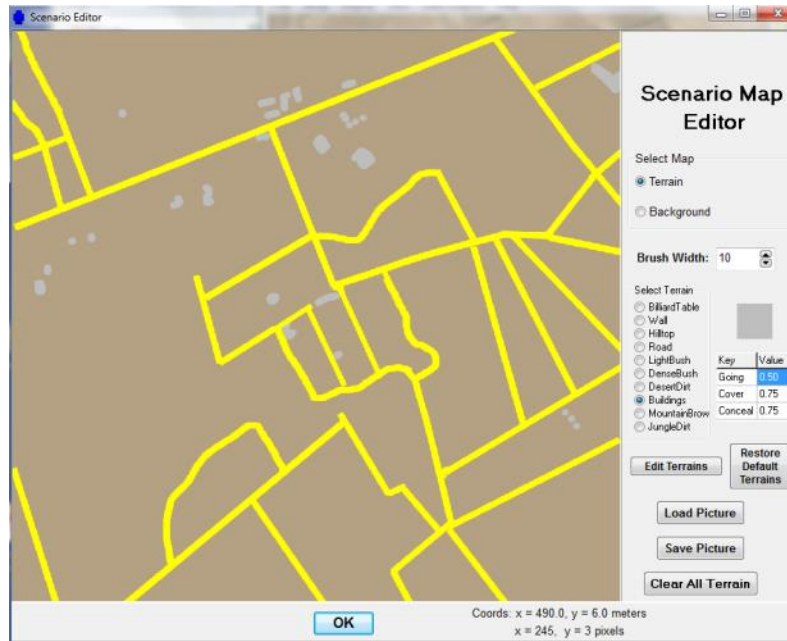


Figure 5.2: Screen Shot of the Scenario Map Editor, Displaying the Terrain Map

Although MANA is a very low resolution model, it has the capability of developing very interactive scenarios that reveal interesting emergent behavior. Agents are assigned weapon, sensor, and protection capabilities along with simple behavior propensities the agents execute based on what is in their situational awareness maps. When multiple agents have simple behaviors that interact with other agents, often times there are interesting results that emerge from these interactions. For our squad enhancement use case we developed four scenarios and focused primarily on the squad's capabilities without exploring behavior propensities. Our intent was to evaluate technologies without changing the squad behavior propensities so that they do not confound our analysis.

5.2. Simulation Scenarios

In every system design situation we are faced with multiple operational environments that the system is expected to achieve effects in. For our squad enhancement use case, we develop four scenarios to model the wide variety of mission contexts the squad is expected to operate in. The four scenarios are the attack, cordon and search, movement to contact, and defense missions; a description of each scenario follows:

Attack: Squad conducts an attack to seize terrain and destroy the enemy. Blue Force is equipped with only organic Squad technologies and communicates with their company HQ element to call for indirect fire. Squad moves towards the objective with their robot in front and UAV flying overhead. Blue Force calls indirect fire on the objective as they approach. Red Force is positioned on high ground with IEDs around the perimeter. Blue Force establishes a support by fire position with the SAW and Grenadiers while the rifleman assault the objective. Red Force calls for another enemy element to reinforce their

position. Blue Force calls indirect fire on the reinforcements once they identify their location. Scenario ends when all Blue or Red Forces are killed.

Cordon and Search: Squad patrols a village populated with several civilians. The cordon and search scenario will last for 72 hours. Blue Force is equipped with only organic Squad technologies and communicates with their company HQ element to call for indirect fire. Enemy insurgents initiate direct fire within the village and retreat within the civilian population. Initial, the Blue Force cannot distinguish between the civilians and the enemy insurgents. Snipers from a long distance outside the village fire on the Blue Force. Civilians flee out of the village in all directions. Enemy opens fire with a crew served weapon from a long distance and approaches the village from multiple directions to close with and destroy the Blue Force. Blue force positions robot in a position outside the village perimeter to provide security and will return to the Blue Force position for power recharging and return to the COP to refuel when necessary. UAV loiters about the village. Scenario ends when all Blue or Red Forces are killed.

Movement to Contact: Squad moves along a road with their robot in front and UAV in the air as they approach an enemy ambush position. Blue Force is equipped with only organic Squad technologies and communicates with their company HQ element to call for indirect fire. Enemy has multiple IEDs emplaced to initiate a complex L-shaped ambush. After the enemy initiates the ambush, another enemy element approaches from a distance to reinforce. Blue force reacts to the ambush, calls for indirect fire after identifying enemy reinforcements, and continues to close with and destroy the enemy. Scenario ends when all Blue or Red Forces are killed.

Defense: Squad is in a defensive position with a combat outpost with 10 foot walls, two gate entry points and fighting positions around their perimeter. The defensive scenario will last for 72 hours. Blue Force is equipped with only organic Squad technologies and communicates with their company HQ element to call for indirect fire; robots are positioned outside the perimeter to act as forward sensors and will return to the Combat Outpost (COP) for power recharging. Six enemy insurgents approach the COP wearing suicide vests and attempt to detonate at the gate to breach into the COP, make entry, and detonate additional suicide vest within the perimeter. The enemy then calls for indirect fire from a mortar position that is beyond line of friendly sight. Enemy crew served weapons open fire on the COP from a long range distance while the enemy approaches COP from three different directions. Blue force calls for indirect fire once they identify enemy force locations beyond line of sight. A UAV will loiter over the area of operations and return to the COP to refuel when necessary. Scenario ends when all Blue or Red Forces are killed.

5.3. Model Inputs

Table 5.1 shows the decision space for the squad enhancement system. To the left are the system and sub-system components that each has a number of local properties. For each local property, there is a stakeholder need that defines the property's desired improvement. These local properties are the decision variables that define the system characteristics of an alternative. We mapped each of the local

properties in Table 5.1 to a MANA model input and established their low and high settings that define the experimental region.

Table 5.1. System value property mapping to stakeholder needs and MANA inputs.

Decision Factors	Stakeholder Needs (Local Properties)	Simulation Input Model Description (MANA)	Low	High
SDR Detection Range	Increase the soldier's detection range.	Distance a soldier can detect a target.	1.5	2.5
SDR AVG Time Between Det	Reduce the time needed for the soldier to detect a target.	For a discrete set of distances, the time it takes for the soldier to detect a target.	2	2
SDR Classification Range	Increase the range that a soldier can classify a target as a threat, friendly, or neutral	Distance a soldier can classify something as a threat, friendly, or neutral.	1	2
SDR Classification Prob	Improve on the soldier's ability to classify a target as a threat, friendly, or neutral.	The probability a soldier can classify a target correctly.	1	2
SDR FOV	Increase the soldier's field of view.	The soldier field of view.	50	180
SDR Speed	Increase the soldier's mobility with an increased load carrying capacity.	Soldier speed.	3	7
SDR No. Hits to Kill	Increase the body armor protection of the soldier.	The number of hits to kill a soldier agent.	3	5
SDR M4 Range	Increase the range of the soldier's Rifle	Range of the soldier's Rifle	1	2
SDR M4 Rate of fire	Increase the rate of fire of the soldier's Rifle	The soldier's Rifle shots per second	1	3
SDR M4 Hit rate	Increase the soldier's Rifle accuracy.	For a discrete set of distances, the probability of hitting a target.	1	2
SDR M249 Range	Increase the range of the soldier's Automatic Weapon	Range of the soldier's Automatic Weapon	1	2
SDR M249 Rate of fire	Increase the rate of fire of the soldier's Automatic Weapon	The soldier's Automatic Weapon shots per second	3	8
SDR M249 Hit rate	Increase the soldier's Automatic Weapon accuracy.	For a discrete set of distances, the probability of hitting a target.	1	2
SDR 40mm Range	Increase the range of the Grenadier Weapons' range	Grenadier Weapon's range.	1	2
SDR 40mm Hit Rate	Increase the Grenadier Weapon accuracy.	The Grenadier Weapon's accuracy.	1	2
SDR 40mm shot radius	Increase the shot radius of the Grenadier Weapon round.	The Grenadier Weapon's shot radius on impact.	5	20
Comms Delay	Decrease the time it takes for a soldier to interpret incoming information from squad members.	The number of seconds between internal squad radio transmissions.	0	15
Inorganic SA - Latency	Decrease the time it takes to call for indirect fire and interpret information from UAV and Robots.	The number of seconds between external radio transmissions.	0	15
Inorganic SA - Reliability	Improve the soldier's ability to send, receive, and interpret information to external assets.	The probability of the sending and receiving an external radio transmission.	0.7	1
No. UAVs	Provide an organic UAV to the squad.	Number of squad organic UAVs	0	2
UAV Speed	Ensure the UAV has enough speed to maintain flight stability.	The speed of the UAV.	50	100
UAV Detection Range	Increase the detection range of the UAV.	Distance the UAV can detect a target.	1.5	2.5
UAV AVG Time Between Det	Reduce the time needed for the UAV to detect a target.	For a discrete set of distances, the time it takes for the UAV to detect a target.	1	2
UAV Classification Range	Increase the range that a soldier can classify a target as a threat, friendly, or neutral with a UAV.	Distance the UAV can classify something as a threat, friendly, or neutral.	1	2
UAV Classification Prob	Improve on the soldier's ability to classify a target as a threat, friendly, or neutral with a UAV.	The probability the UAV can classify a target correctly.	1	2
No. UAV Missiles	Provide a kinetic munitions to destroy threats.	Number of missiles on one UAV.	0	2
UAV Missile Shot Radius	Increase the shot radius of the UAV munitions without increasing collateral damage.	The UAV missile shot radius on impact.	10	50
UAV Missile Hit rate (Prob of Hit)	Increase the accuracy of the UAV munitions.	The probability of the UAV missile hitting a target.	0.3	1
No. Robots	Provide an organic Robot to the squad.	Number of squad organic Robots.	0	2
Robot Speed	Ensure Robot can traverse in a variety of terrain types.	The speed of the Robot.	3	10
Robot Detection Range	Increase the detection range of the Robot.	Distance the Robot can detect a target.	1.5	2.5
Robot Classification Range	Reduce the time needed for the Robot to detect a target.	Distance the Robot can classify something as a threat, friendly, or neutral.	1	2
Robot AVG Time Between Det	Increase the range that a soldier can classify a target as a threat, friendly, or neutral with a Robot.	For a discrete set of distances, the time it takes for the Robot to detect a target.	1	2
Robot Classification Prob	Improve on the soldier's ability to classify a target as a threat, friendly, or neutral with a Robot.	The probability the Robot can classify a target correctly.	1	2
Robot IED Sensor Class Prob	Increase the ability for a robot to detect and classify an IED.	The probability a Robot can detect and IED.	1	2
Robot No. Hits to Kill	Increase the Robot protection from kinetic fire.	The number of hits to kill a Robot agent.	4	6
Robot FOV	Increase the Robots field of view.	The Robot's field of view.	50	180
Robot Stealth	Reduce the size of the Robot in order to decrease the ability to detect the Robot.	The enemy's probability of detecting the Robot.	0.3	1

In some cases, the model input was implemented as a multiplier on the current capability of the squad. For example, the Grenadier's Weapon Range factor varying between 1 and 2 meant that the Grenadier's Weapon Range was (at the low end) as it was defined in the scenario base case, and (at the high end) increased up to twice its current capability. Figure 5.3 contains a screen shot of the properties of the Grenadier's weapon. For example, if the Grenadier Weapon Range input is 1.5 for a system alternative then each of the 7 ranges listed in the range table is multiplied by 1.5 and therefore, represents a 50 percent increase in capability.

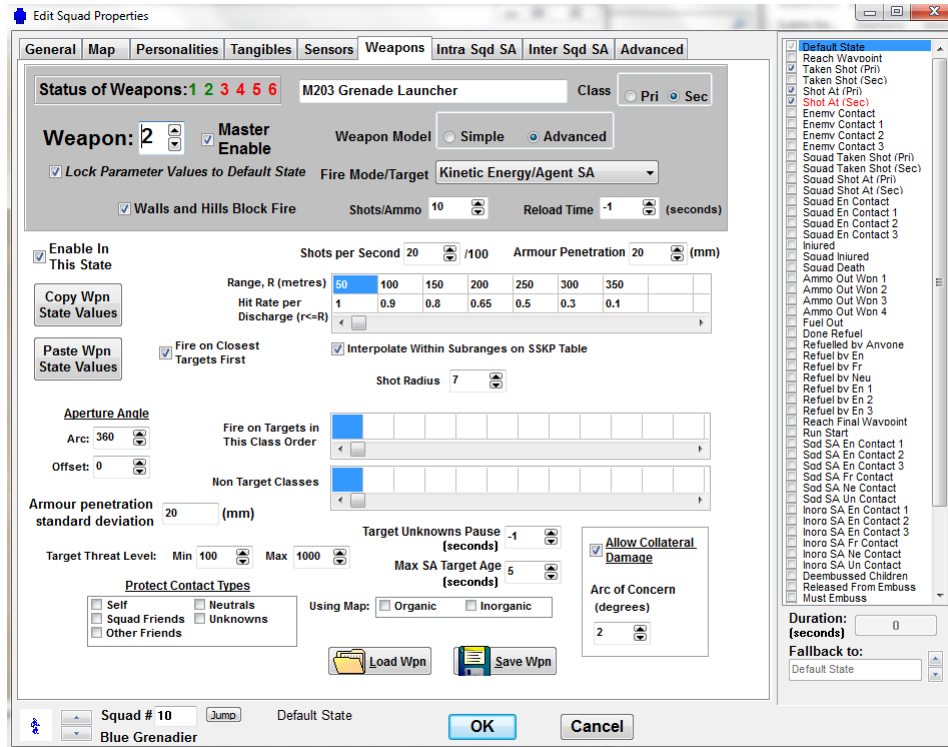


Figure 5.3: Screen Shot of the Weapons Properties of the Grenadier

The collection of MANA model inputs allows us to define a wide range of system alternatives that constitute the experimental region. In Section 7, we will discuss how we use an experimental design to explore the experimental region defined by the low and high settings shown in Table 5.1.

5.4. Model Outputs

The MANA simulation provides a wide range of output measures that allow us to evaluate system alternatives. These model outputs represent the measures we use to evaluate the operational domain within different mission contexts. Our squad enhancement use case uses the following model outputs for each scenario:

Protect: calculated as the total number of hits taken by Blue forces during the simulation run.

Aware: a metric that represents the time-weighted average of the number of Red classifications made by Blue forces during the simulation run. As the simulation progresses, the total number of enemy that the Blue force is aware of can be viewed as a state trajectory. Over time, the state trajectory increases or decreases based on the total number of Red Forces that the Blue Force is aware of. The *Aware* measure is calculated by finding the area under the state trajectory and dividing it by the total time of the simulation run. The measure represents the proportion of time the squad was aware of the enemy; the higher the proportion the more situationally aware the squad was during the simulation run.

Lethal: a measure that represents the time-weighted average of the number of enemy agent entities killed in action (RedKIA) during the run. It is calculated as the area under the state trajectory curve of RedKIA versus time. A high score (area under the curve) could be achieved by killing all Reds early in the simulation run. A lower score could be the result of killing a few Reds early, or by killing more Reds but much later in the run. The lowest score would be achieved by killing very few reds, late in the simulation. After dividing the total area under the RedKIA-versus-time curve by the total simulation time multiplied by the number of Red entities, we will get a metric that ranges between 0 and 1.

IEDProtect: a Binary measure of whether the squad avoided all Improvised Explosive Devices (IEDs), 1 means yes, 0 means no.

BlueKIA: Total number of squad members killed during the simulation run.

BLOS: Proportion of detections discovered beyond line of sight (from UAVs and robots) compared with those detect by line of sight.

LOS: Average Classification Distance of the soldier agents that evaluates the line of sight capability.

Sustain: Total number of rounds fired by the squad during the simulation.

In addition to the MANA model outputs that represent the operational domain, we incorporated other notional model outputs for other domains. These outputs include a manufacturability readiness level (levels 1-9, where higher levels indicate higher levels of manufacturability maturity), weight, cost, and time schedules that represent the maximum time along the system alternative's critical path.

6. Model Based System Engineering Approach

MBSE is a new paradigm that supports the specification, analysis, design, and verification of a complex system using an integrated system model with a dedicated tool. According to the International Council of Systems Engineering (INCOSE), MBSE is a methodology characterized by a collection of processes, methods, and tools used to support systems engineering design in a "model-based" context (NDIA 2011). The MBSE approach is gaining popularity and is expected to become a common state of practice in the near future (NDIA 2011). Some of the key benefits to the MBSE approach include investigating requirement compliance to system elements within the architecture, change impact assessments on requirement changes, and conducting trade space analysis for alternative architectural configurations during the conceptual design phase (Kim et al. 2013). The integrated system model effectively manages auditable records of a system design by defining a system element once to be used throughout the model. As a result, once a change is made to an element in the integrated system model, the dedicated tool will instantly identify how the change will impact the system. Applying the MBSE approach within the ERS Architecture is an important aspect for the systems representation and modeling technical thrust area. Some might say that the ERS Architecture is a means to apply the MBSE approach for systems design.

The three pillars of MBSE involve a modeling language, a methodology, and modeling tool (Delligatti 2013). A common language used by system engineers is the System Modeling Language (SysML); SysML is a visual language with a common semantic and notation standard that facilitates MBSE (Friedenthal et al. 2011). An MBSE methodology is a road map of design tasks that is applied to a specific domain or organization. A modeling tool is a software application that conforms to one or more modeling language and integrates all modeling artifacts into a cohesive system reference model. Our research proposes a MBSE methodology that incorporates design of experiments as a means to capture insights into a complex system design problem.

6.1. System Modeling Language (SysML)

The MBSE approach utilizes an integrated system model that contains a set of elements and relationships between them. The SysML language, defines several types of elements that represent different aspects of the system. The most commonly used structural and behavior elements are blocks that define elements of structure, activities that express a sequence of behavioral actions, interactions between elements, state machines that classify a block's state behavior and the event occurrences that trigger transitions to other states, requirements, use cases that define a system's context boundary and constraint blocks that bind value properties to mathematical expressions. SysML uses nine diagram types to depict different views of the system model. The following provides a brief description and examples from the large and small squad enhancement use case (defined in Section 5) for each diagram type:

Block Definition Diagrams (bdd): Bdds are diagrams that show blocks that have structural and behavioral features. A block is a type of structure that may exist within a system. The types of structural features include properties that represent parts, references to other external elements, values that represent instances of quantities, or textual descriptions, ports that represent the interaction points at the boundary of the block, and constraints that can be either mathematical expression that binds the value properties or external simulation models that define behavior. Bdds show the arrangement of blocks that represent elements of definition; these elements are the types of structural blocks that could exist within the system. Bdds are used often during several system engineering activities including stakeholder needs analysis, requirements definition, architectural design, tradeoff analysis, test and evaluation, and system integration. Figure 6.1 shows an example of a bdd from the large squad enhancement use case; the figure shows the part and value properties within each block.

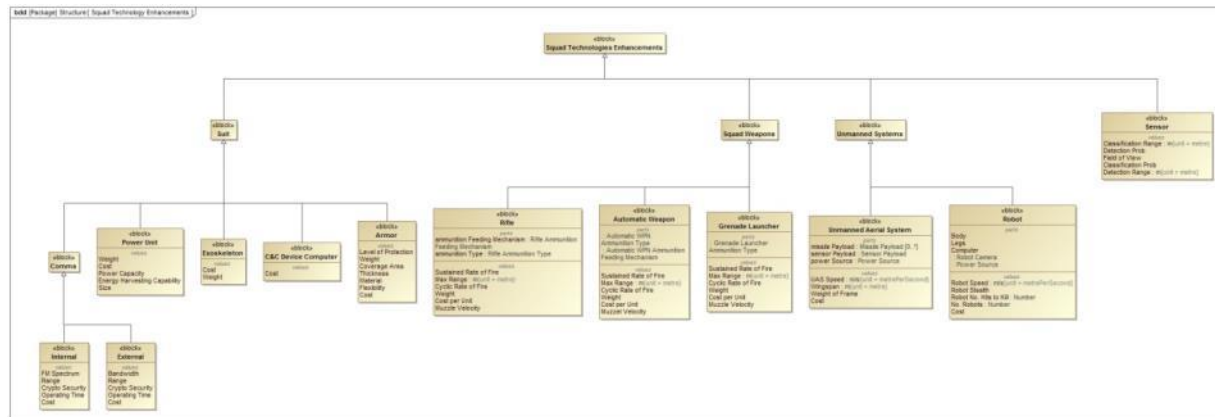


Figure 6.1. Block definition diagram from the large squad enhancement use case.

Internal Block Diagrams (ibd): Ibds display the internal structure of a block or group of blocks. Specifically, they show the type of connections between part and reference properties, the type of matter, energy, or data that flow across connections and the services that are provided and required across the connections. Because the large and small squad enhancement use cases are an example within the conceptual stage, we do not have internal block diagrams for them.

Use Case Diagrams (uc). The purpose of the use case diagram is to show the externally visible services that a system performs or provides. They are used to show a system boundary context diagram that reveals the external actors or elements that interface with the system. Figure 6.2 shows the uc diagram that shows the attack, cordon and search, movement to contact, and defense uses cases. The four use cases represent the services the system performs. The lines in the diagram indicate which external actors interact with the squad enhancement system during each use case.

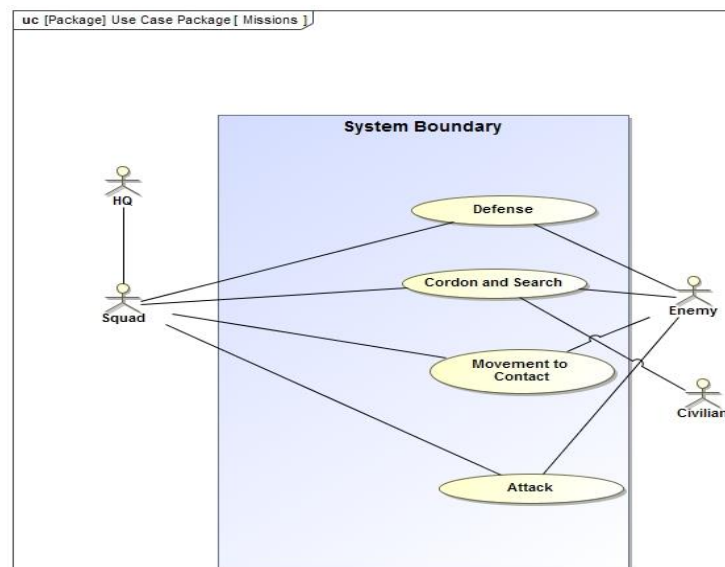


Figure 6.2. Use case diagram.

Activity Diagram (act). Activity diagrams express the dynamic behavior by showing the sequence of actions that flow in a specified order over a period of time. The diagrams are equivalent to the functional flow block diagrams system engineers use to express functional architectures. The diagrams consist of a series of actions, object nodes, and control nodes connected by edges that represent object and control flows. The purpose of the activity diagram is to convey the system's complex behavioral narratives for stakeholders to agree on; typically, they are nested under a Use Case model element. Figure 6.3 shows four Activity diagrams for each of the use cases shown in Figure 6.2. The diagram includes the actions that will provide the opportunities for all the squad enhancement system components to perform their functions. Not only do they allow stakeholders to agree on the system's intended behavior, they also provide valuable input to the simulation model builder by providing the sequence of behaviors needed to build the scenario. Additionally, the activity diagram in Figure 6.2 shows the actions that are allocated to block elements. These allocations are what constitute the functional allocation of the behavior elements to the system structure elements.

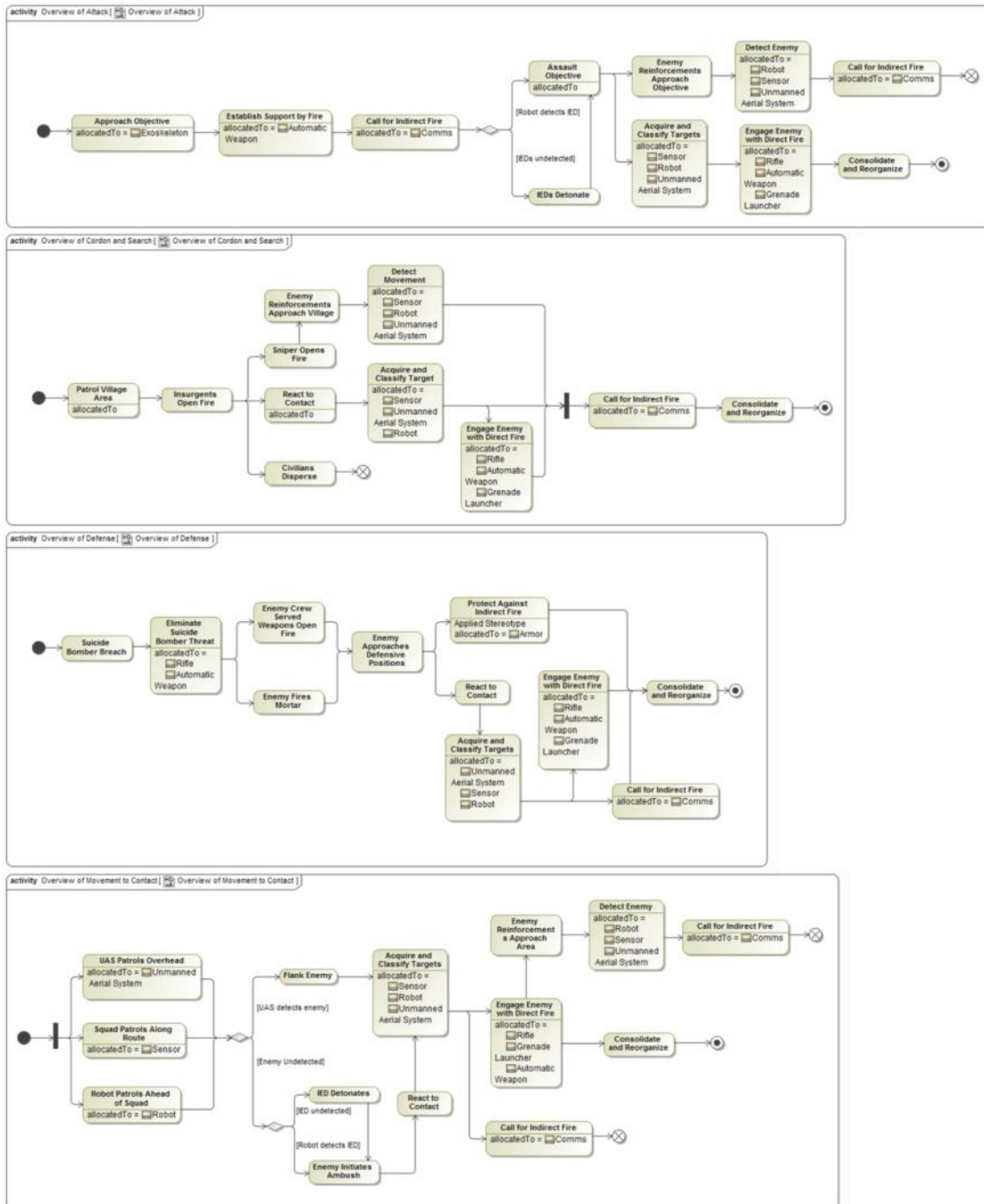


Figure 6.3. Activity diagrams for the attack, defense, movement to contact, and cordon and search use cases.

Sequence Diagrams (sd). Sequence diagrams reveal how part properties of a block or the blocks themselves interact with one another with operational calls and signals to produce emergent behavior. Unlike activity diagrams, the sequence diagrams indicate which blocks are invoking behavior by the types of messages they exchange. Lifelines are used to convey a participant in the interaction that corresponds to a part property or block. Each sequence diagram represents a type of interaction consisting of a collection of event occurrences that either send or receive messages or

start and terminate behavior. Figure 6.4 shows a Sequence diagram from the squad enhancement system example. Specifically, it shows the agent block elements of the agent-based simulation as lifelines at the top and the messages that are sent and received horizontally across lifelines; the arrows represent these messages. The event occurrences are labeled above each message arrow. The sequence of event occurrences executes from top to bottom; the vertical dotted lines represent the lifetime of the part or block during the interaction. Two types of combined fragments are shown within the diagram, par and loop. The par fragment is a subset of interactions separated by the dotted line that occur in parallel while the loop fragment occurs iteratively during a single execution of the interaction.

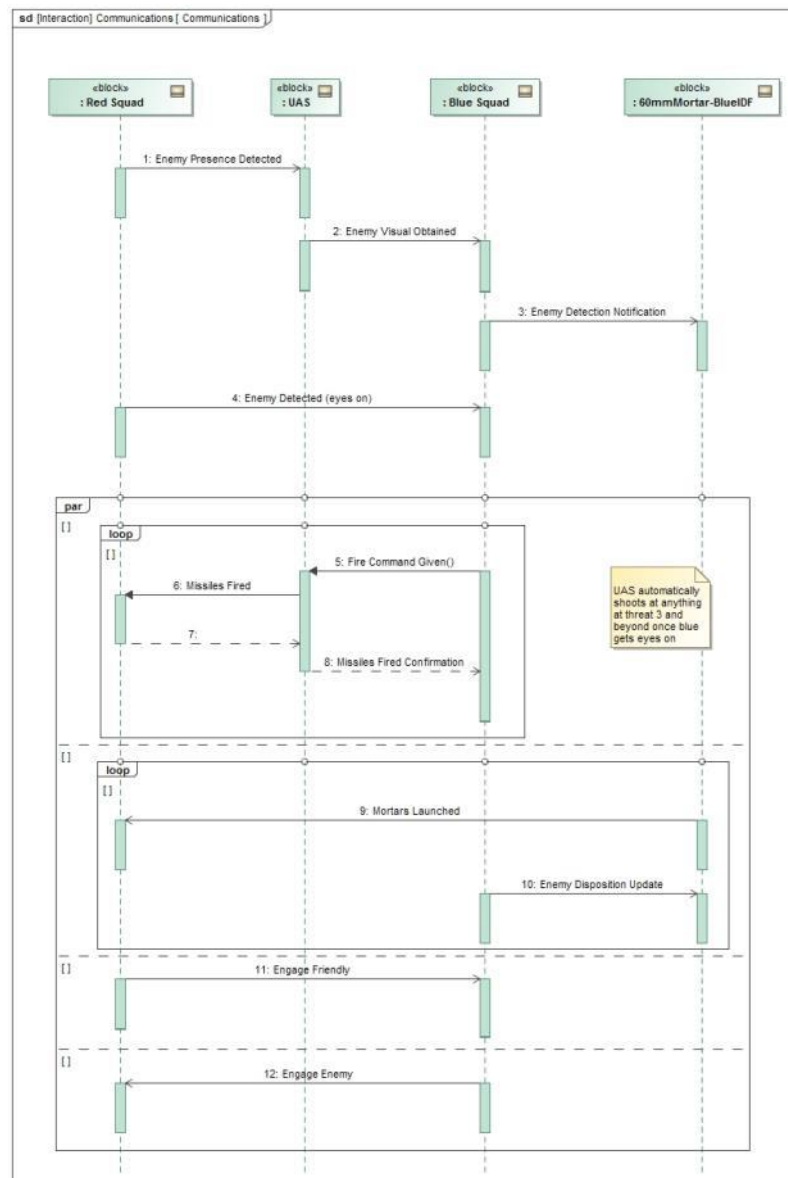


Figure 6.4. Sequence diagram showing the interactions between agents within the simulation model.

State Machine Diagrams (sm). The state machine diagram is a third type of behavioral diagram that displays how an instance of a block transitions between states in response to event occurrences. Systems and sub-components often have a defined set of states that it can exist in during the system operation. State machines diagrams allow modelers to express how event occurrences trigger a system to change from one state to another; only one state can be active at any one time. These diagrams are especially useful during the construction of an agent-based simulation model. Figure 6.5 shows a State Machine diagram for the blue infantry agents that represents the squad in the agent-based simulation model. The diagram classifies the behavior of the blue infantry agent within the simulation. The ovals represent states and the arrows represent the transitions with triggers that instantiate the state change. The large rectangle labeled “Combat States” represents a composite State Machine. When the composite state is active, then exactly one of its internal states is active; when the composite state is inactive, all of the internal states are inactive.

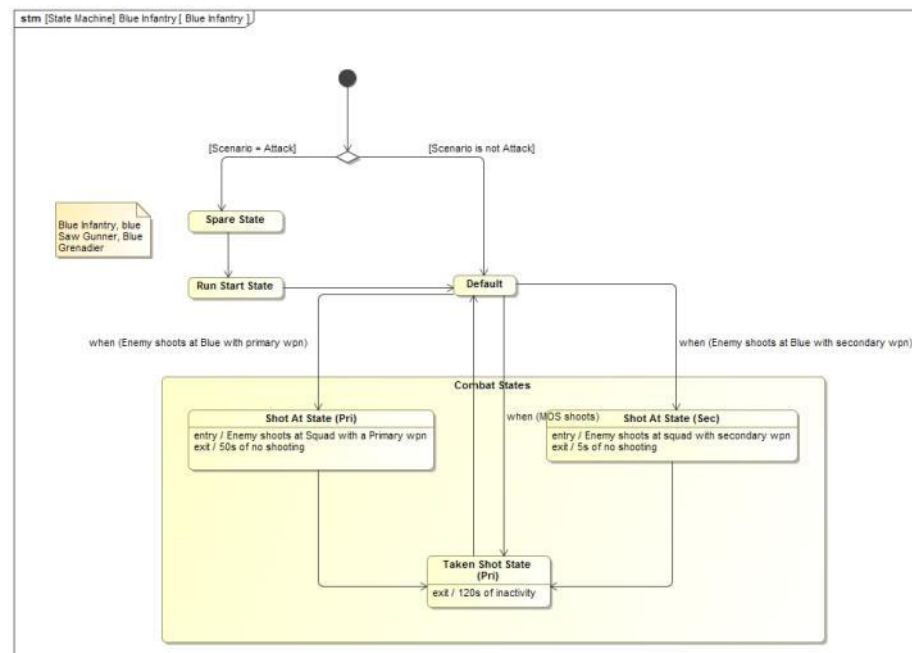


Figure 6.5 State machine diagram that classifies the behavior of an agent within the simulation model.

Parametric Diagrams (par). Modelers use parametric diagrams to express information about the system’s constraints. We can define constraint blocks to represent model equations and inequities that constrain or bind value properties of the structural system. Binding value properties allows systems engineers to impose fixed mathematical relationships on value properties that constrain the feasible system configuration. These constraints allow modelers to perform trade studies by comparing alternatives and identifying when system configurations become infeasible by violating the binding constraints. Constraints can also represent external models. Instead of a mathematical expression, the external model calculates the output results as another type of value property. Figure 6.6 shows a parametric diagram from the small squad enhancement system example. The diagram at the bottom left is a bdd that defines the constraint block for the weight and cost constraints. Each constraint has parameters that are binned to value properties in the parametric

diagram shown in the upper right of Figure 6.6; the dashed boxes represent blocks containing value properties binded by the constraint parameters.

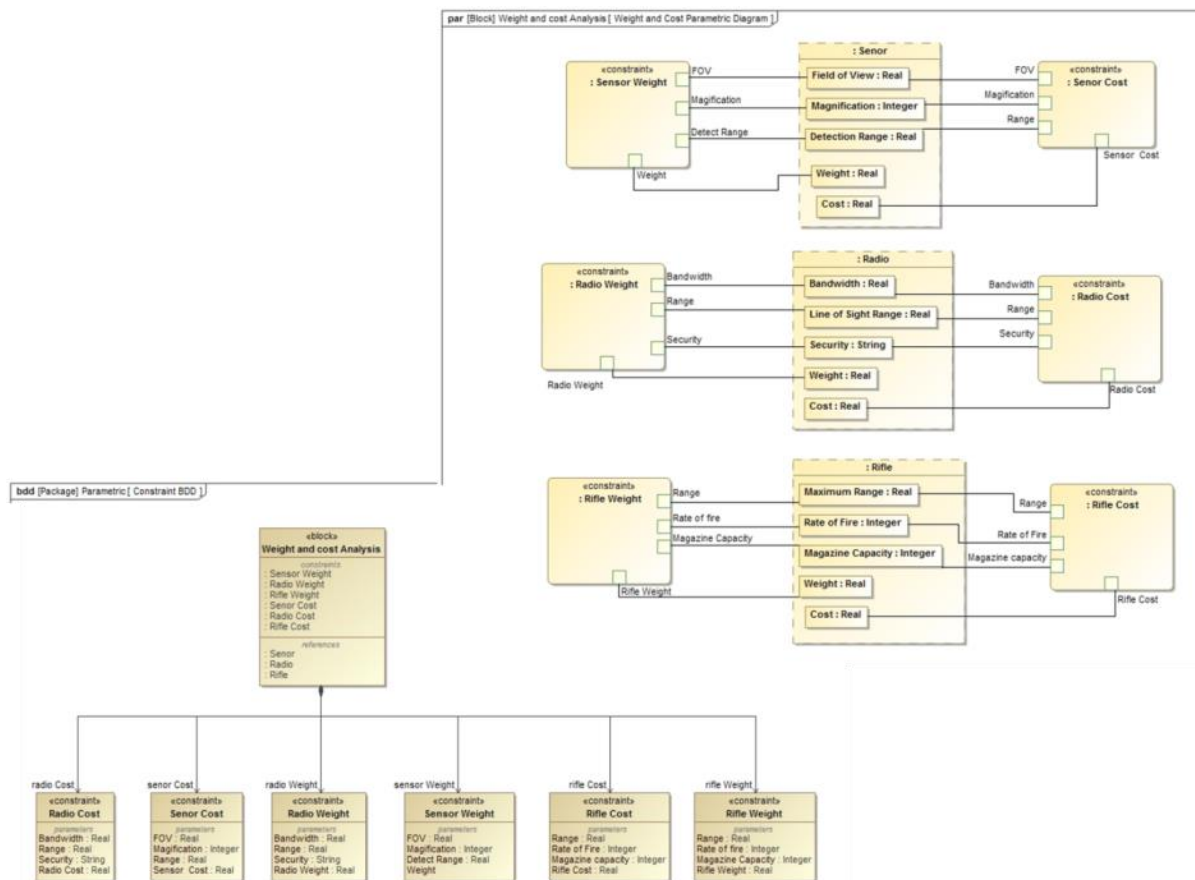


Figure 6.6. Parametric and block definition diagram of constraints.

Package Diagram (pkg). Packages are used to organize the integrated system model into different namespaces that nest different structural blocks and behavioral elements within a hierarchy of packages. Package diagrams convey the organizational structure of the integrated system model. Figure 6.7 shows an example of a Package diagram from the squad enhancement system example.

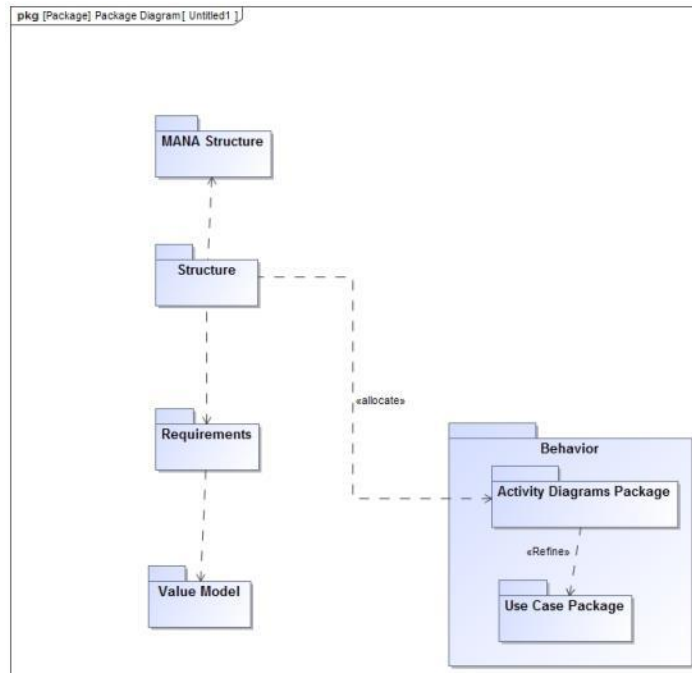


Figure 6.7. Package diagram for the squad enhancement system example.

Requirements Diagram (req). Requirements diagrams allow modelers to show text based requirements and the relationships between them and the system elements. The types of relationships include containment, trace, derived requirement, satisfy, and verify. The requirement diagrams allow modelers to show the requirements traceability to the system elements that depend on them. Figure 6.8 shows a requirements diagram for the emergent properties that our external simulation models evaluate.

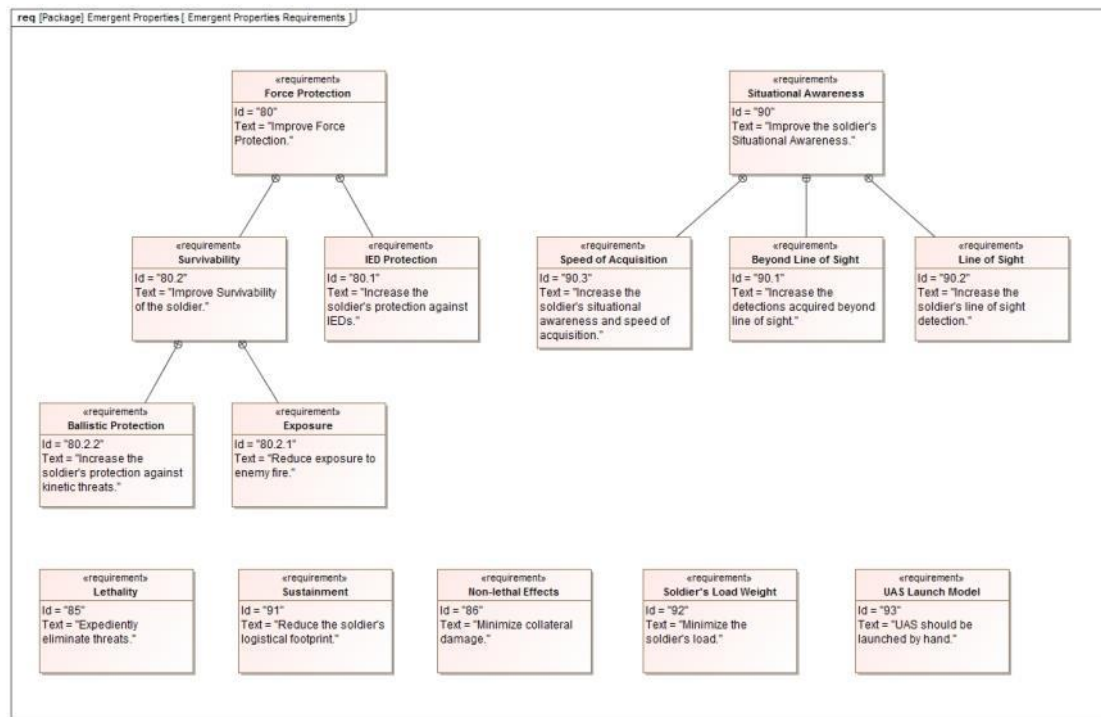


Figure 6.8. Requirements diagram for the squad enhancement emergent properties.

MBSE provides the way to perform conceptual modeling and allows system engineers to develop a wide variety of alternatives. The MBSE integrated system model is the conceptual model referred to in the data-driven approach illustrated in Figure 3.3. By expressing the conceptual model using the nine SysML diagrams, system engineers can configure systems before analyzing alternatives allowing them to easily redesign and rework new alternatives before they are frozen during the system lifecycle. The ERS architecture is a means for a community of users to perform conceptual modeling using the MBSE approach in order to engineer resilient systems that are flexible to change.

6.2. External Model Integration

A key limitation of SysML is that it is only descriptive in nature and cannot produce analytical results to inform system effectiveness. The parametric diagrams allow the modeler to incorporate mathematical equations that a tool can solve as a system of equations but they are limited to simple expressions. In order to achieve the full benefit of the MBSE approach the systems engineering community must also rely on external models that capture more sophisticated analysis across a wide variety of domains.

The types of domain models range from simple analytical equations and spreadsheet models to simulation models that capture the dynamic complexities of a system over time. Each model is an abstraction of reality that represents a unique viewpoint of the system within a domain of interest. Examples of these model domains include simulations that measure operational effectiveness, life cycle costing models, physics-based computational simulations, manufacturing models, and many more. Generally, the common state-of-practice is to analyze these models separately to gain insight into the domains or viewpoints they represent. In order to effectively analyze the trade-offs between these

various domains, we must integrate the models so that we understand the interrelations between the domains. Every model has unique inputs and outputs. In order to discuss how to integrate models, we must first understand how these inputs and outputs relate to a system design problem.

Model inputs can be categorized into two types. The first type represents the design parameters of the system; if the system is a vehicle, examples may include the fuel capacity, number of wheels, or weight. The Physical Architecture Design section of the Guide to the Systems Engineering Body of knowledge (SEBoK) defines these design parameters as local properties that are located in a single system element (BKCASE Editorial Board, 2015). The settings of the local properties define the system alternative configuration and are typically under the control of the systems engineer. The second type of model input is known as noise variables that represent threat or environmental uncertainties and are not under the control of the systems engineer; examples may include enemy force size and type, weather, terrain, or road conditions. The design parameters and noise variables are defined as value properties of a block element within the SysML integrated system model.

Systems engineers use model outputs to measure a system's performance, effectiveness, feasibility or any other life cycle consideration that pertains to the model's domain. Model outputs are generally used to understand what the SEBoK defines as properties which are meaningful only when attributed to the whole, not to its parts, otherwise known as emergent properties (BKCASE Editorial Board 2015). Properties that emerge from the arrangement and interactions of system elements can only be truly assessed during operational testing. During the early stages of design, system engineers rely heavily on simulation models when operational testing is not feasible because the system does not exist. An example of an emergent property for a cargo transport vehicle may be the time it takes to arrive at a destination in different terrain and weather environments. The time it takes a vehicle to reach a destination cannot simply be analyzed with a local property such as the vehicle speed. We must evaluate the vehicle configuration with all its local properties specified in order to evaluate its effectiveness in an environmental setting. Identifying the vehicle that has the shortest time of arrival will generally never be the desired alternative due to the multiple competing objectives involved with its design. Systems engineers must also consider the physical feasibility constraints, costs, human factors, manufacturability, and development schedules, just to name a few. Integrating model inputs and outputs that capture each of these previously mentioned considerations or domains will allow system engineers to effectively explore tradeoffs among the multiple objectives. For example, an operational simulation can inform designers of the system alternative effectiveness within a particular scenario context while a physics-based computational model can inform the alternative configuration performance or feasibility.

In order to evaluate a system alternative across multiple model domains, the system engineer must link the local properties of the system elements to the collection of model inputs for each domain model. The mapping of these system local properties to model inputs is often not direct. For example, an operational simulation may have an input parameter that represents the speed of a vehicle while a physics-based computational model may have inputs that represent the number and type of engines. In order to effectively integrate these models, we must understand how the number and type of engines affect speed. To establish these relationships we can collect data, develop look up tables, build other models, or make assumptions based on subject matter expertise. Translating system local/value properties to model inputs allows the systems engineer to evaluate alternatives across

multiple domains. Once the system local/value properties are mapped to model inputs properly, we can then explore ways at understanding their effects on model outputs.

The concept of a translator is shown in Figure 6.9. To the left of the diagram, value properties within the blocks (shown as dotted boxes) that have a direct mapping to a simulation model input have a binding connector that is directly linked to the external model constraint block. The value properties that are not directly mapped have a translator constraint block. The translator constraint block transforms value properties into the model input value that will represent the system configuration setting during the agent-based simulation model run. The right of the diagram shows a value model constraint block with binding connectors that link external models outputs to the inputs of the value model. See Section 10 for a discussion of the value model development and use.

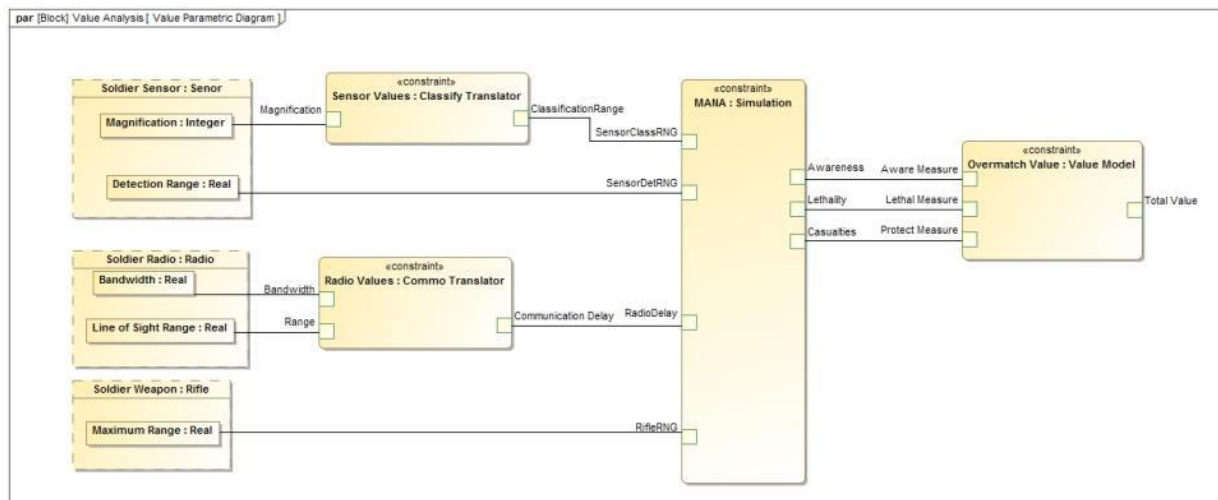


Figure 6.9. Parametric diagram of the agent-based simulation model and the translators.

The most effective way to determine the relationships between the local properties (model inputs) and emergent properties (model outputs) is to leverage the method of statistical experimental design. The field of design of experiments allows the analyst to identify which experimental factors or local properties effects an output of interest that represents an emergent property. Incorporating DOE methods within a MBSE design methodology allows for the exploration of a wider range of alternatives. As systems become more complex, incorporating DOE methods within the MBSE approach is a natural merger that can reveal key insights during the design of a system. Figure 6.10 shows an illustration of our MBSE methodology. In the center is the MBSE integrated system model. Each corner represents the DOE analysis for different domains; system element value properties are mapped to experimental design factors, experiments are performed on high performance computing clusters, an analysis is conducted to capture insights that are fed back into the integrated system model.

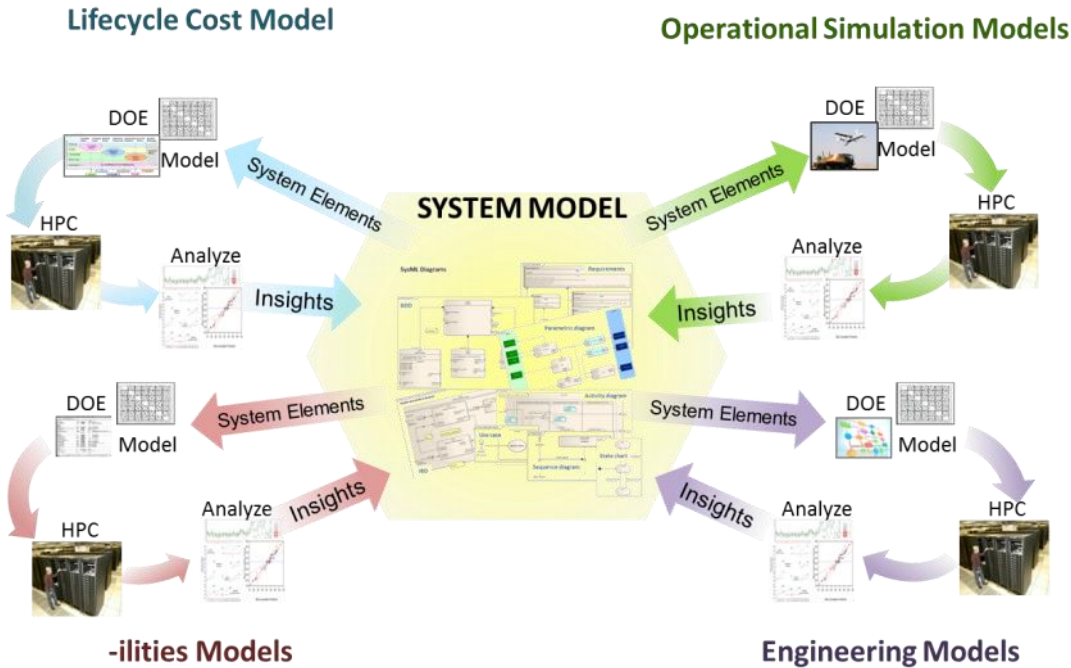


Figure 6.10. Experimental design insights that refine the integrated MBSE system model.

The insights we developed from DOE are primarily a result of fitting statistical metamodels that approximate the external models' behavior by acting as a surrogate to these models. Our next section discusses the concept of a metamodel and how we develop and use them to gain insights.

7. Building Statistical Metamodels using Simulation Experimental Designs

A statistical metamodel is an empirical model developed from either observational or experimental design data that relates a set of inputs to an output. (Grayson and Gardner 2015; Hastie et al. 2009). The model has the following general form:

$$Y = f(X) + E, \quad (1)$$

where Y is the outcome of the simulation model output, otherwise known as the response, X are the inputs, $f(X)$ describes the predictable or explainable variation of Y , and E describes the non-predictable or non-explainable variation of Y , otherwise known as the residual error. There are a wide variety of methods that fit metamodels that generally are suited for one of two purposes (Grayson and Gardner, 2015; Hastie et al. 2009; Kuhn and Johnson, 2013.) The first purpose is to understand the behavior of a process, system, or model by interpreting the effects of the inputs on the outputs (responses). In this case, we place a higher emphasis on estimating the parameters of the metamodel and its functional form. The second purpose is to predict the outcome (response) of a process, system, or model by using the metamodel as a function. In this case, we focus more on how the metamodel predicts and are less concerned about the form of the model. Finding the true model that fully characterizes a process, system, or model can be very difficult if not impossible. By building simpler models we can describe the

process, system, or model with functional forms that are useful for understanding or predicting behavior.

Fitting metamodels using observational data imposes problems that degrade the ability to fit a useful model; these problems include correlations in the input data, missing data, and not having enough data. The most preferred way to fit a metamodel is to use an experimental design that specifies the inputs and eliminates these problems. (Kleijnen, 2015; Kleijnen et al. 2005) In this section, we introduce the methods of design of experiments (DOE) with an emphasis on simulation experiments and discuss the types of metamodel methods used to understand and predict behavior. Our goal is to highlight the power of DOE and describe how it fits into our MBSE methodology.

7.1. Statistical Design of Experiments Introduction

A common state-of-practice when performing tradeoff analysis is to start with a baseline system alternative of a system configuration. Two typical methods for experimentation are to vary one factor at a time or to develop excursions from the baseline to see what happens. Varying one factor at a time does not allow us to identify system interactions or synergies. In the context of simulation experiments, an interaction is when a model input's impact on the output depends on the setting or value of another model input; this definition is different from the interactions associated with the system element interfaces. A positive interaction implies that model inputs complement each other and a negative interaction implies that model inputs substitute each other. Most complicated systems contain multiple synergies and understanding model input interaction translates into where these interactions may exist between the system elements. Examining excursions from the baseline usually means that the model inputs may be varied simultaneously; this may result in confounding effects, thus making it impossible to identify which model input caused the observed impact on the model output. To address these concerns, analysts use the methods of experimental design to untangle the effects of model inputs by eliminating the confounding between them and allow the opportunity to identify influential interactions (Sanchez and Wan 2009)

The statistical concepts of experimental design date back to the 1920s within the agricultural domain (Fisher, 1925) and since then have had applications in all areas of science. Analysts use DOE to help understand how the world works and have applied DOE principles primarily on physical experiments. As computers progressively became more powerful and accessible, experimental designs for computer simulations have become an active research area (Kleijnen 2015). A common goal when performing DOE is to identify a short list of influential experimental factors from a long list of many. In the context of system engineering and SysML, these experimental factors are considered value properties that are mapped to simulation model input parameters. In the previous section, we defined these model inputs as either design parameters or noise variables. Identifying which model inputs are the key design drivers and understanding how they affect the measures of effectiveness allow the systems engineer to make better design decisions. When there is a clear linkage between the model input parameters and the value properties in the MBSE integrated system model, we can gain insights into how each of the structural blocks of the system affects the performance of the emergent properties.

Performing a simulation DOE involves selecting an experimental design, running computational experiments, and fitting a statistical metamodel that approximates the behavior of the model. The experimental design, otherwise known as the design matrix, is the complete specification of the model input settings over a set of model runs; the columns in the matrix represent each model input and the rows are the settings for each experiment. At a minimum, the number of experiments must be greater than the number of design parameters. Additional experiments provide more data within the design space and capture more system configurations. (Kleijnen 2015)

After performing the experiments and assembling the output data, the engineer has a wide variety of metamodeling methods to choose from. The most common metamodeling method used to quantify the relationships between the model inputs and outputs is to fit a parametric polynomial function using statistical regression (Barton 1998). Other methods include neural networks, non-linear regression, Gaussian processing, sequential bifurcation, partition trees, boosted trees, bootstrap forests, and many more. (Grayson and Gardner 2015; Hastie et al. 2009; Kuhn and Johnson 2013) Generally, polynomial regression metamodels are excellent at describing individual model input impacts on model outputs while the other metamodeling methods mentioned earlier are better able to predict effectiveness by interpolating in-between simulated points. Ultimately, the analyst must understand which methods to apply for either understanding or predicting model behavior.

Developing a basic understanding spans across two extremes; on the one end, we want to gain insight into the mechanisms of a vague, ill-defined, or not-well-understood problem with limited, real-world data. On the other end, we want to perform detailed analysis on a verified and validated simulation model. No matter where we are in between these two extremes, there are a number of benefits of performing DOE that can help us understand a simulation model. These benefits include uncovering detailed insight into the model's behavior, allowing us to examine the modeling assumption implications, helping us frame the questions when we do not know what to ask, challenging or confirming expectation of directional model input effects and their relative importance, and uncovering problems of program logic. It is important to note that a model input's importance depends on the context of the simulation experiment; a model input may be influential in one mission context, but not in another. Our next section will focus on how we can gain insights into understanding the behavior of simulations using the polynomial regression metamodel and partition tree methods.

7.2. Understanding Complex Behavior

In order to reveal the benefits of using regression, we must first define the functional form of the polynomial regression metamodel. According to (Myers and Montgomery 2009), the second order polynomial model is the most common metamodel used to model real-world problems and has the following form:

$$y = \beta_0 + \sum_{j=1}^k \beta_j X_j + \sum_{j=1}^k \beta_{jj} X_j^2 + \sum_{i=1}^{k-1} \sum_{j>i}^k \beta_{ij} X_i X_j + \varepsilon, \quad (2)$$

where β_0 is the intercept term representing the mean of the data; β_j is the coefficient of the X_j term and represents a model input's rate of change or effect on the model outputs y when all other model inputs are held constant; X_j^2 is the quadratic term for the j^{th} model input, β_{jj} is the quadratic term's

coefficient; X_iX_j is the two-way interaction between the i^{th} and j^{th} model input, and β_{ij} is the coefficient of X_iX_j . The error term ε represents other sources of variation not accounted for by the model inputs.

The polynomial regression model provides readily interpretable parameter β coefficients that provide key insights into the model's behavior. The magnitude and sign of β_j, β_{ij} , and β_{jj} , express the nature of the model input's effect on the model outputs. The functional form of the metamodel is known as the response surface that we can visualize in two or three dimensions. For higher dimensions, we must view cross-sections of two model input parameters at a time of the response surface. In order to fit a second order polynomial metamodel, we must expand the design matrix into what is known as the regression matrix that includes columns representing the quadratic and two-way interaction effects. Figure 7.1 a design matrix expansion into a regression matrix.

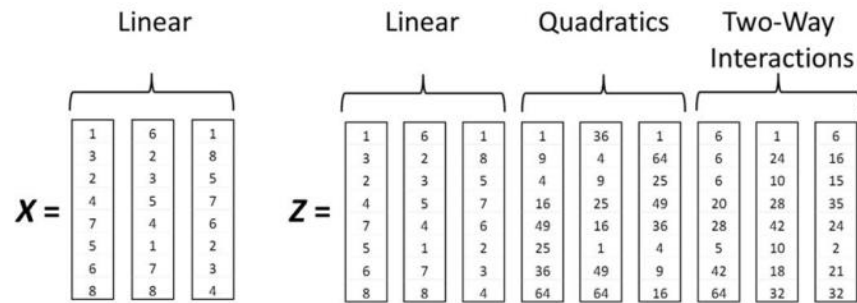


Figure 7.1. Design and second order regression matrix with three model input design columns.

The expanded regression matrix allows us to fit a second order polynomial metamodel using the method of least squares (Montgomery 2012). Figure 7.2 shows an illustration of how we can learn about the real world by performing simulation experimental designs and fitting polynomial metamodels that represent the response surface landscape.

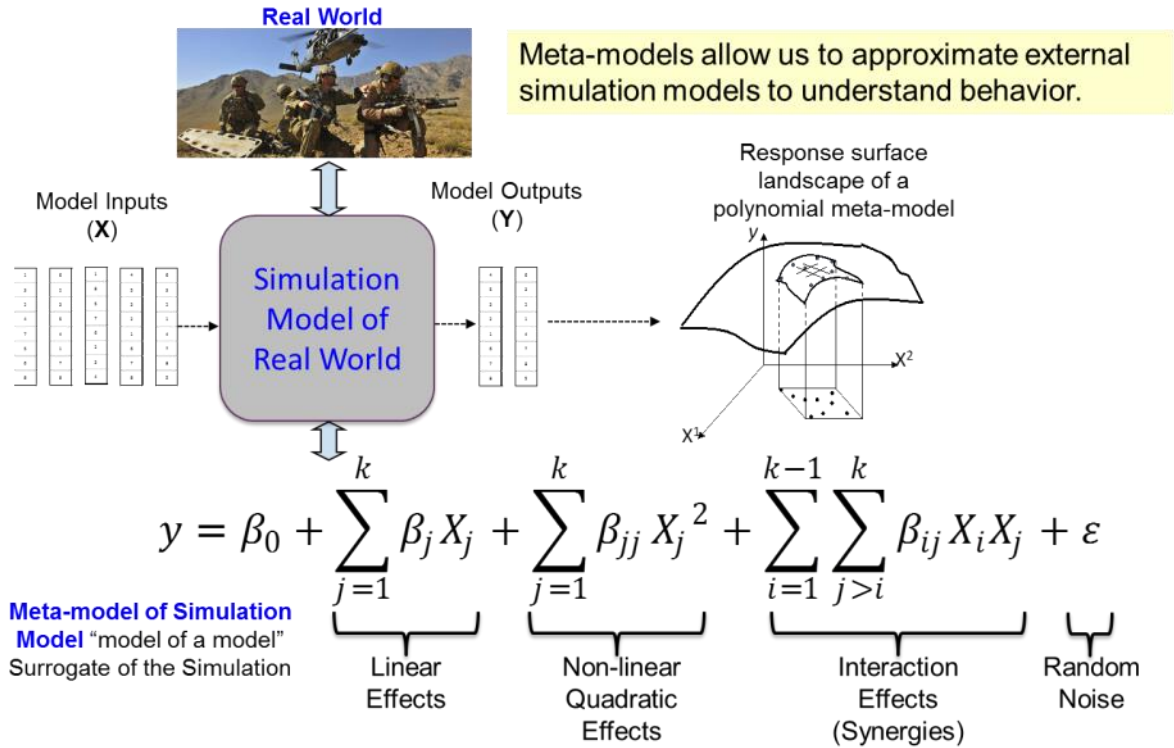
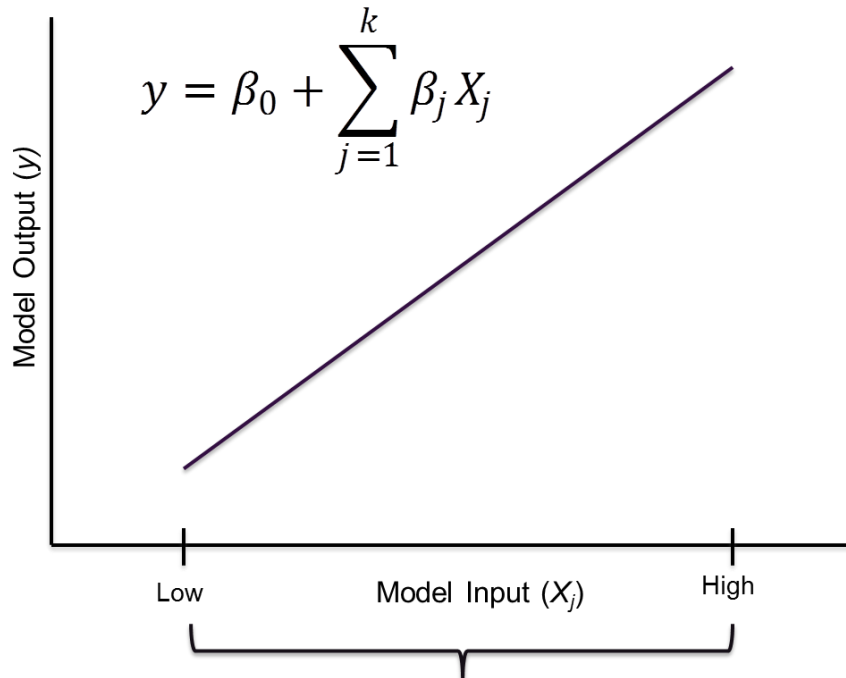


Figure 7.2. Metamodel development from a simulation experimental design.

Each of the linear, non-linear quadratic, and interaction effects have a unique interpretation that translates into insights the systems engineer can use to understand what influences a design decision.

7.2.1. Design Drivers

Understanding which X_j model inputs have an effect on the y model outputs provides key insights into which design parameters or noise variables drive the system's performance. We can identify these insights by examining the linear effects that estimate the slope or range of change on the y when we increase X_j . If the coefficient $\beta_j \neq 0$ then the j^{th} model input has an effect on the model output. These insights translate to the most important design parameters or noise variables that affect the emergent behavior. In a high-dimensional design problem, identifying the design parameters that are insignificant, where $\beta_j = 0$, is as important as finding the ones that are. These design drivers are responsible for determining the effectiveness of the system and should be carefully assessed during the design decisions. As we increase the level of the model input, a positive coefficient sign means that the model output will increase while a negative sign means that the model output will decrease. Figure 7.3 illustrates a positive linear effect.



Linear effects indicate how much resources in factor X_j must increase in order to increase/decrease output y .

Figure 7.3. Positive linear effect.

7.2.2. Synergies/Interactions

Identifying where there are interactions between system elements is an important endeavor while designing a system. These interactions are difficult if not impossible to find unless the system is evaluated in a mission context. Haphazardly selecting system alternatives or varying one model input at a time to evaluate performance provides the system engineer no way at clearly identifying where the interactions exist. Experimental design provides the means to identify interactions by specifying the model input settings for each experiment in order to clearly interpret each coefficient in the polynomial metamodel. The interaction term coefficient, β_{ij} , reveals a model input effect's dependence on the setting or level of another model input; a positive β_{ij} sign indicates that the two model inputs complement each other, while a negative β_{ij} sign indicates that they reverse each other. For example, the presence of a sensor and weapon type in a defense system may together result in vastly different effectiveness than the presence of each of them separately. Figure 7.4 shows the effect of an interaction term on the model output.

$$y = \beta_0 + \sum_{j=1}^k \beta_j X_j + \sum_{i=1}^{k-1} \sum_{j>i}^k \beta_{ij} X_i X_j + \varepsilon$$

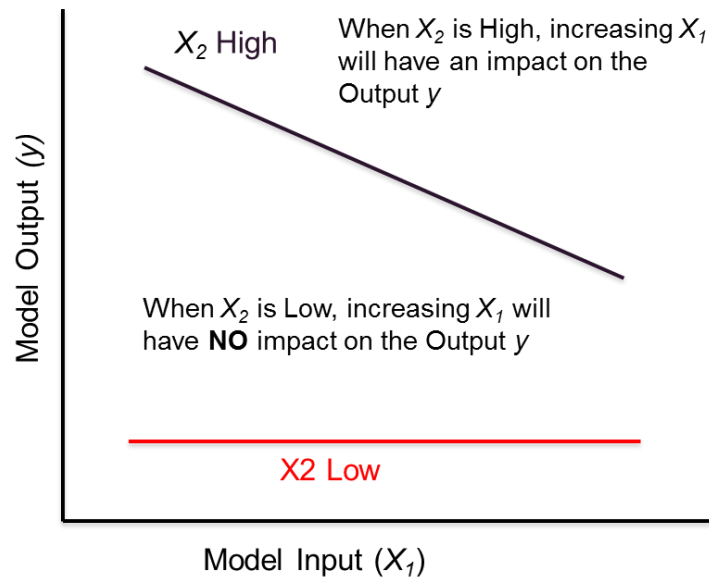


Figure 7.4. Interaction effect where X_1 and X_2 negatively reinforce each other.

7.2.3. Diminishing or Increasing Rates of Change

The β_{jj} coefficient describes a nonlinear trend that indicates a model input's diminishing or increasing rate of change on the model output. For example, as we increase a continuous model input we may identify a "knee in the curve" that indicates a point of diminishing returns with respect to a model output. These insights can save significant resources when we find model input settings where the return on the model output levels off. The effect on the model output is a result of the combination of the linear effect β_j and the quadratic effect β_{jj} . Figure 7.5 illustrates the quadratic effect and shows the four combinations of quadratic effect types that result from the coefficient signs.

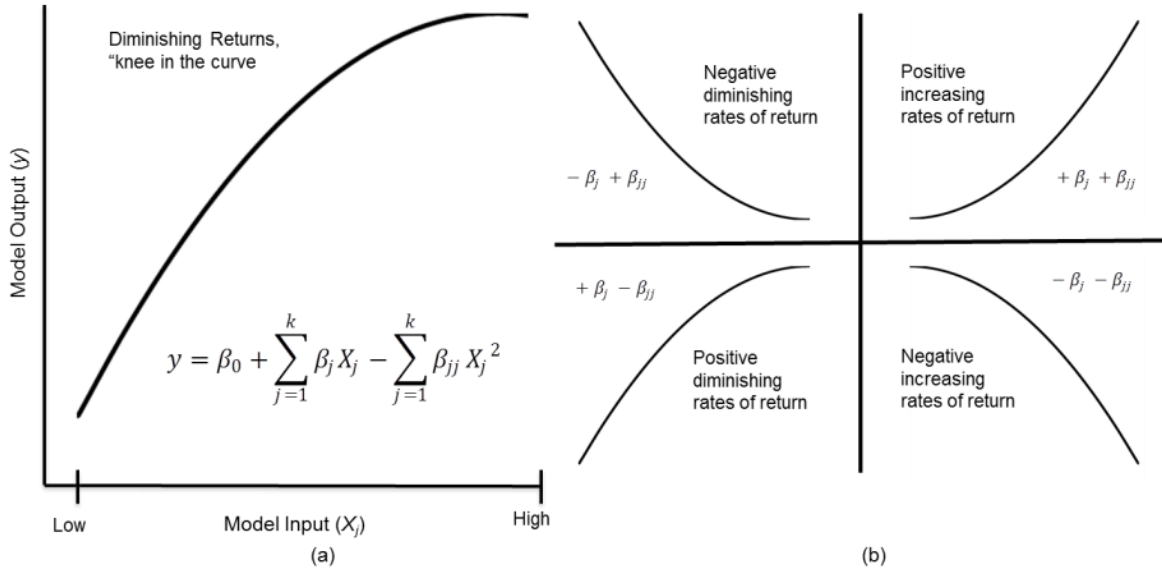


Figure 7.5. (a) Quadratic effect that reveals a “knee in the curve.” (b) Four types of quadratic effects defined by the combination of linear and quadratic effect coefficient signs.

7.2.4. Identifying Thresholds with Partition Trees

A significant limitation of the polynomial metamodel is that it can only approximate a smooth, nonlinear form and cannot identify a discontinuous step function that may exist within the response surface landscape. A step function or threshold is an area where the model output performance is vastly different from another area. Identifying the presence of a step function can lead to important insights when analyzing a system. For example, during the test and evaluation of the maximum allowable weight of a cargo parachute, the rate of decent may increase linearly or nonlinearly as the weight increases, up until a weight threshold. Once we exceed this threshold, the parachute will collapse and increase, or step up, the rate of decent by a significant amount. Identifying the weight threshold for a cargo parachute is, therefore, critical for those involved with its use.

Effective statistical methods that can identify thresholds are classification and regression trees, otherwise known as partition trees. A partition tree finds the optimal split in a data set where the distance between the two group means is the greatest (Loh 2011). Each split occurs at a model input setting that separates the model output data into two groups, one below and one above the split. The split occurs where the mean difference between the two groups is maximized. As a result, we can identify a model input threshold that has the highest impact on the model output. These splits can be interpreted as the minimum or maximum design parameter thresholds that achieve a desired level of effectiveness. Figure 7.6 shows a partition split of a notional response surface with an indicator function that steps up the response when x is greater than 0.5.

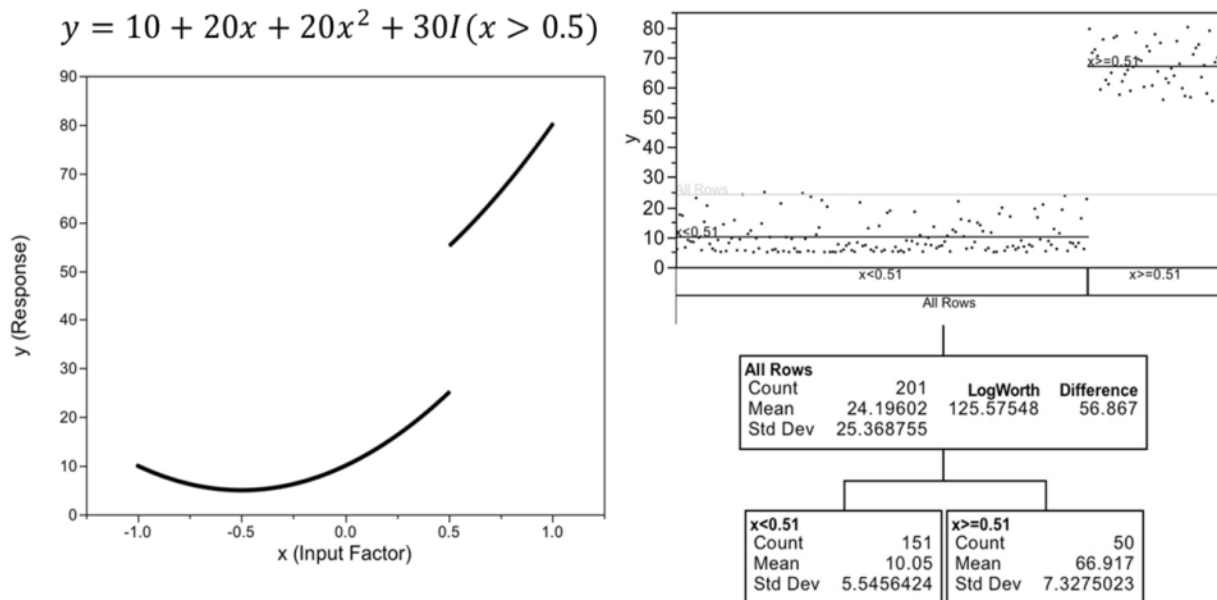


Figure 7.6. Partition tree split of a response surface with a step function where the model output performance is vastly different from another area.

7.3. Predicting Simulation Model Outputs

There are a number of methods that can fit predictive models where our primary focus is to accurately predict the outcome of a response given a set of inputs. Predictive models are more of a “black box” rather than a model with easily interpretable model parameters like the linear regression metamodel described in the previous section. In this section we focus on four methods, Stepwise Regression, Neural Nets, Boosted Trees, and Bootstrap Forest. We refer to (Grayson and Gardner 2015) for details on how these methods were implemented with the JMP® statistical software (www.jmp.com).

7.3.1. Stepwise Regression

Stepwise regression is way to find reasonably good linear regression models without having to examine all possible metamodels for a given set of inputs. Equation 2 from Section 7.2 shows the functional form of the second order polynomial metamodel. Because of the two-way interaction terms, the number of total terms increases exponentially as the number of inputs increases. As a result, fitting all possible combinations of models from an experiment with a large number of input columns becomes infeasible. Two types of stepwise regression methods are the forward and backward selection methods. The forward stepwise regression method starts out with a base model with only the intercept and adds the term not already included that explains the most variation. The algorithm continues to add terms until it reaches a stopping criterion. We use the cross validation criterion that compares the R^2 from the training set with the validation set. When the R^2 difference between the training set and the validation set begins to increase, the stepwise regression algorithm stops adding terms (see Section 7.3.5 for the definition of R^2). The backward stepwise regression algorithm works the same way as the forward

selection except that it starts with the full model and subtracts the terms that explain the least amount of variation.

7.3.2. Neural Nets

Neural networks are flexible non-linear models used for predicting complicated response behavior. The neural network is constructed using a series of hidden layers that have transfer functions as nodes. Figure 7.7 shows a diagram of two inputs, two hidden layers each with three nodes, and one output.

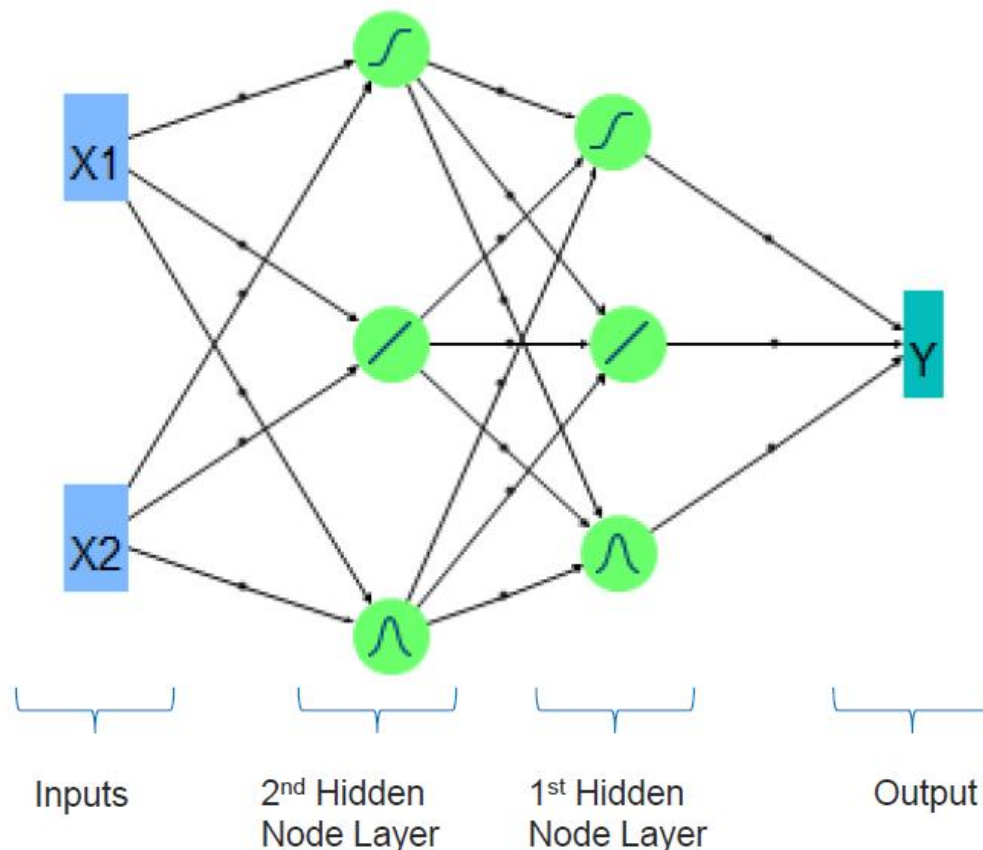


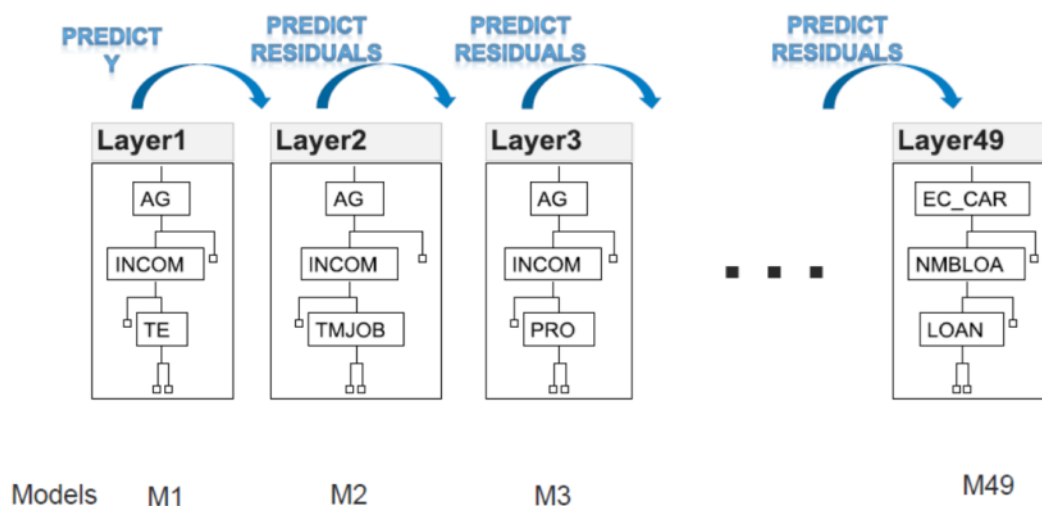
Figure 7.7. Neural net with two hidden layers.

Neural nets take a linear combination of the inputs into a single node to transform them to a value in accordance with a function (hyperbolic tangent functions, linear function, or Gaussian function). From the second hidden node layer shown in Figure 7.7 we get new values and take linear combinations of them into the first hidden node layer. The last step is to make a prediction by taking a final linear combination from the first hidden node layer. Each line segment and node in Figure 7.7 represents a different parameter in the model that results in a highly complicated non-linear equation developed using numerical optimization algorithms. Before running the Neural Net algorithm, the user must decide the number of nodes, layers, and transformation function types; the more nodes and layers there are,

the more time intensive and complicated the model becomes. Neural nets can be viewed as a weighted sum of nonlinear functions represented by the nodes in Figure 7.7. We can model a wide variety of relationships with complicated response surfaces; however, these models are very prone to over-fitting the data.

7.3.3. Boosted Trees

Boosted Trees is another predictive metamodeling method that uses a weighted combination of partition tree layers. These layers are models of nested if statements that split the data into branches in the same way as the partition tree method described in Section 7.2.4; each branch split occurs where the average response between branches is maximized. The first layer builds a small simple partition tree model from the data that explains as much variation as possible. The next layer uses the residuals from the first layer as the data to fit a second simple partition tree model; the residuals are the error difference between the data and the simple partition tree model predictions. The algorithm continues to fit simple partition models on the residuals from the previous layer for a specified number of iterations. The final model becomes a weighted sum of the individual models. Figure 7.8 is an illustration of 50 layer iterations, the final layer being the weighted accumulation of all model layers.



Final Model

$$M = M1 + \varepsilon \cdot M2 + \varepsilon \cdot M3 + \cdots + \varepsilon \cdot M49$$

Figure 7.8. Boosted tree layers.

7.3.4. Bootstrap Forest

The Bootstrap Forest method is similar to the Boosted Trees method in that it uses multiple partition trees to develop a predictive metamodel. For each tree, the Bootstrap Forest method takes a random sample of the inputs and builds a partition tree from this subset. The algorithm continues to build partition trees by resampling from the inputs for a specified number of iterations. The final model becomes the average of all the models. The benefits of the Bootstrap Forest method is that it allows the dominant input variables to be excluded from some of the partition tree iterations. In this way, all input

variables have a chance for selection. Figure 7.9 illustrates the Bootstrap Forest method of the average of 100 sampled partition trees.

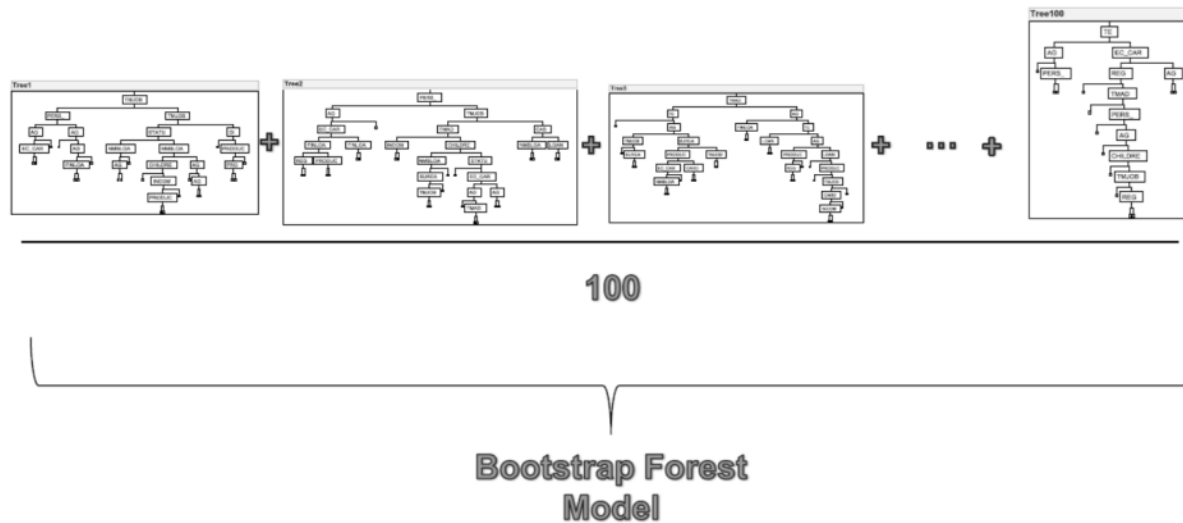


Figure 7.9. Bootstrap Forest method.

7.3.5. Model Comparison

A key concern when fitting predictive models is over-fitting the metamodel to the data. A metamodel may predict very well with the data used to fit the model but when we use other data to predict an outcome, the metamodel performs poorly. An effective way to test the accuracy of a metamodel is to hold data from the model building process and validate and test the metamodel with the other subset of data; this is known as the cross validation method. The data is partitioned into three sets, a training, validation, and test set. The training set fits the predictive metamodel. The validation set is used to select the size and complexity of the metamodel. The test set is used to evaluate how well the metamodel predicts data not used to fit or select the metamodel; this allows us to assess how well the metamodel will perform with a new set of input data.

It is important to fit a variety of metamodeling methods in order to find the best performing predictive metamodel. We applied the four methods described in the previous sections for each of our responses in the squad enhancement use case. We used two metrics to compare the metamodels in order to select the best performing model. The Root mean squared error (RMSE) and the R^2 metrics. To obtain the RMSE, we take the average of all the squared differences between the data and the predicted outcome. These squared differences are also known as the sum of the squared residuals. We then take the square root in order to get the average error in the metamodel. We derived R^2 using the following formula:

$$R^2 = 1 - \frac{SSE}{TSS} \quad (3)$$

where SSE is the sum of the squares of the residuals and TSS is the total sum of the squared differences between the data and the mean. R^2 is a number between 0 and 1, 1 means that the metamodel predicts the data perfectly and 0 means that the metamodel is no better than the mean of the response. We can interpret the R^2 metric as the percent of the data variation that is explained by the metamodel.

Generally, the R^2 metric is primarily used for linear regression models but in practice show similar results as the RMSE. Figure 7.10 shows the cross validation of the actual data versus the stepwise regression prediction plots for the *ATK_Aware* response.

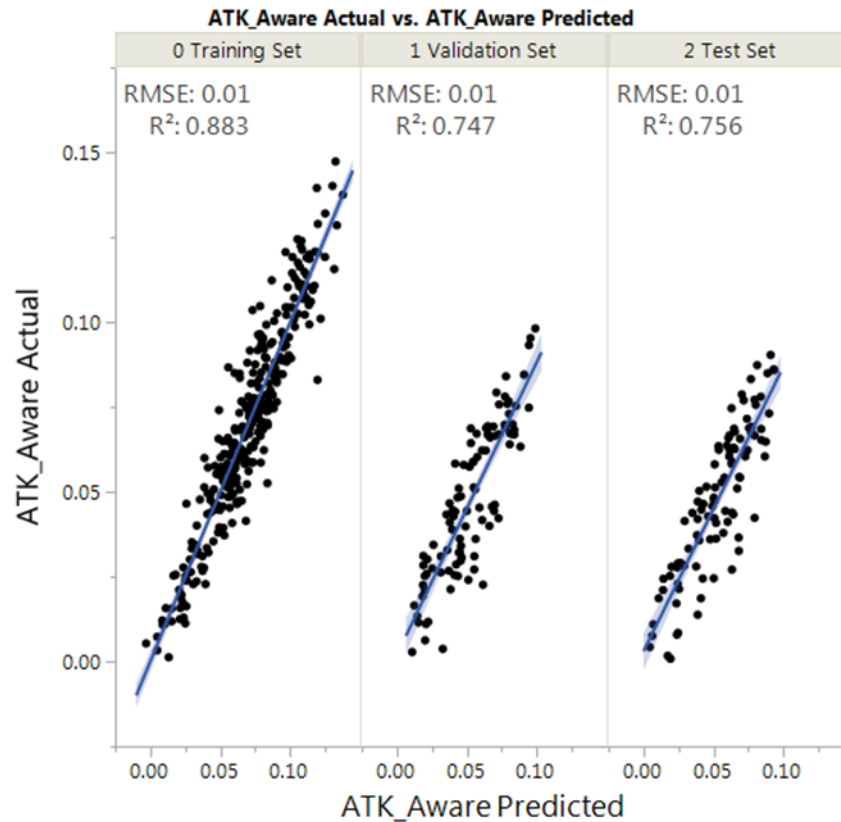


Figure 7.10. Cross validation comparison of the *ATK_Aware* stepwise regression training, validation, and test data sets.

Table 7.1 shows the RMSE results for all metamodeling methods we used to find the best predicting metamodel that pertain to the Attack scenario. The lowest RMSE determines the best performing metamodel. The Neural(1,1,1) is a Neural Net with one layer of three nodes, with a hyperbolic tangent, linear, and Gaussian function. The Neural(3,3,3) is a Neural Net with two layers of nine nodes with three hyperbolic tangent, three linear, and three Gaussian functions. For the *ATK_Aware* response, there is a tie between the Stepwise regression and Neural(3,3,3) RMSEs. We selected the stepwise regression metamodel because it is less complicated than the Neural Net metamodel. The R^2 metric shows the same results as the RMSE.

Table 7.1. The Root Means Squared Error (RMSE) comparison for the five predictive metamodels. Best performing RMSE are highlighted in bold.

Response	Boosted Tree RMSE	Bootstrap Forest RMSE	Fit Least Squares RMSE	Neural(1,1,1) RMSE	Neural(3,3,3) RMSE
ATK_LOS	20.1201	106.7295	23.2483	28.1781	25.8704
ATK_BLOS	0.0862	0.1878	0.0984	0.1756	0.1705
ATK_Survive	1.406	1.8606	1.4118	1.8377	1.8329
ATK_Protect	6.8189	8.4911	6.4104	8.8473	8.4804
ATK_Sustain	251.8785	285.4452	311.3128	263.8386	270.5063
ATK_Lethal	0.0437	0.0633	0.0471	0.0515	0.0473
ATK_Aware	0.013	0.0246	0.0122	0.0113	0.0112

To illustrate the forms of the metamodeling methods described above, Figure 7.11 shows each metamodel type for the *Awareness* response from the small squad enhancement use case. The stepwise regression metamodel shows a smooth polynomial function without any quadratic terms. We can clearly identify the highest impact model input as the *SensorClassifyRNG* due to the magnitude of the slope. The Boosted Tree and Bootstrap Forest methods show a jagged form that result from the nested if statements. The Neural Nets have a highly complicated non-linear form that can fit through the data very effectively. The Boosted Tree metamodel has the lowest RMSE and is therefore the best performing predictive metamodel.

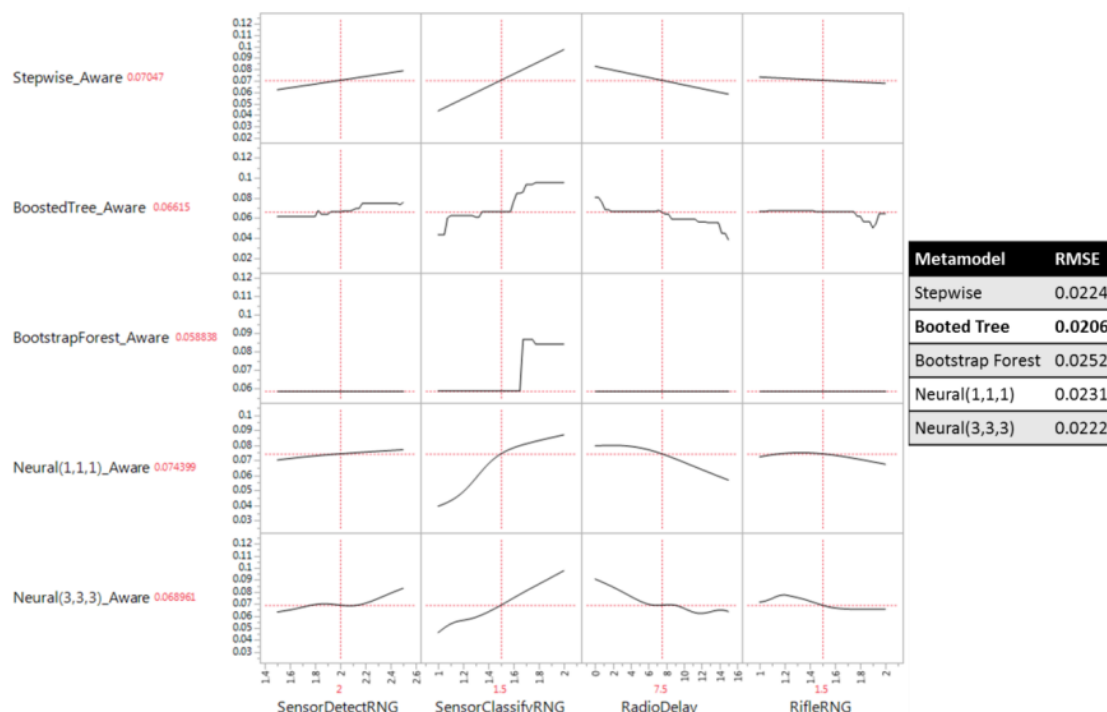


Figure 7.11. Vertical cross sections of the Stepwise Regression, Boosted Tree, Bootstrap Forest, and Neural Net metamodels and their Root Means Square Error (RMSE) values.

Comparing the metamodels in Figure 7.11 allows us to understand each method's individual benefits. The Stepwise Regression metamodel provides coefficients we can interpret to include two-way interactions, when they are present, to help our understanding of the model's behavior. The Boosted Trees, Bootstrap Forest, and Neural Nets are better suited for predicting as we can see by their forms shown in Figure 7.11. Now that we have an understanding of the metamodeling concept, we now will discuss the types of experimental designs available to perform simulation experiments.

7.4. Experimental Design Types

There are many types of experimental designs used for different purposes. The type of design used depends on a number of different considerations; some of these include: the number of potential model inputs, the number of experiments in the design matrix, the types of metamodels that will be fit, and the assumptions about the form of response surface; for example, if we experiment using only the low and high setting of the model inputs we have no idea what happens in-between and therefore must assume that there are no existing non-linear quadratic effects. The number of factors and the response surface complexity assumptions may be the two most important considerations. If we have a small number of potential model inputs and we can assume that there are no higher-order terms (i.e. the response surface is linear), then we can use full-factorial, two-level designs to identify which of the potential factors are important. The number of design points needed to perform a full-factorial design increase exponentially as the number of potential factors increase; therefore, we may need to use other types of designs that require fewer design points. (Sanchez 2015; Sanchez et al. 2014; Sanchez and Wan 2012)

DOE is particularly useful in simulation studies because of the large number of potential factors, the complex response surfaces that are involved, and the user can control the experiments without the need to block confounding effects typically required during physical experiments. Some of the differences between physical and simulation experiments are listed in Table 7.2.

Table 7.2 . Differences between physical and simulation experiments.

Characteristic	Physical	Simulation
Number of factors	Few	Many
Number of levels	Few	Many
Number of responses	Single	Multiple
Error variance	Homogeneous	Heterogeneous
Presence of interactions	Negligible or limited	Important and complex
Error structure	<i>Independent, identically distributed</i> Normal	Complex structure
Response surface form	Linear	Nonlinear

Because of the complicated nature of simulations, there are a number of newly created experimental designs that allow for the analysis of these types of problems. Before we discuss the types of experimental designs within the field of DOE, we first will explain two key design characteristics that are relevant to analyzing simulations—correlation and space-filling characteristics.

7.5. Correlation and space-filling design characteristics

Least squares estimation is the most common method used to estimate the β coefficients. The precision of these estimates depends upon the correlations among columns of the regression matrix. In order to ensure that model input effects are not confounded with other effects, the design should minimize the correlations among the columns within the regression matrix. If a model input correlates well with a model output but has a high correlation with another model input, then we cannot tell for certain which model input contributes to the observed change in the model output variable. To demonstrate the impact of correlation of the coefficient estimates, Figure 7.12 illustrates a design with two columns and its correlation matrix that includes the expanded quadratic and two-way interaction term. The figure shows a notional “true model” of the response surface landscape in the upper right that in reality we never know with complete certainty. Because there are correlations between the higher order effects (quadratics and two-way interactions) the coefficient estimates of the model terms are significantly different than the true coefficients for the majority of the polynomial models fitted (as noted in the red color).

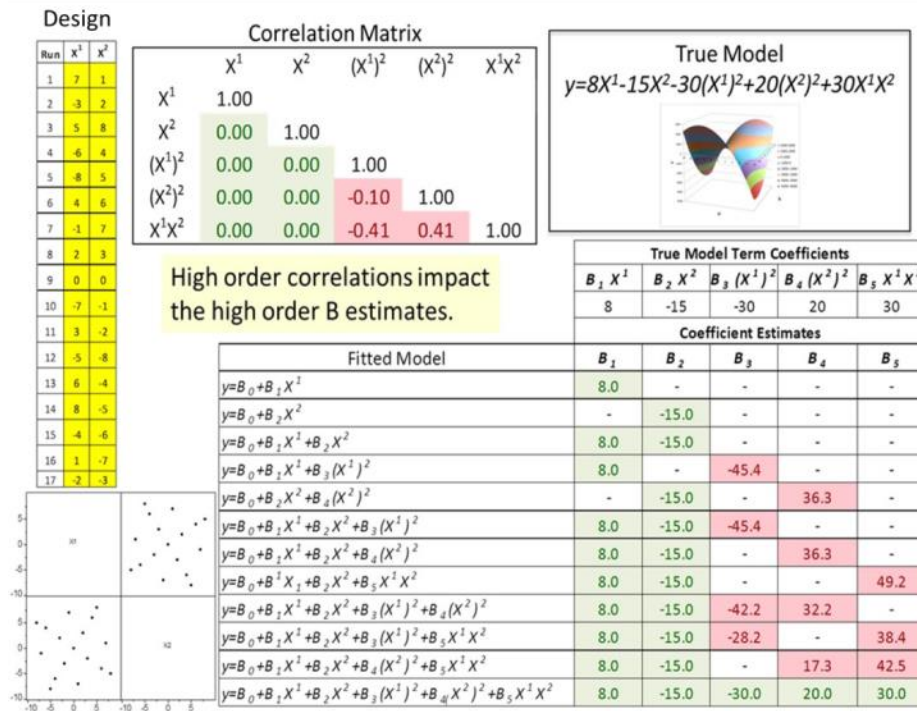


Figure 7.12. Impact of high correlations on the metamodel estimates. Within the fitted model matrix, red indicates a deviation from the true model coefficient, green indicates an accurate estimate.

In the example shown in Figure 7.12, a design that has no correlation among the columns will result in coefficient estimates that equal the true model coefficients. In a high dimensional simulation experiment, the number of polynomial terms will be significantly high. Therefore, in order to guarantee that our coefficient estimates are not confounded we desire to utilize experimental designs that minimize correlations among the columns in the regression matrix.

Correlation also impacts the length of the confidence interval around β_j , making it harder to

identify the true impact of a model input on the model output. Figure 7.13 demonstrates how correlation inflates the variance of the estimates by varying the angle between two vectors, X^1 and X^2 , from 1 to 90 degrees (a correlation of 0 is analogous to two vectors that are orthogonal, with 90 degrees between them). Each of the vectors has a unit length of 1 and is anchored at the origin. Assuming that the variance of the response (σ^2) is constant, the variance of the estimates can be determined analytically (Montgomery 2012). We created the graph in Figure X by rotating X^1 from 1 to 90 degrees and evaluating $\text{var}(\hat{\beta})$. We can see from Figure 7.13 that when the angle between two vectors is less than 50 degrees, the variance of the estimates is inflated significantly.

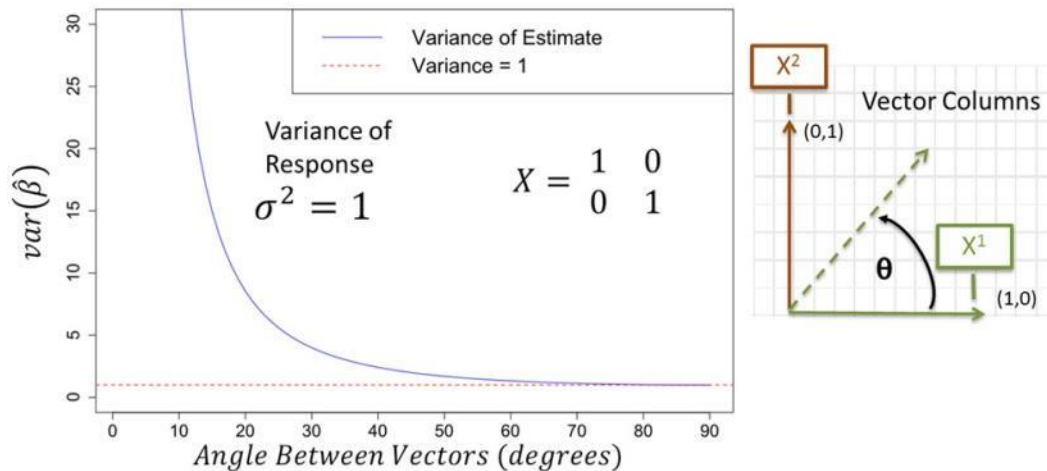


Figure 7.13. Impact on the coefficient estimate variance as the angle between vectors increases. The variance is inflated as the angle between vectors approaches zero. A 90 degree angle between vectors is analogous to 0 correlation.

Space-filling is another design characteristic that is important during simulation experiment studies. The amount of coverage in between the low and high settings across all model input dimensions is what defines a design's space-filling characteristic. Figure 7.14 shows a comparison between a traditional design that only experiments at the low, middle, and high setting and a space-filling design that covers the entire design space. Both designs have zero correlation between model term coefficients but the traditional designs do not explore the interior of the design region as well as the space-filling design.

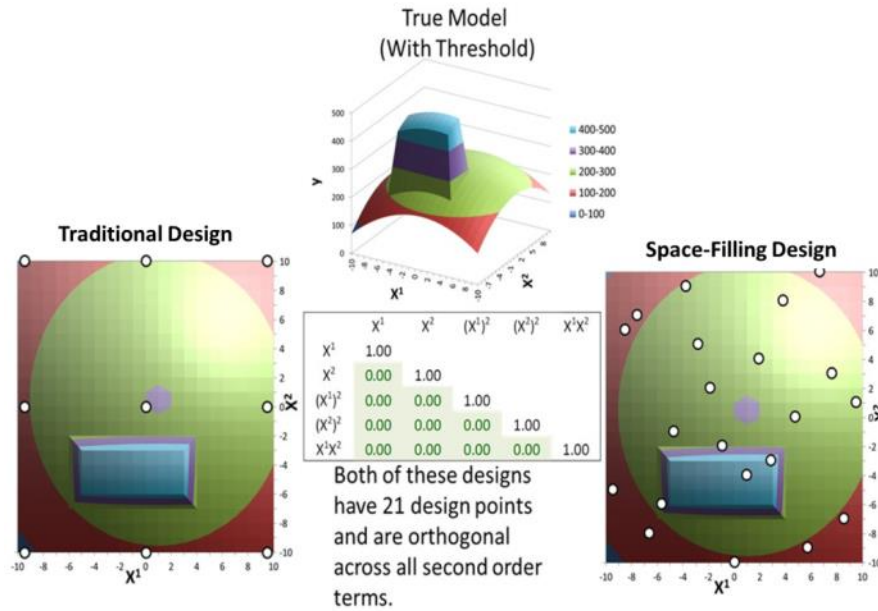


Figure 7.14. Two-dimensional projections of a traditional and space-filling designs, overlaid onto a contour plot of a true response surface with threshold.

7.6. Traditional and Space-Filling Designs

There are a number of traditional experimental designs that experimenters use for physical experiments. Two-level full-factorials designs are used when there are a reasonable amount of design columns and we can assume that there are no interesting behavior that occurs in-between the low and high setting of the design region. The number of experiments needed to perform a full-factorial design is 2^k where k is the number of experimental factors. To alleviate the burden of the high number of experiments, we can use fractional factorial designs but as a result, we cannot understand fully the impact of all the two-way interactions that may exist. When we need to understand how something behaves in the interior of the design region we can use Central Composite Designs (CCD) that supplement fractional factorial experiments with additional “star” points and a center point. The number of CCD experiments grows by a factor of $2^k + 2^k + k$ and very quickly becomes inefficient.

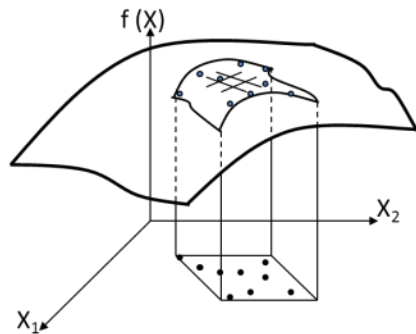
Computer-generated optimal designs are often used when traditional designs are not applicable. For example, when there is an irregular experimental region that has factor constraints, qualitative factors, and/or if we want to fit a nonstandard model that excludes a subset of quadratics or interactions (Myers et al. 2009). Two of the most common types of optimal designs are the D-Optimal that minimizes the determinant of the covariance matrix and the I-Optimal design that minimizes the average prediction variance; both for a pre-specified metamodel (usually a main effects model, with constant variance).

Space-filling designs are better suited for identifying unknown response surfaces, where multiple complex forms and thresholds are possible (Myers et al. 2009). The most common type of space-filling designs is the Random Latin hypercube (RLH). Figure 7.15 defines the RLH where n is the

number of experiments or design points, k is the number of experimental factors and $f(x)$ is the response.

The Latin Hypercube (LH)

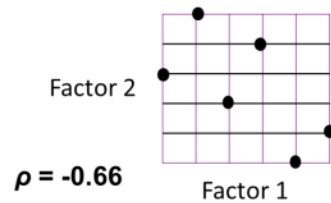
- In its basic form, each column in an n -run, k -factor LH is a random permutation of the integers $1, 2, \dots, n$.
- The n integers correspond to levels across the range of the factor.
- Design points are typically spread over the factor ranges: good for exploratory purposes.



A 6-run, 2-factor design

Factor 1	Factor 2
1	4
5	1
6	2
4	5
3	3
2	6

Pairwise projection



Random LH designs often result in correlations among factors.

Figure 7.15. Definition of the Random Latin Hypercube.

As indicated by Figure 7.15, a significant limitation with the RLH is that they generally do not minimize the correlations between the columns in the regression matrix and as a result, the coefficient estimates of the polynomial metamodel are confounded. (Hernandez et al. 2012) addressed the RLH correlation problem by constructing a nearly-orthogonal Latin hypercube (NOLH) with a mixed integer program optimization algorithm were the maximum absolute pairwise correlation between columns is less the 0.05.

Simulation experiments often have a mix of model input types. Continuous model inputs range between a low and a high setting, while discrete and categorical model inputs have a predetermined number of levels. The discrete levels have a numeric meaning, while each categorical level represents qualitative categories. LH designs are useful for the exploration of continuous factors because they provide insight throughout the experimental region but by themselves cannot effectively handle discrete or categorical experimental factors, because of the increase to the correlations due to rounding. Experimenters use orthogonal arrays when exploring discrete and categorical factors. (Rao 1945) introduced orthogonal arrays in order to ensure that qualitative factors are not confounded with each other. A significant limitation of orthogonal arrays is that the number of experiments needed for a moderate number of experimental factors is too large. For example, an orthogonal, full-factorial design, with 10 discrete factors, each with 10 levels, requires 10 billion experiments, making them extremely inefficient. To address the inefficiencies of orthogonal arrays, (Vieira, Jr. et al. 2011) developed known as Nearly Orthogonal/Balanced (NO/B) designs in order to explore all types of factors simultaneously

(continuous, discrete, and categorical) in a reasonable amount of experiments. For an in-depth review of the different types of experimental designs used for simulation studies, see (Kleijnen 2005).

7.7. State-of-the-art space-filling designs

In order to understand the complex nature of a system design problem, we must be able to detect the system drivers and understand how they impact the system effectiveness. Computer simulations and DOE enable us to model a system by simultaneously exploring numerous model inputs that may affect the complex nature of multiple simulation model outputs. When these model inputs are mapped to system design parameters and noise variables we can identify the ones responsible for determining the effectiveness of the system. These experiments are critical in the early phases of the system design process, when there is little information and no existing system. Simulation model outputs often have complicated, high-order, response surfaces that may include thresholds or step functions in different regions of the experimental space. The simulation analyst needs experimental designs that can best capture the significant system drivers, thresholds, synergies/interactions, and the model input's diminishing or increasing rates of return with respect to multiple model outputs.

A new class of space-filling designs, known as the 2nd Order NOLH and NO/B designs developed using a genetic algorithm combines the benefits of near orthogonality (minimal correlations), the space-filling characteristic, and the ability to experiment with continuous, discrete, and categorical experimental factors simultaneously (MacCalman 2013). Figure 7.16 illustrates how these new state-of-the-art designs contribute to the body of knowledge of the space-filling and second order design domains.

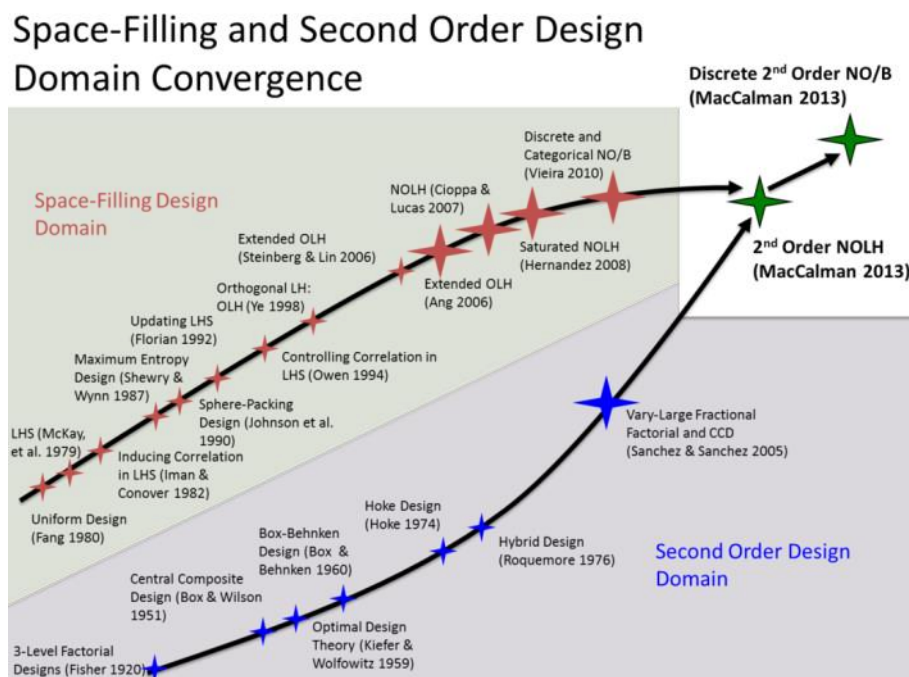


Figure 7.16. Space-filling and second-order design domain convergence; see (MacCalman 2013) for the embedded citations.

(MacCalman 2013) developed a freely available custom design generator to construct 2nd Order NOLH and NO/B designs for simulation studies that can be downloaded at <http://harvest.nps.edu/software.html>; Appendix B contains the custom design creator user manual.

7.8. Use Case Experimental Design

To perform the experiment, we utilized three custom built space-filling designs with 38 columns that represent each of the MANA model inputs. The design used for the training set had 300 rows, while the design used for the validation and test sets had 100 rows; each row represented the model input settings for one simulation experiment. Because the MANA simulation is stochastic, we performed 100 replications for each simulation experiment, for a total of 50,000 runs. After performing the experiments on a high performance computing cluster and post-processing the output data, the design matrix and model output columns were imported into a statistical package to fit our surrogate metamodels.

8. Technical Requirements for High Performance Computing Clustering

In order to outline the technical requirements necessary to perform simulation experimental designs, we developed an IDEF0 functional diagram that highlights the nine major activities with each of their input and output requirements. An IDEF0 diagram shows a box for each activity with arrows representing the inputs (left arrow), outputs (right arrow), mechanisms (bottom arrow) that perform the activity, and the controls (top arrow) that set the conditions, such as guidelines, procedures, or standards. Figure 8.1 shows the major activities starting with identifying the tradable/decision variables from our SysML diagrams and ending with the simulation output data and statistical metamodels that are imported into the ERS tradespace visualization environment.

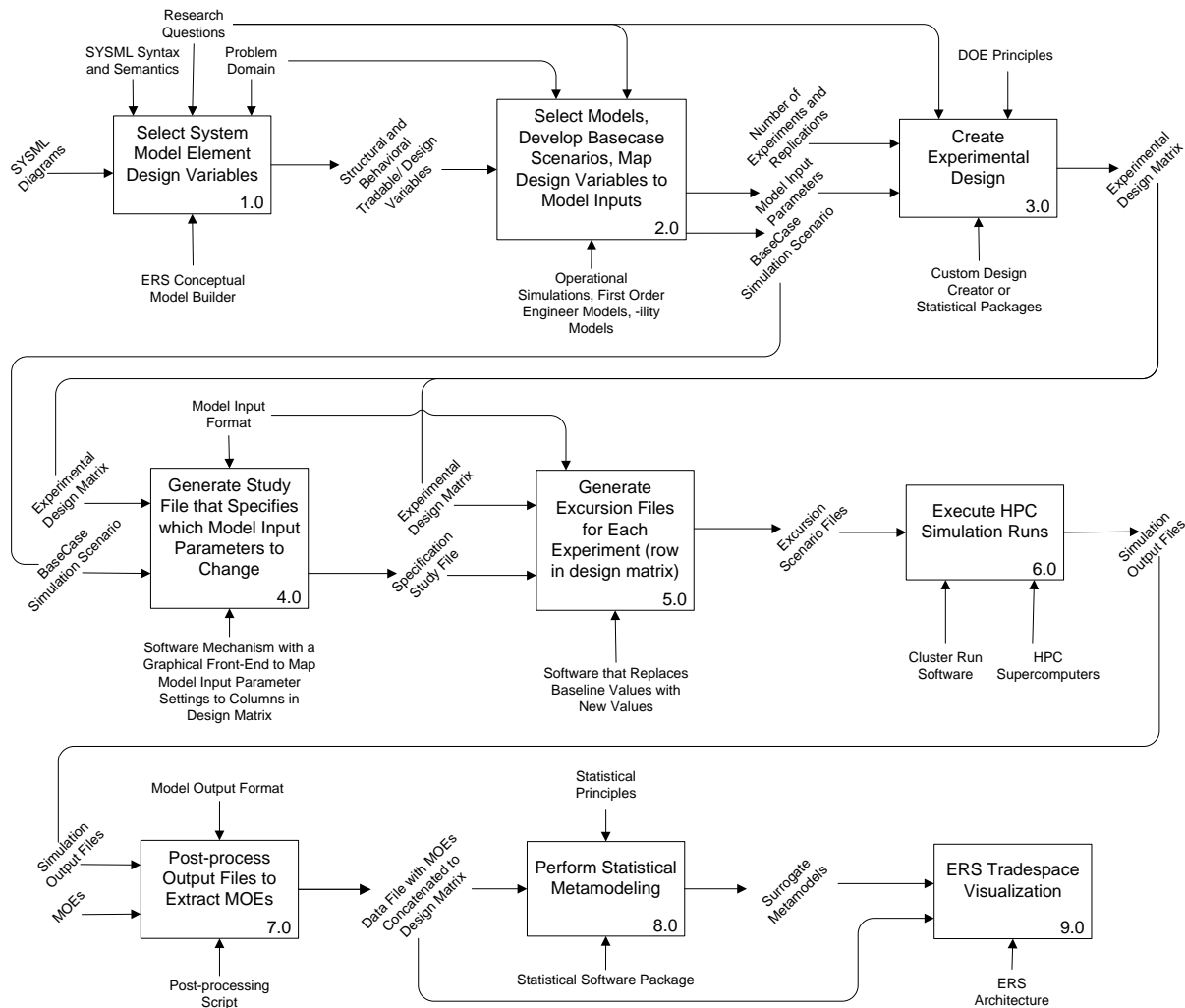


Figure 8.1. IDEF0 Functional Diagram outlining the activities and input/output requirements necessary to perform a simulation experimental design study.

The next subsections describe in more detail each of the nine activities depicted in Figure 8.1.

8.1. Select System Model Element Design Variables

A principle artifact of the MBSE paradigm is the “System or Reference Model” that represents the system of interest, implemented using an MBSE Tool. This model captures all design specifications throughout the system lifecycle and uses a language to express different views of the system. We assume that the Conceptual Model within the ERS Architecture will be what MBSE practitioners refer to as the “System Model,” that uses the nine diagram types of the System Modeling Language (SysML). Our first activity involves selecting the design variables from the conceptual model and defining the experimental design region we intend to explore with our external models. The inputs to this activity are the value properties from the structural block diagrams of the integrated system model. The controls that govern the activity are the SysML syntax and semantics, the research questions, and the problem domain. The

mechanism to perform this activity is the MBSE modeling tool and the outputs are the value properties representing the tradable variables to explore in the experimental design study.

Select Design Variables: Once the Conceptual Model is built in ERS, the analyst may want to analyze the performance/effectiveness of the system and identify the most important design variables, otherwise known as tradable variables. These variables can be structural and behavioral system model elements defined in the conceptual model and viewed in different SysML diagrams. Structural elements are viewed using the Block Definition and Internal Block Diagrams while the behavioral elements are viewed using the Use Case, Activity, Sequence, and State Transition Diagrams.

Define Design Region: Each variable should be classified as either a continuous, discrete, or categorical type. The analyst must define the experimental design region by selecting the low and high settings for the continuous variables, the number of levels for the discrete variables, and the number of categories for the categorical variables.

8.2. Select Analytical Models, Develop Baseline Scenarios, and Map Design Variables to Model Inputs

The inputs to this activity are the tradable variables that will be investigated during the experimental design study. The controls that govern the model selection are the problem domain and the research problem. The mechanisms that perform this activity are the models and simulations that will be used in the study. The outputs of the activity are the model inputs and the baseline simulation scenario. In order to effectively perform a simulation experimental design, all models must have the ability to modify their input parameters programmatically and execute an instance via a command-line without a Graphical User Interface (GUI) or human in the loop.

Select Analytical Models: We assume that the analyst wants to analyze the system using different external operational simulation models and first-order engineering models. Each model examines a different aspect of the system and has their own unique set of input parameters and output measures. Generally, the operational simulations inform the system effectiveness within different mission contexts while the first-order engineering models define feasible system configurations.

Construct Baseline Scenario Models: For each model selected, the analytical team must construct operational mission context scenarios that are verified and validated for each aspect of the problem the model represents. These scenarios may be reused from previous studies but generally, the models will require some redesign to meet the needs of the intended research questions. These models will have baseline input parameter settings that will eventually get modified during the high performance computer cluster runs.

Map Design Variables to Model Input Parameters: After selecting the external models that will inform the system's effectiveness and feasibility, the analyst must map design variables derived

from the Conceptual Model (elements from the SysML Diagrams) to the model input parameters; each model will have their own set of elements that map to their own set of inputs.

8.3. Create Experimental Design

The inputs to this activity are the model inputs that will represent each of the factors in the experimental design and the number of simulation replication runs. The mechanisms to create an experimental design are a statistical software package or a custom design builder. The controls are the DOE principles and the output is the experimental design matrix that will be used to perform the experiments.

Select Experimental Design Type: After mapping the design variables to model input parameters, the analyst must now select the type of experimental design. This decision is based on the model run time, the available computational resources, the number and type of design variables (continuous, discrete, or categorical), and the desired complexity to capture in the analysis. For example, a screening design can analyze over 100 model inputs but not clearly capture the higher order complicated behavior in the form of synergies between variables or non-linearities. After identifying the most important system elements that drive the output behavior, we can then perform subsequent experiments on a reduced set of variables to further identify complicated behavior. The field of DOE has several types of experimental designs to choose from, each with their own set of properties and purposes. Space-filling designs are particularly useful for simulation studies where the analyst can expect to find interesting behavior anywhere within the experimental design region. In addition, designs that allow for a mix of continuous, discrete, and categorical factors are important for system simulation studies due to the large number and type of design variables.

Determine the Number of Experiments: An experiment is referred to as a design point within the DOE literature. At a minimum, the number of design points must at least be greater than the number of design variables. Additional design points provide more data within the design region and capture more system configurations. Performing brute force simulation runs on all possible configurations of a high dimensional system problem quickly becomes infeasible due to the run time requirements of a simulation model. A key benefit in doing DOE is to leverage *efficient* experimental designs that sample at the right system configuration settings within the design region in order to get as much information as possible from your experiments.

Generate Experimental Designs Using a Statistical Package or Custom Design Builder: Once the analyst selects the type of design with the required number and type of variables, design points (experiments), and defines the design region, he/she must now either generate a custom design or select an existing cataloged design from the literature. The analyst can generate the design either with a custom design builder or an external statistical package. The end result is a matrix with columns that represent design variables, and rows that are the input parameter specifications for each model experiment. We should expect that there be a unique

experimental design for each external model with unique input parameters and output measures.

8.4. Generate a Study File that Specifies which Model Input Parameters to Change

The controls that govern this activity are the simulation model input formats. In order to perform a simulation experimental design study, we need a mechanism for changing the base case scenario input parameters to the settings specified in the design matrix. The two inputs to this activity are (1) a base case simulation scenario typically created using a GUI that forms the basis for modification by the experimental design matrix and (2) an experimental design, which are a set of model input parameters and the values for each experiment. The mechanism is software that finds, selects, and modifies the model input parameters of the base case scenario to create a set of excursions, one for each experiment or row in the design matrix. The mechanism used to set the values depends on the model input format.

Input formats for simulation models can range from one or more plain-text files, such as CSV (comma separated values) files, to structured text files, like XML, YAML, or JSON, to entries in a database table, or combinations thereof. Each of these formats present different challenges in the task of modifying model input (NATO Science and Technology Organization 2014). The format must have a structured way of defining data and at the same time provide meta-information describing the semantics of the contained data. The XML structure allows for the navigation through the input file programmatically to identify various sections of the inputs easily, e.g., with the help of XPath (Bray et al. 2008). This is greatly advantageous over plain text files, as parsing XML is also, due to the organized structure of such a file, a fairly standard task which does not need to be implemented by hand from the ground up. The software mechanism that will perform this activity should have a graphical user interface that allows the user to map model input parameters to the experimental design columns and save these mappings into a specification study file. The output of this activity is a specification study file that specifies which model input parameters to change.

8.5. Generate Excursion Files for Each Experiment (row in the design matrix).

Once we generate the specification study file, we now need a means to create individual excursion files with the new model input parameters set to the values specified in the design matrix. For each experiment or design point in the design matrix, there should be one excursion file. The mechanism that performs this activity is a software solution that accepts the design matrix and specification study file as an input and produces the set of excursions needed to perform the simulation experiments. If we are using stochastic simulations, we must perform replications of each excursion in order to analyze the output measures.

8.6. Execute HPC Simulation Runs

To execute the simulation excursion runs on an HPC we need a job queuing mechanism that enforces a scheduling policy and priority scheme while monitoring the computer resources that will complete the jobs. A job is one or more runs of the simulation model excursion. The mechanism is a software management system that interfaces with the HPC and excursion files and distributes jobs across available computer resources while managing the transfer of files. The software we used for this research was HTCondor (see <https://research.cs.wisc.edu/htcondor/>). Once an excursion run is complete, the simulation output files are stored in a specified location.

8.7. Post-Process Output Files

There are three necessary requirements to post-process output for operational simulation data that involve extracting, aggregating, and appending data to the experimental design. The controls that govern this activity are the model output data log formats. Figure 8.2 shows visually each of these sub-activities.

Extract Data: A post-processing script is necessary to *extract* the desired measure from one or more data log files. Typically operational simulation model outputs are saved as data logs of events that occurred during the run of the scenario. These files may contain 1000s of records for target acquisitions, state variables changes, or event occurrences. Although there may be common output measures used often for a wide variety of studies, there will always be unique measures that require a script to extract from the data log. For example, one output measure for an operational simulation may be the number of civilians killed by indirect fire and is something that the model does not provide directly to the user. The output data log may record data fields that include the entity status at different time steps. A post-processing script must iterate through the data log to tally the number of civilians that were killed during the model run.

Aggregate Data: The second post-processing requirement is to *aggregate* the individual output data log files into one file. Generally, simulations save data as individual output files for each experiment and each replication.

Append Data to Experimental Design: The final step is to append the output measure data to the experimental design. The final artifact should be a matrix with each experimental design column and output measure columns appended to the right. The matrix should have each experiment or row repeated for each replication.

interesting system configurations with respect to multiple output measures; exploring the functional form of the metamodels helps illuminate the viable system variants. Our purposed methodology is tool agnostic but in order to demonstrate it we must select the tools for each activities listed in Figure 8.1. Table 8.1 shows the tools we intend to use for each of the nine activities. The hyperlinks in the table provide additional information for each of the tools listed.

Table 8.1. Software tools used for each functional activity.

Functional Activity	Tool Link
1.0 Select System Model Element Design Variables	MagicDraw SysML Plugin
2.0 Select Analytical Models, Develop Baseline Scenarios, and Map Design Variables to Model Inputs	Map Aware Non-uniform Automata Simulation
3.0 Create Experimental Design	Custom Space-Filling Design Creator
4.0 Generate a study file that specifies which model input parameters to change	XStudy Data Farming Tool
5.0 Generate excursion files for each experiment (row in the design matrix).	OldMcData Data Farming Tool
6.0 Execute HPC Simulation Runs	HTCondor
7.0 Post-process Output Files	Post-Processing Script written in R
8.0 Perform Statistical Metamodeling	JMP
9.0 ERS Tradespace Visualization	JMP

9. Simulation Analysis and Tradespace Visualization

Despite our efficient selection of experimental design variable setting there is still a high volume of multi-dimensional data that requires sophisticated analytical techniques to gain insights. Some of these techniques include descriptive statistics, visualizing distributions to understand the spread of the output measures and identify outliers, visualizing scatter and contour plots to understand the output measure landscape, and many more. To demonstrate how we can apply these statistical techniques we will use a small use case example that explores a soldier enhancement system that includes a radio, sensor and a rifle. Our demonstration will include the following three measures of effectiveness (MOEs): *ATK_Aware*, *ATK_Lethal*, and *ATK_Protect*; see Section 5.4 for the description of these MOE responses. We will first discuss the exploratory analysis techniques and then describe the dashboard used to identify key tradable variables and viable system variants. In this section, we refer to the term model input as a factor and to a model output as a response.

9.1. Exploratory Analysis

Before fitting meta-models, we begin by examining the distributions of our three MOEs of interest and the pairwise correlations between them. Figure 9.1 contains the set of histograms and statistical summaries of the MOEs, produced with the JMP® statistical software (www.jmp.com). The data set upon which these are based represents the mean of each of the 500 design points, hence N=500 for each of

these. From these histograms, we note a couple of things: (1) there is reasonable variation across the design point, meaning that at least one factor had a significant impact on each MOE, and (2) some MOEs are closer to being normally distributed than others. For example, *ATK_Lethal* appears bi-modal, a fact we'll seek to explain with the metamodeling approaches.

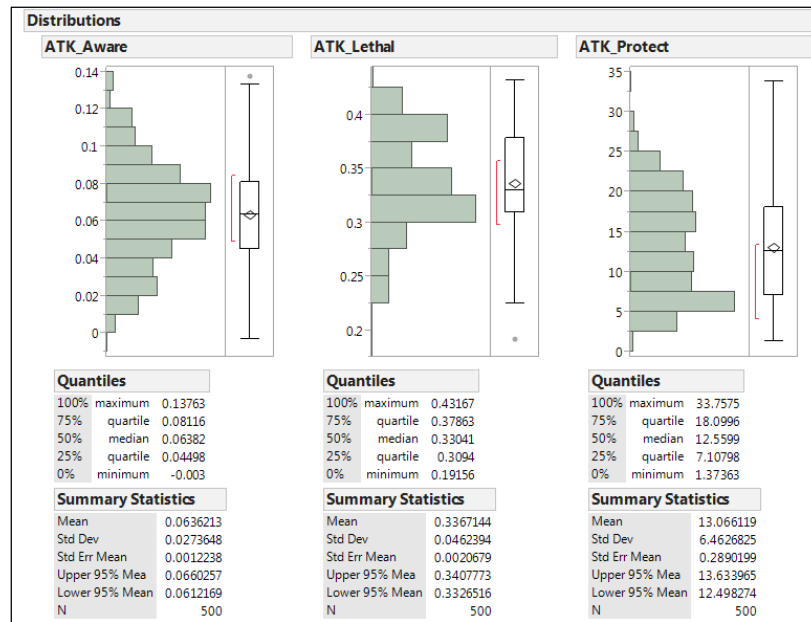


Figure 9.1: Histograms and Summary Statistics for the 3 MOEs

Figure 9.2 contains the set of scatterplots and pairwise correlations for our MOEs. *ATK_Aware* and *ATK_Lethal* have a moderate positive correlation (approximately .37), which makes sense, since one would expect that an increase in awareness of Red would lead to an increase in being able to kill more Red, earlier. *ATK_Aware* and *ATK_Protect* have a moderate negative correlation (approximately -.35), which also makes sense, in that an increase of awareness of Red should lead to an improved ability for Blue to protect itself, thus leading to a lower total number of hits taken. The strongest correlation, between *ATK_Lethal* and *ATK_Protect* (approximately -.83), confirms intuition that killing more red, earlier, leads to a dramatic decrease in the total number of Blue hits taken by Red. The fact that the two ‘moderate’ correlations are not stronger simply means that there is more to the story, which we seek to uncover with the metamodels. Another insight we can gain from examining correlations is whether or not MOEs “trade off” with each other, or, whether we can simultaneously improve all MOEs together. The latter is the case for these 3 MOEs, they do not trade off with each other.

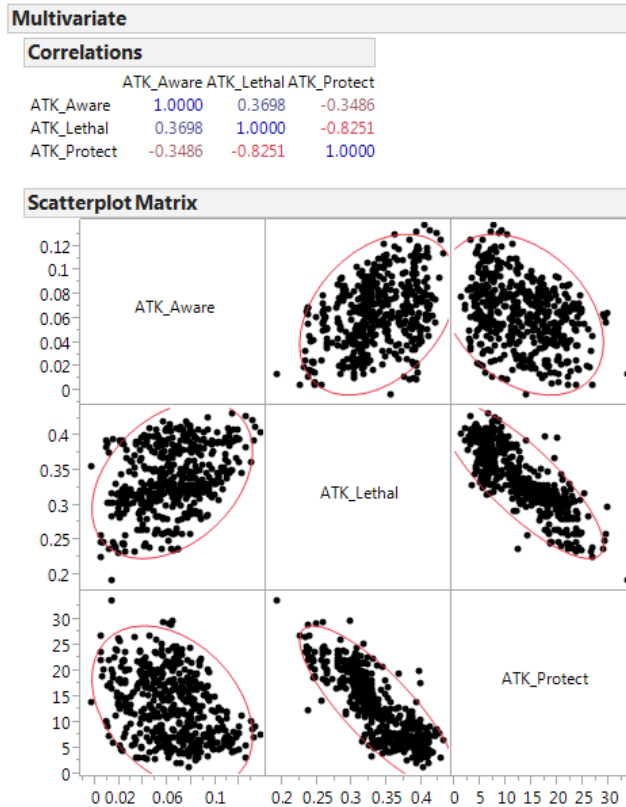


Figure 9.2: Scatterplots and Correlations

ATK_Aware Metamodels: Figure 9.3a depicts the sorted parameter estimates for the *ATK_Aware* regression model (with R^2 of .92). This particular model contains only main effects, no interaction or quadratic effects. Included in the list of most significant main effects are the design parameters for number of UAV (*NoUAV*), soldier classification range (*SDRClassRng*), number of robots (*NoRobots*), and internal communication delay between squad members (*InCommDelay*). An effective way to show how each design parameter impacts an MOE is with a prediction profiler, shown in Figure 9.3b. A prediction profiler displays the partial derivatives for each design parameter in a metamodel. These profilers show how changes in each design parameter or design driver impact the MOE, while the other design parameters are held constant. From both a. and b. we can see that *InCommDelay* has a negative effect on overall awareness, while the other three design parameters have a positive effect. Figure 9.3c contains a partition tree with splits in the experimental data that show thresholds for the 3 of the 4 design parameters that appeared in the regression model.

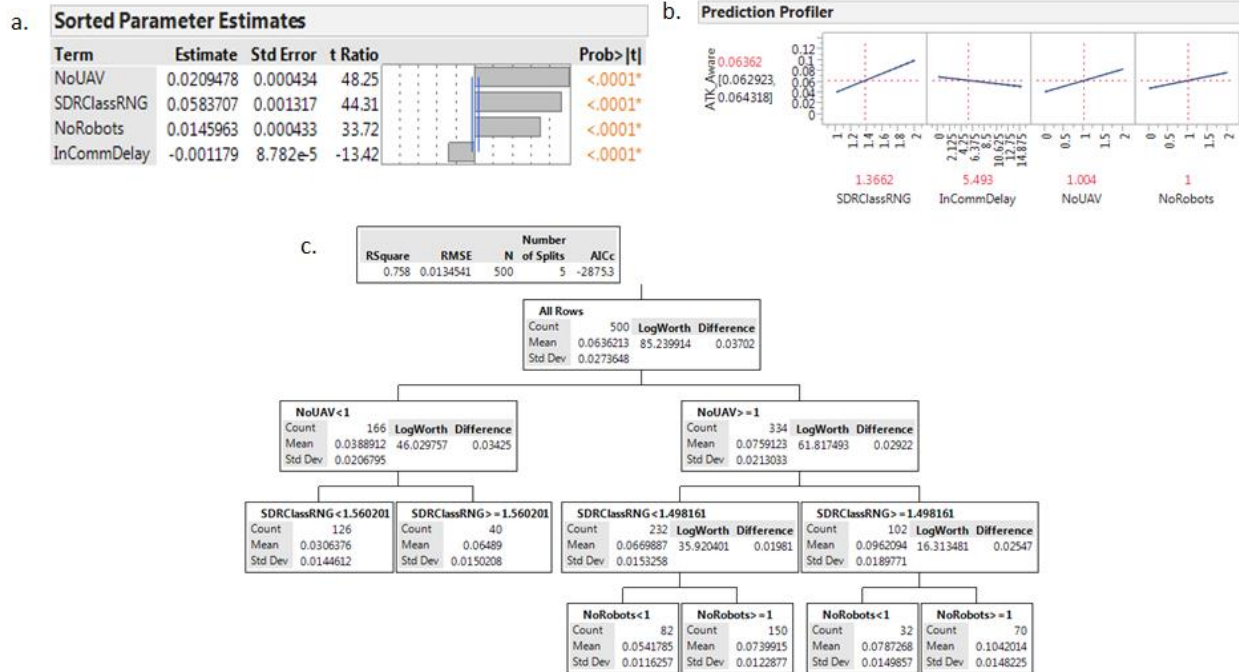


Figure 9.3: (a) Sorted Parameter Estimates. (b) Prediction Profiler. (c) Partition Tree.

A summary of the insights captured from the polynomial metamodel coefficients are summarized in Table 9.1. These insights are based on the assumptions behind the agent-based simulation model, the scenario, and the established ranges of the design parameters. We must remember that when a design parameter is insignificant according to the metamodel it does not mean it is not important within a different scenario or outside these established ranges. All models are abstractions of reality and must be verified and validated. Like the simulations themselves, the metamodels are also models that must be validated in order to provide valuable insights that help understand a complex system design problem.

Table 9.1. DOE insights from the ATK_Aware MOE from the attack scenario.

Type of Insight	Design Parameter	Coefficient Sign	Description
Design Drivers	NoUAV	Positive	Increasing the number of UAVs improves awareness.
	SDRClassRNG	Positive	Increasing the soldier classification range improves awareness.
	NoRobots	Positive	Increasing the number of robots improves awareness.
	InCommDelay	Negative	Increasing the delay between internal squad communications degrades awareness.
Synergies / Interactions	N/A	N/A	
Diminishing rate of change	N/A	N/A	
Thresholds	NoUAV	N/A	Solider system shall incorporate at least one Unmanned Aerial System.
	SDRClassRNG	N/A	Solider classification range multiplier shall be at least 1.5.
	NoRobots	N/A	Solider system shall incorporate at least one Robot.

ATK_Lethal Metamodels: Figure 9.4a depicts the sorted parameter estimates for the *ATK_Lethal* regression model (with R^2 of .88). This model contains main effects, one interaction effect, and one quadratic effect. Included in the list of most significant main effects are the design parameters for soldier M4 weapon range (*M4RNG*), soldier classification range (*SDRClassRng*), number of UAV (*NoUAV*), and soldier detection range (*SDRDetRNG*). Figure 9.4b contains the prediction profiler, indicating that *M4RNG* has the strongest impact on this MOE. Figure 9.4c is an interaction matrix where each plot shows the synergies between two design parameters, one in a row and the other in a column. The design parameters in the columns show its effect on the MOE when the design parameter in the row is set at the low and high levels. The line segments represent the low and high settings of the row design parameter; when the slopes of these lines are different, this means that the effect of the column design parameter depends on the setting of the row design parameter (23). Figure 4d contains the partition tree, with R^2 of .83.

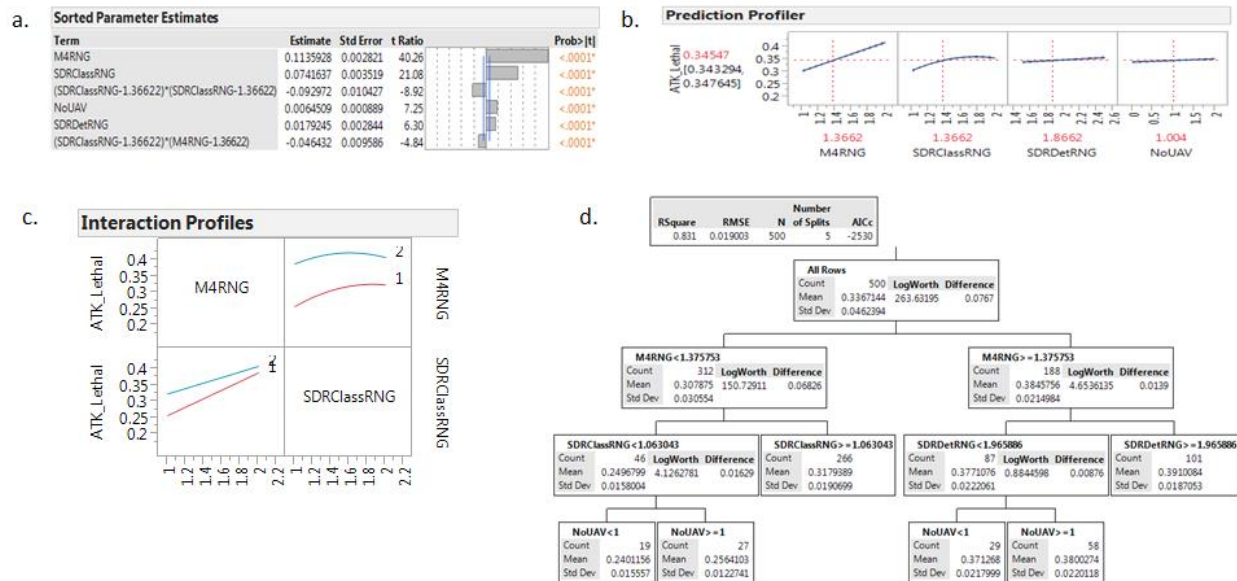


Figure 9.4: (a) Sorted Parameter Estimates. (b) Prediction Profiler. (c) Interaction Profiler. (d) Partition Tree.

A summary of the insights captured from the polynomial metamodel coefficients are summarized in Table x. As before, these insights are based on the assumptions behind the agent-based simulation model, the scenario, and the established ranges of the design parameters.

Table 9.2. DOE insights from the ATK_Lethal MOE, using the attack scenario.

Type of Insight	Design Parameter	Coefficient Sign	Description
Design Drivers	M4RNG	Positive	Increasing the soldier's M4 weapons range improves lethality.
	SDRClassRng	Positive	Increasing the soldier's classification range improves lethality.
	SDRDetRng	Positive	Increasing the soldier's detection range improves lethality.
	NoUAV	Positive	Increasing the number of UASs improves lethality.
Synergies / Interactions	SDRClassRng / M4Rng	Negative	The soldier's classification range interacts with his M4 weapon range. Increasing the M4 weapon range has more of an impact when the classification range is low. (This is a mild to moderate interaction effect.)
Diminishing rate of change	SDRClassRng	Negative	Analysis indicates that there is a point of diminishing returns beyond a class range multiplier of 1.5.
Thresholds	M4RNG	N/A	Soldier M4Rng multiplier shall be at least 1.4.
	SDRClassRng	N/A	Solider classification range multiplier shall be at least 1.06.
	SDRDetRng	N/A	Solider detection range multiplier shall be at least 2.
	NoUAV	N/A	Solider system shall incorporate at least one Unmanned Aerial System.

ATK_Protect Metamodels: Figure 9.5a depicts the sorted parameter estimates for the *ATK_Protect* regression model (with R^2 of .92). This model contains main effects, interaction effects, and one quadratic effect. Included in the list of most significant main effects are the design parameters for soldier M4 weapon range (M4RNG), soldier classification range (*SDRClassRng*), soldier detection range (*SDRDetRNG*), internal squad communications delay (*InCommDelay*), and soldier field of view (*SDRFOV*). Figure 9.5b contains the prediction profiler and Figure 9.5c is the interaction matrix. A strong interaction between *SDRClassRng* and *InCommDelay* indicates that increasing *InCommDelay* over these ranges has no effect when classification range is high, but does significantly increase the number of blue hits taken when classification range is low. This indicates that increased classification range can mitigate the negative effect of increased internal squad communications delay. Figure 9.5d contains the partition tree, with R^2 of .68.

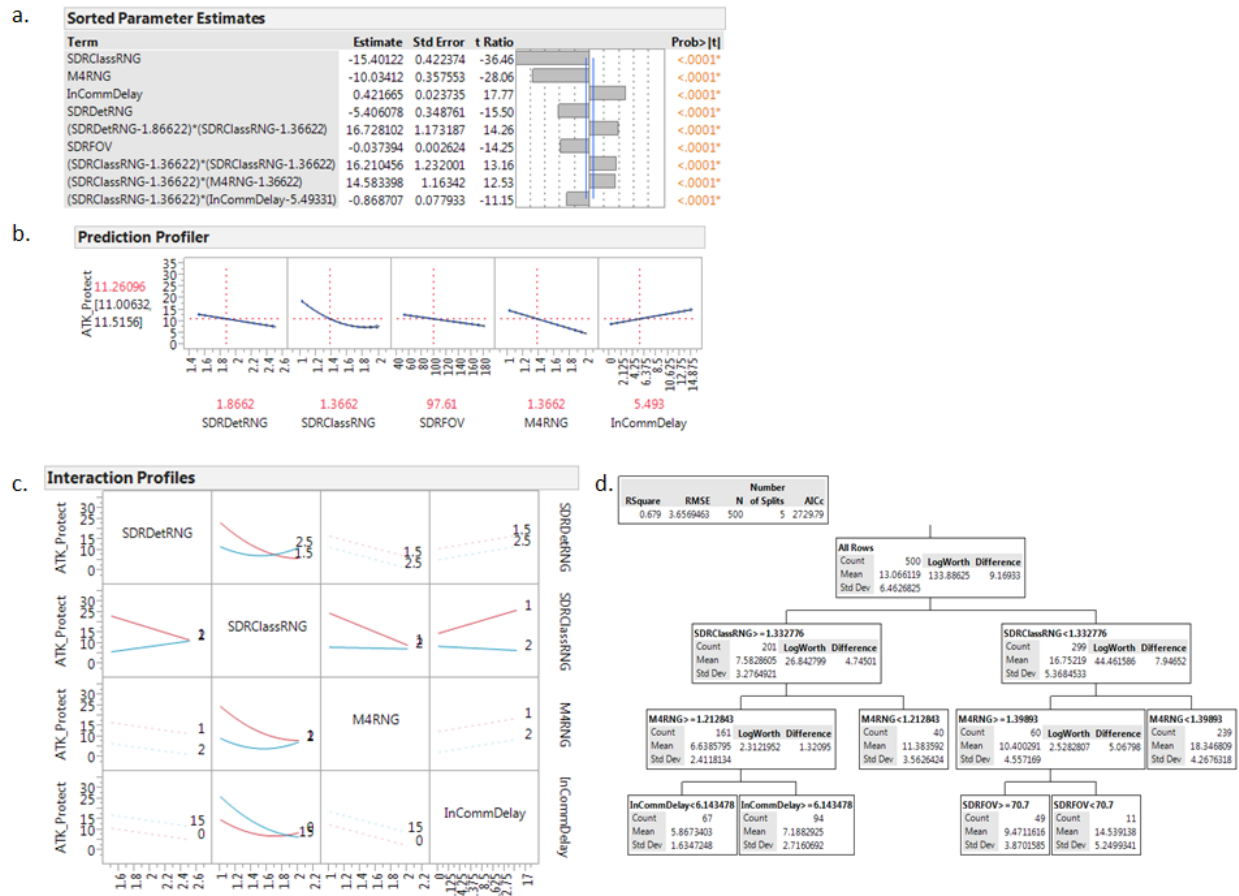


Figure 9.4: (a) Sorted Parameter Estimates. (b) Prediction Profiler. (c) Interaction Profiler. (d) Partition Tree.

A summary of the insights captured from the polynomial metamodel coefficients are summarized in Table 9.3. As before, these insights are based on the assumptions behind the agent-based simulation model, the scenario, and the established ranges of the design parameters.

Table 9.3. DOE insights from the ATK_Protect MOE, using the attack scenario.

Type of Insight	Design Parameter	Coefficient Sign	Description
Design Drivers	M4RNG	Negative	Increasing the soldier's M4 weapons range improves the ability of the squad to protect itself.
	SDRClassRng	Negative	Increasing the soldier's classification range improves the ability of the squad to protect itself.
	SDRDetRng	Negative	Increasing the soldier's detection range improves the ability of the squad to protect itself.
	InCommDelay	Positive	Increasing the delay between internal squad communications degrades the ability of the squad to protect itself.
	SDRFOV	Negative	Increasing the soldier's field of view improves the ability of the squad to protect itself.
Synergies / Interactions	SDRClassRng / InCommDelay	Negative	The soldier's classification range interacts with the squad's delay in internal communications. Increasing InCommDelay has no effect when classification range is high, but does significantly increase the number of blue hits taken when classification is low.
Diminishing rate of change	SDRClassRng	Negative	Analysis indicates that there is a point of diminishing returns beyond a class range multiplier of 1.5.
Thresholds	SDRClassRng	N/A	Solider classification range multiplier shall be at least 1.3.
	M4RNG	N/A	Solider M4 weapon range multiplier shall be at least 1.2.
	SDRFOV	N/A	Solider field of view shall be at least 70 degrees.
	InCommDelay	N/A	Internal squad communications delay shall be less than 6 seconds.

After acquiring the insights derived from our DOE analysis we can now capture them within our MBSE system integrated model. Figure 9.6 shows a number of SysML relationship elements that convey the insights captured. The figure shows two requirement containment structures for the emergent and local properties. Analyzing simulation model output results are one way a systems engineer can validate a system requirement. The system drivers identified above can reveal which value properties or structural blocks satisfy the requirements classified as emergent properties. Figure 9.6 shows these satisfy relationships between value properties and emergent property requirements. The threshold and interaction insights derived new requirements that provide more specific requirement statements. Below the emergent properties in Figure 9.6 are derived requirement relationships for the local properties of the system. Additionally, the rationale comments highlight the interactions that exist between these local property requirements.

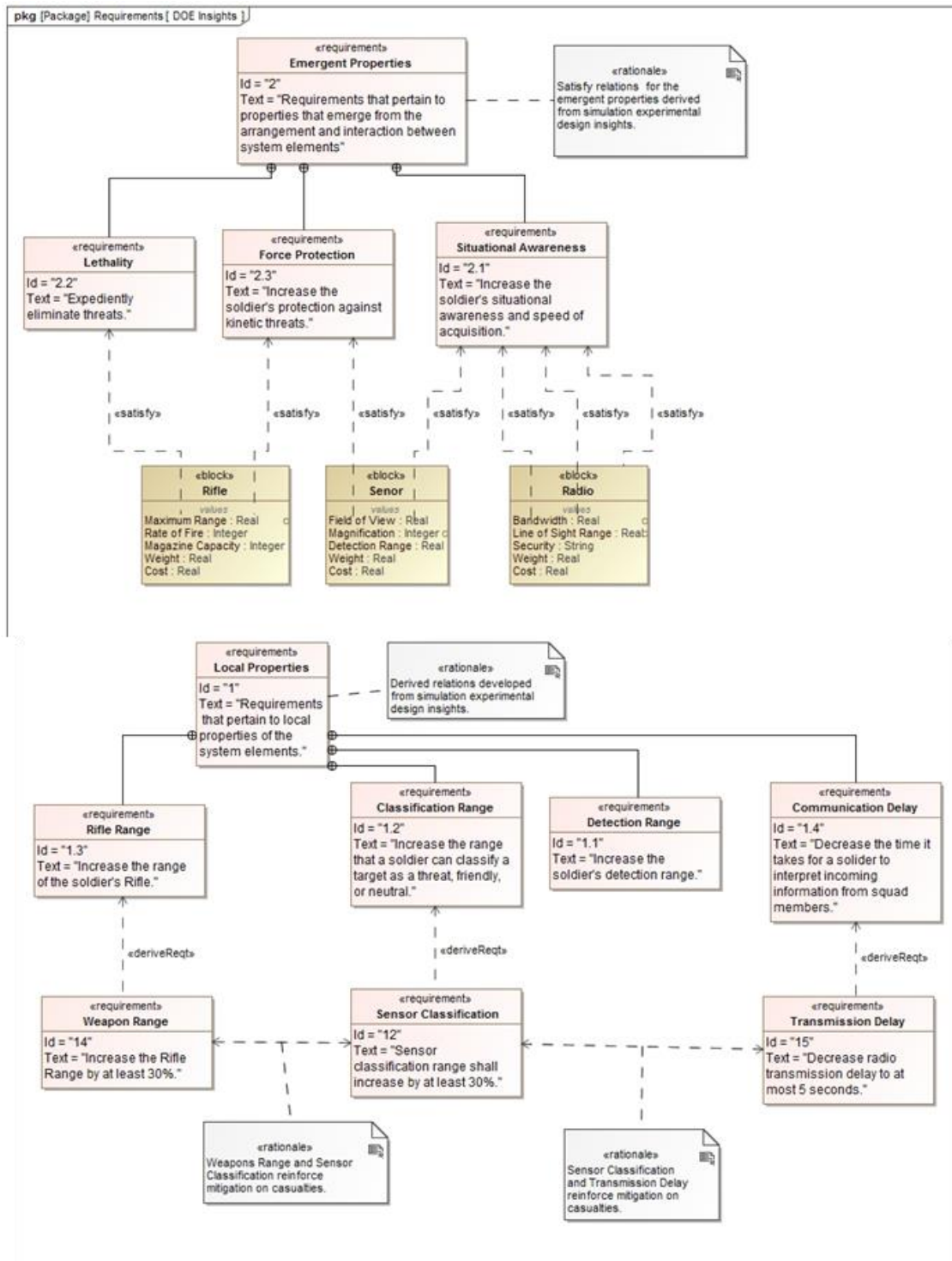


Figure 9.6. DOE insights captured within the MBSE system integrated model.

9.2. Dashboard Tradespace Visualization

We can easily visualize the response surface landscape with a surface plot when there are only three dimensions. Because there are often several more dimensions in a systems design problem we must visualize cross sections of the response surface with respect to only two model inputs at a time. A contour profiler is a two dimensional projection that is a horizontal cross section of a response surface within the experimental design region. We define this region when we scale and translate the design matrix to the desired high and low settings for each model input prior to performing the simulation experiments. Visualizing the selected projections allows the user to interactively explore how multiple responses depend on two selected model inputs. The contour profiler allows us to set limits on the responses to help define infeasible and feasible regions in the response surface; the shape of these shaded regions is dependent on the functional form of the multi-dimensional metamodel. For analytical and technical details on the profilers used in this section, see (SAS Institute 2015).

Figure 9.7 illustrates how a contour profiler is a horizontal cross section of a notional response surface. The crosshairs within the contour profilers indicate the model input settings depicted along each axis. If the user changes the setting of a model input other than the ones shown in each axes, the shape of current projection will change; this is because the change reflects a movement to a different area of the response surface. The differences in Figure 9.7a and 9.7b are a result of changing the model input not shown in the contour profiler. We also note that the response surface with respect to X_1 and X_2 is much different than X_1 and X_3 .

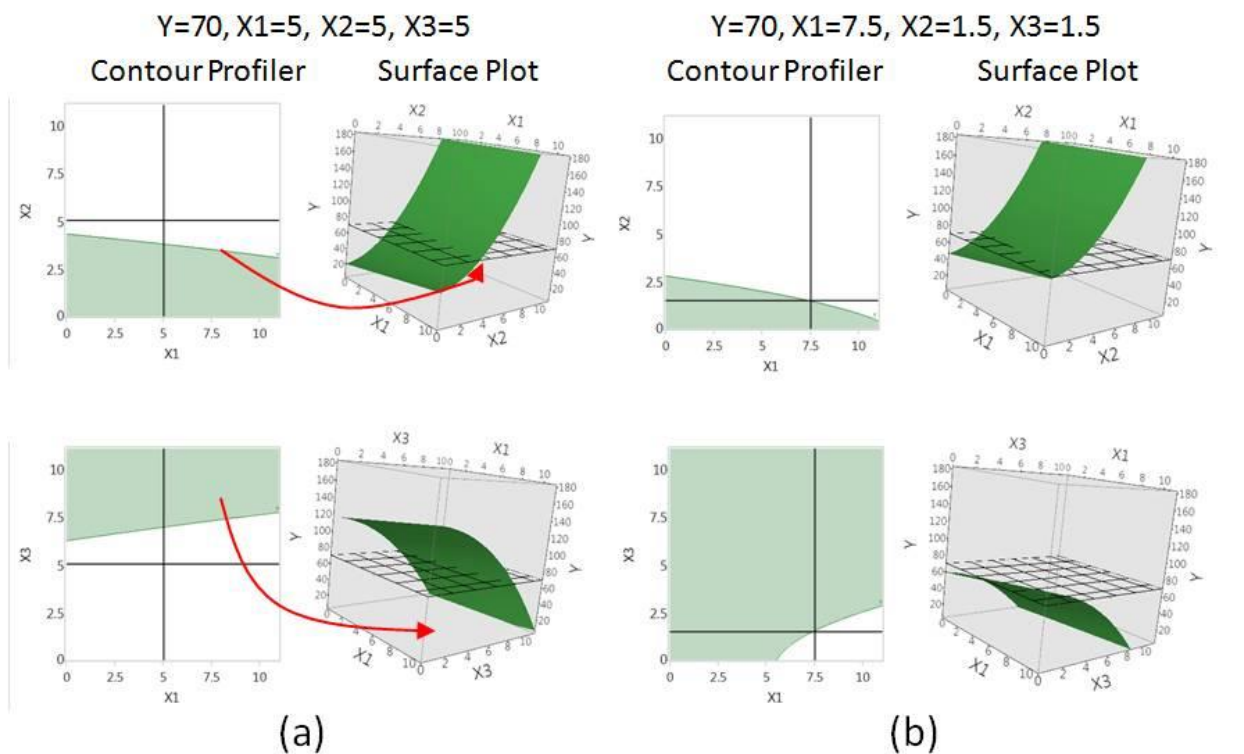


Figure 9.7. Contour profilers showing horizontal cross sections of the response surface.

The red arrows in Figure 9.7a that point from the contour profilers to the surface plots show that the contour line is where the horizontal grid plane intersects the response surface while the shaded region indicates everything underneath the horizontal grid plane.

In addition to the contour profilers that show horizontal cross sections we can also visualize the response surface landscape using prediction profilers that show vertical cross sections. Figure 9.8 shows the vertical cross section for the same notional response surface from Figure 9.7 for each of the model inputs, X_1 , X_2 , and X_3 . The prediction profilers allow us to understand which model inputs are most significant and how they affect the responses.

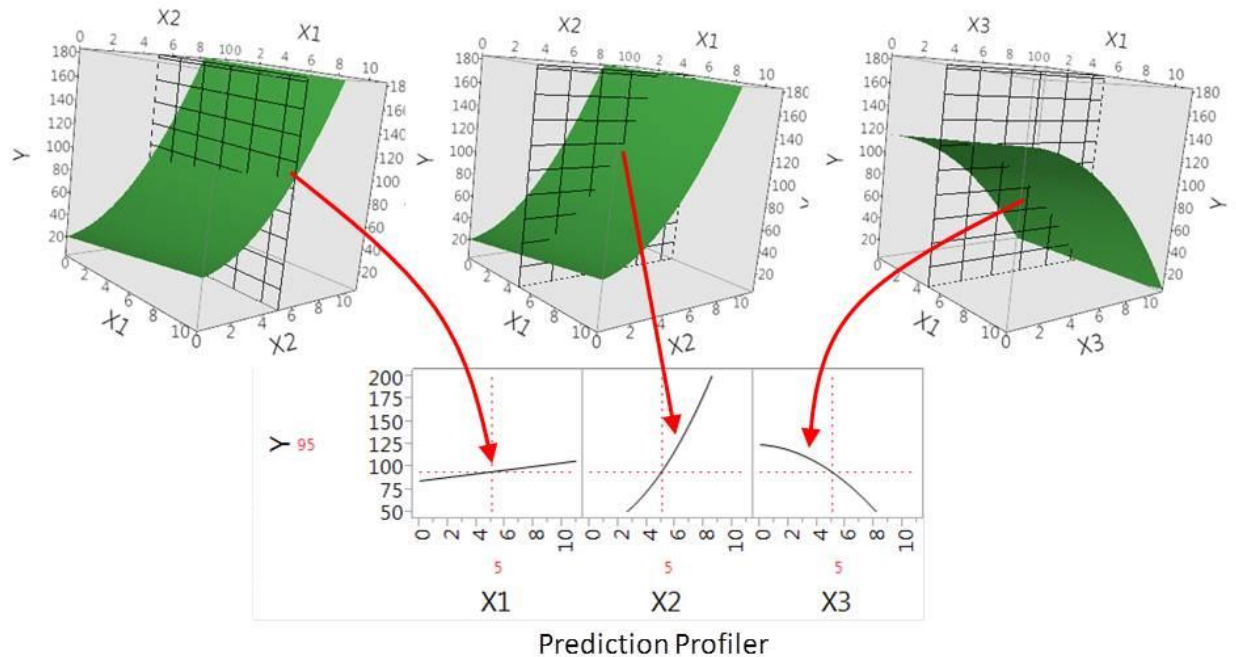


Figure 9.8. Prediction profilers showing vertical cross-sections of the response surface.

Each graph within the prediction profiler shows how changing the model input from the low to high setting affects the response on the y-axis when all other model inputs are held constant. The shapes of these effects are a function of the metamodel fit using linear regression.

We created a dashboard that incorporates the horizontal and vertical cross-section profilers for multiple responses in order to accomplish two objectives. First, we want to easily identify where the tradeable variables are and second, we need an efficient way to identify a reduced set of viable system variants that will span the solution space. The dashboard consists of three components, the contour profiler, prediction profiler and the Monte Carlo filtering components. Our next sections will describe each of these components and how they are used to identify tradable variables and viable system variants.

9.2.1. Prediction Profiler Dashboard Component

The prediction profiler component shows a matrix where each column represents a model input and each row a model output (response). Each cell in the matrix shows the vertical cross section of the

row's response. The horizontal cross sections reveal the impact of the model input on each of the responses. There are two key features that the Prediction profiler provides; first, it identifies tradable variables with the color profiler algorithm and second, it optimizes solutions to find model input settings that perform well across multiple responses. We will now describe each of these features in more detail.

Color Profiler Algorithm. The matrix cells in the Prediction profiler are colored such that green indicates a positive impact to the response, red indicates a negative impact, white indicates a response with a target value, and black indicates no impact. Additionally, there is a color gradient applied so the cells with a higher impact are darker and the cells with a lower impact are lighter. The algorithm colors these gradients based on the magnitude of the response change between the low and high settings of the model input. The dashboard allows the user to specify whether we want to maximize, minimize, or achieve a specified target of the response. The coloring algorithm uses these specifications to color the cells appropriately. Figure 9.9 shows a screenshot of the prediction profiler dashboard component.

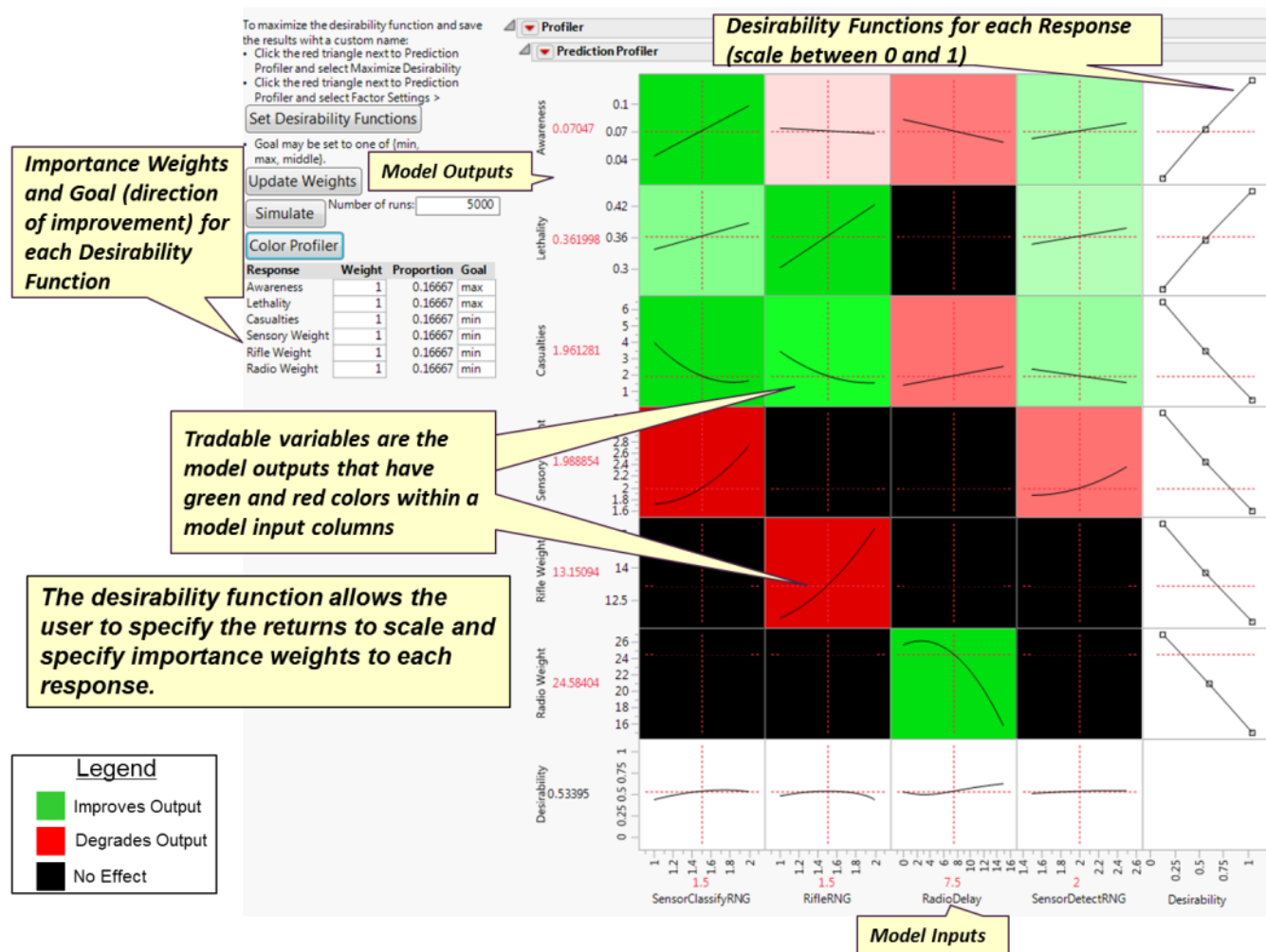


Figure 9.9. Prediction profiler dashboard component showing vertical cross sections.

Within each column in the matrix, the responses that have opposite green and red colors show where the key tradable variables are with respect to each model input. For example, in Figure 9.9, the column that represents *RifleRNG* (a model input representing the range of the rifle) has a red cell for *Rifle Weight* and green cells for the *Casualties* and *Lethality* responses. We can interpret these cells to mean that as we increase the rifle range, we improve our ability to reduce casualties and provide lethal fire but we degrade our goal of reducing the rifle weight. Increasing the rifle range has a positive impact on our operational measures (*Casualties* and *Lethality*) but has a negative impact on a physical consideration (weight). Therefore, we must tradeoff weight to achieve a higher operational effectiveness. The model input columns are ordered such that the ones that have the highest impact across all responses are ordered from left to right. In this way, we can easily identify the highest impacting model inputs across all responses.

Multiple Response Optimization. The prediction profiler component has an optimization feature that allows the user to specify a weighted desirability function for each response and find a balanced solution. The desirability function translates the response scale to a value between 0 and 1; 0 indicates the least desirable value and 1 indicates the most desirable value. The user specifies a response goal to maximize, minimize, or achieve a specified target value for each response. Figure 9.10 shows the three types of desirability functions that translate response values along the vertical axis to desirability values along the horizontal axis.

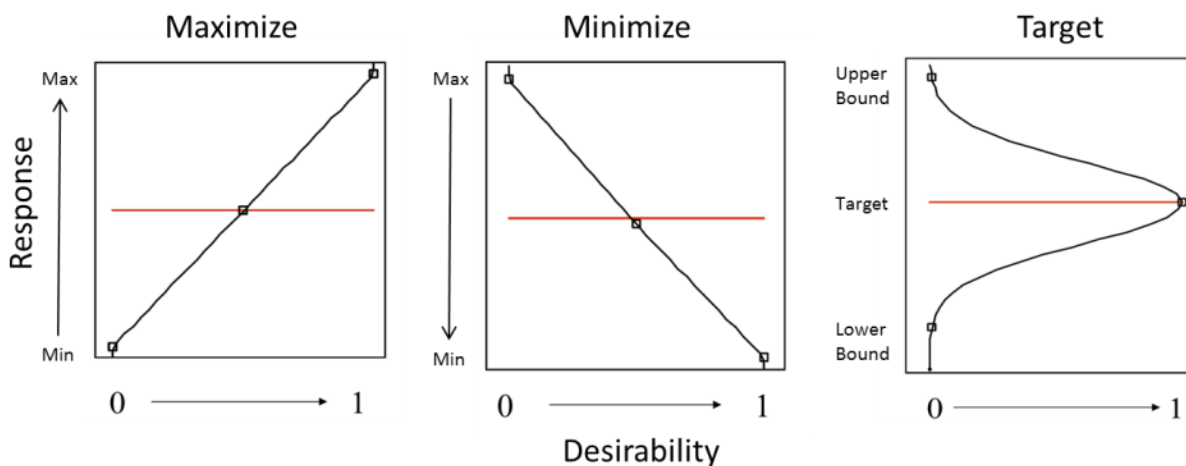


Figure 9.10. Desirability function types. Red line indicates the response outcome for the current model input settings.

The purpose of desirability transformation is to allow the user to specify the returns to scale along the range of response outcomes and to establish a common scale across all responses. The common scale allows us to aggregate all responses using a weighted total desirability function. The total desirability function is based on the average of the natural logs of the desirabilities, otherwise known as the geometric mean, and has the following form:

$$D^* = \frac{1}{k} [w_1 \ln(d_1) + w_2 \ln(d_2) + \dots + w_k \ln(d_k)], \quad (4)$$

where k is the number of responses, d_k is the desirability function for response k , w_k is the importance weight for response k , and D^* is the optimal total desirability. Equation 4 is the objective function for the optimization algorithm that finds desirable solutions across multiple responses.

9.2.2. Contour Profiler Dashboard Component

The contour profiler dashboard component shows a collection of contour profilers categorized within different domains. Each contour profiler has one or more responses that pertain to each domain category. The profilers show a two-dimensional projection of the response. The two dimensions are displayed along the vertical and horizontal axis and represent two model inputs. The cross hair inside the profiler indicates the settings for each of the model inputs. A slider bar above each contour profiler allows the user to set low and high limits on the response. These limits represent desired effectiveness and constraints that shade the profilers to indicate infeasible areas of the design space. If the cross hairs fall within a response's shaded region, then the current model input settings will not satisfy the desired effectiveness or constraint for that response. Figure 9.11 shows a screenshot of the contour profiler dashboard component. Also shown in Figure 9.11 is a floating control panel that allows the user to set the two model inputs that are shown in each of the contour profilers. By selecting different model inputs, we can see different parts of the multi-dimensional response surfaces.

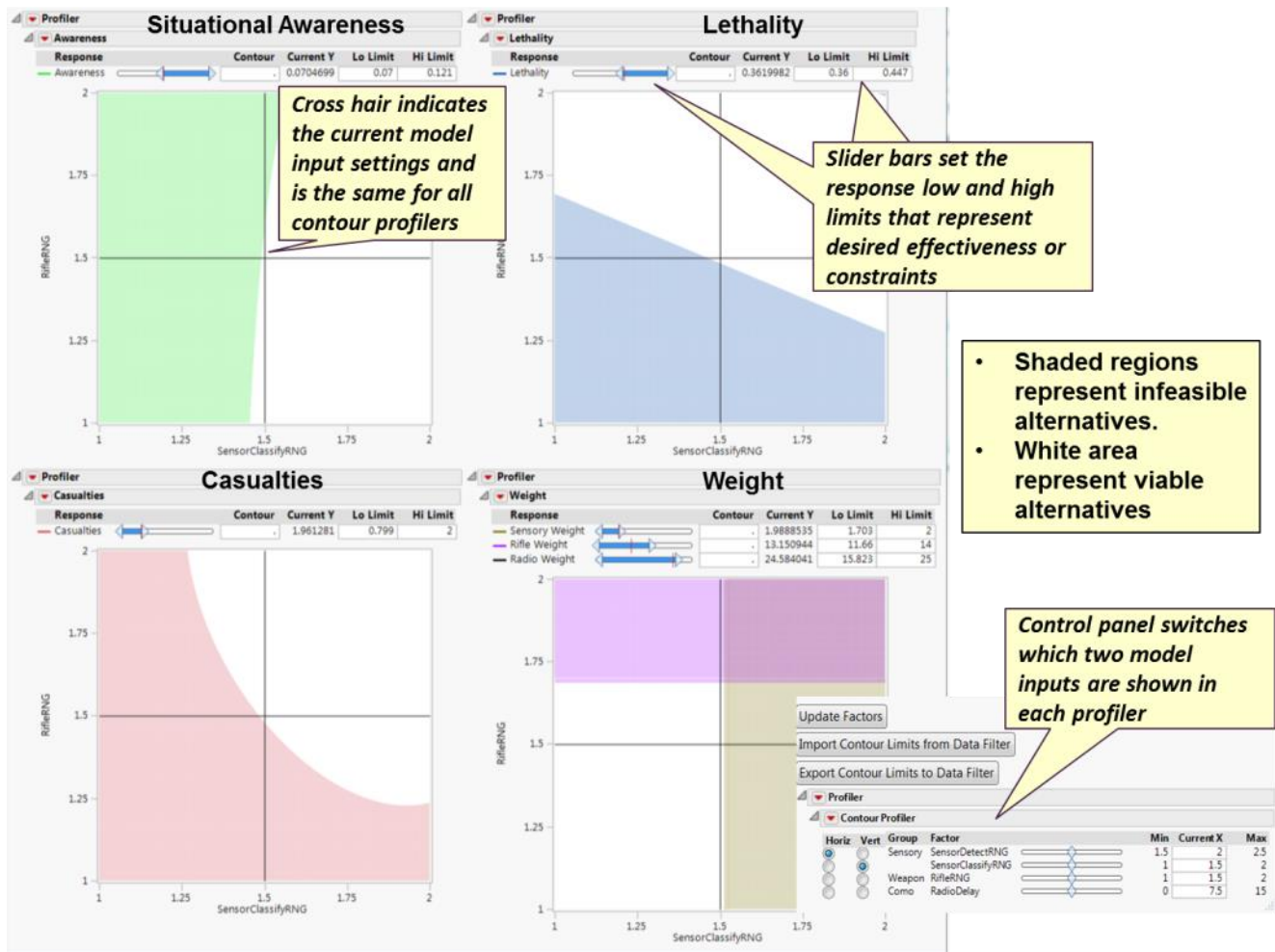


Figure 9.11. Contour profiler dashboard component.

9.2.3. Monte Carlo Filtering Component

The purpose of the Monte Carlo Filtering component is to find a collection of feasible alternatives that satisfy all response limits specified in the contour profiler component. A powerful benefit of the response metamodels is that they act as surrogates to the simulation. Rather than having to run a lengthy simulation to obtain the results of a new system configuration, we can leverage the metamodels to obtain a result in a matter of seconds instead of hours, days or weeks. These approximations can save a tremendous amount of time, especially for a system design problem with several complex models with lengthy run times. The Monte Carlo Filtering dashboard component allows the user to generate thousands of system design configurations using a Monte Carlo simulation. Each simulation run creates a unique system design alternative by drawing a uniform random variate between the low and high settings of each model input. Within the Prediction profiler dashboard component, the user enters the number of simulations and presses the “Simulate” button. A scatter plot matrix appears with a floating response data filter. Each dot in the scatter plot matrix represents a single system alternative. The user can apply the data filters to the responses in order to eliminate alternatives (dots) from the scatter plot. In addition, the user can import and export the response limits set in the contour profiler. After exporting the response limits from the contour profiler to the scatter plot, the alternatives that remain

become the feasible set that resides within the white region of the contour profilers. The more Monte Carlo simulations the user specifies, the more solutions there will be within the white region. The end result is a reduced set of viable system variants that satisfy all desired effectiveness constraints. Figure 9.12 shows a screen shot of the Monte Carlo Filtering component along with the prediction profiler and floating panels.

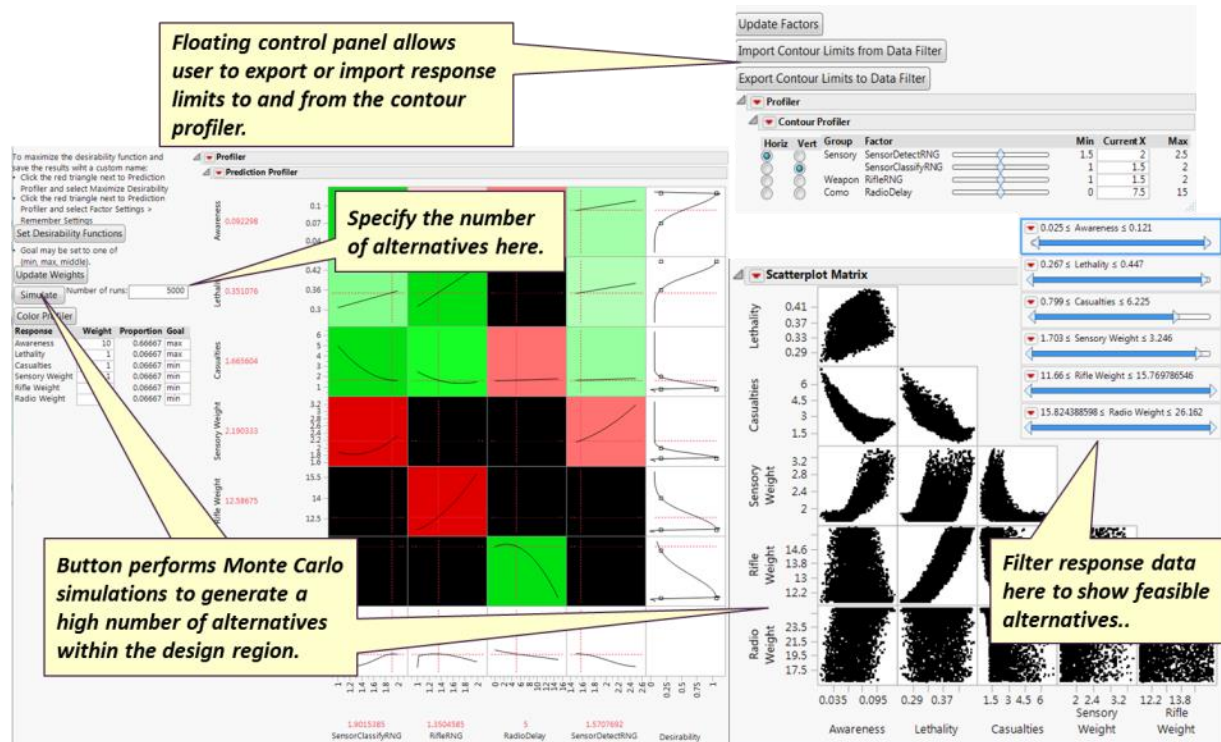


Figure 9.12. Monte Carlo Filtering Component.

9.2.4. Viable Variant Exploration

We will now walk through the steps to find viable system variants that meet our desired effectiveness and are feasible.

Step 1: Set constraints and minimum acceptable response levels. Initially, our contour profilers have no limits set. When using the dashboard to find viable variants, we first set the constraints of the responses that we cannot violate and the minimum acceptable response settings for our desired effectiveness. We then return to the Prediction profiler component and press the "Set Desirability Functions" button. This button changes the shape of the desirability function so that we only obtain value within the limits that were set in the Contour profiler component. For example, in Figure 9.11, the high limit for the *Casualties* response in the contour profiler is set to 2. In Figure 9.13, the high setting for the *Casualties* desirability function in the Prediction profiler is also set to 2. Redefining the desirability functions to match the limits set in the Contour profiler allows us to optimize the weighted desirability function to achieve a solution as close to the limits as possible.

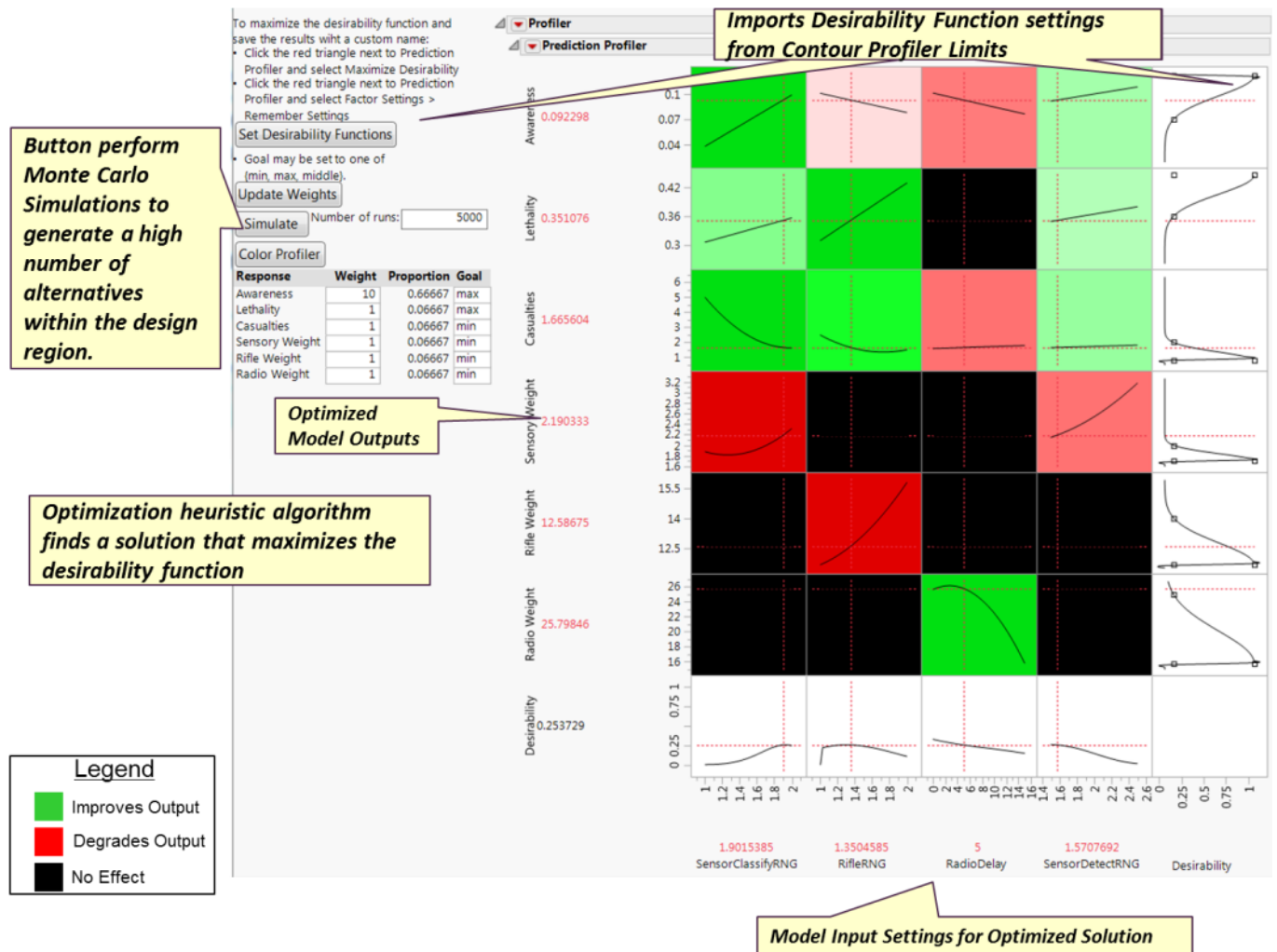


Figure 9.13. Prediction profiler optimization algorithm.

Step 2: Set response importance weights and run the optimization algorithm. To the left of the Prediction profiler component in Figure 9.13, there is an area to specify the importance weights for each response. A system design problem has several stakeholder views and scenarios that prioritize the responses differently. The dashboard allows the user to explore system design solutions with different response prioritization schemes to account for different stakeholder views and scenarios. To run the optimization algorithm, the user clicks the red triangle at the top next to the Prediction Profiler title and selects “Maximize Desirability.” After the optimization algorithm completes, the dashboard displays the model input settings of the solution.

Step 3: Explore changes to model input settings to find feasible solutions. Because the contour profilers are linked to the prediction profilers, we can examine the contour profilers to identify where the optimized solution does not meet the desired effectiveness or feasibility constraints; this occurs where the contour profiler crosshairs are in a shaded region. Figure 9.14 shows the contour profiler dashboard component with the optimized solution from Figure 9.13. The cross hairs are positioned over

two shaded regions indicating that the solution does not satisfy the *Radio Weight* and *Sensor Weight* responses.

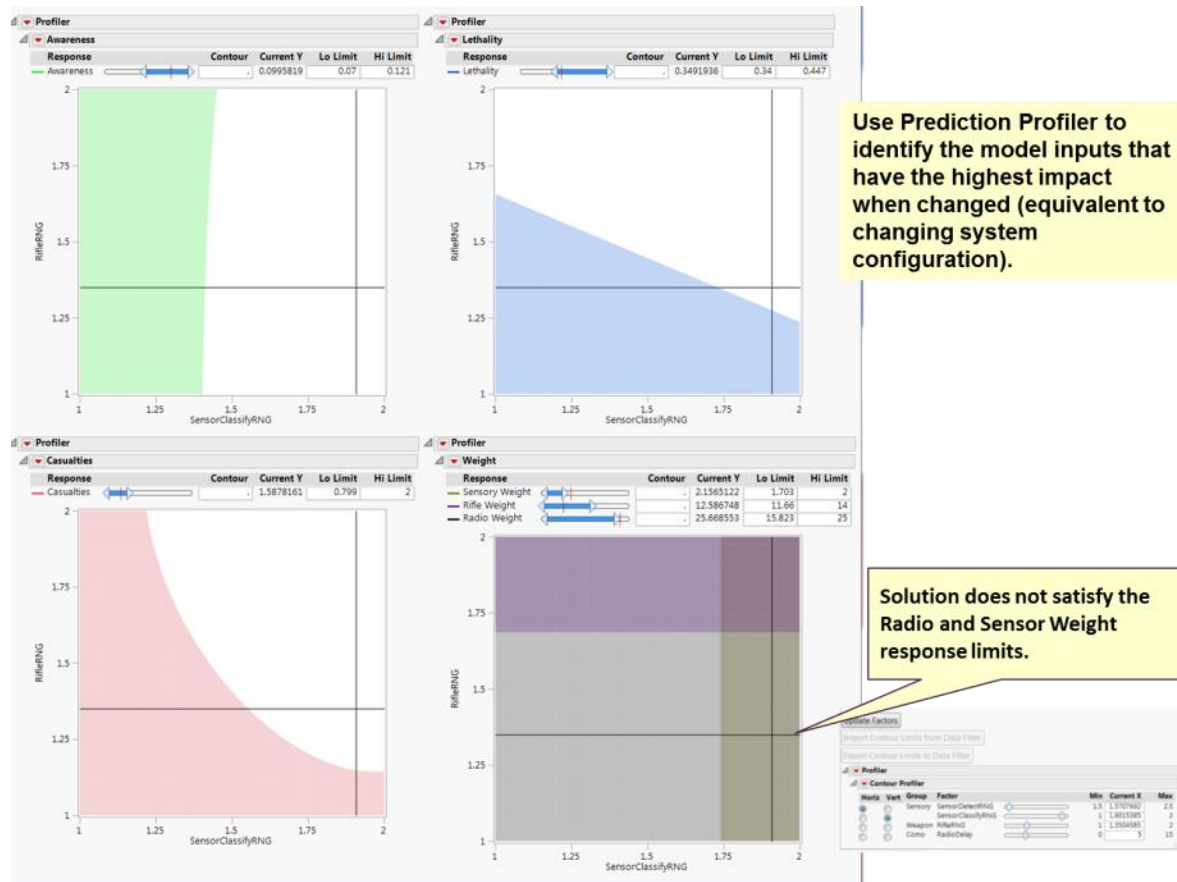


Figure 9.14. Contour profiler dashboard component showing an infeasible solution.

The prediction profiler identifies the model inputs that have the highest impact on the responses; therefore, we can use it to decide which model input to change. Model inputs represent value properties of a system block. As a result, we must consider the practical design implications of changing the value properties that coincide with a model input change. We may find that the highest impacting model input represents a value property that is too costly or infeasible to change.

The Prediction profiler in Figure 9.13 indicates that *RadioDelay* is the model input that has the highest impact on *Radio Weight*. The floating window shown in Figure 9.14 allows us to change the two-dimensions shown in the contour profilers. Changing the vertical axis to *RadioDelay* and moving the slider bar in the floating window allows us to find a feasible solution that meets the *Radio Weight* limit. Figure 9.15 shows the contour profiler after changing the *RadioDelay*.

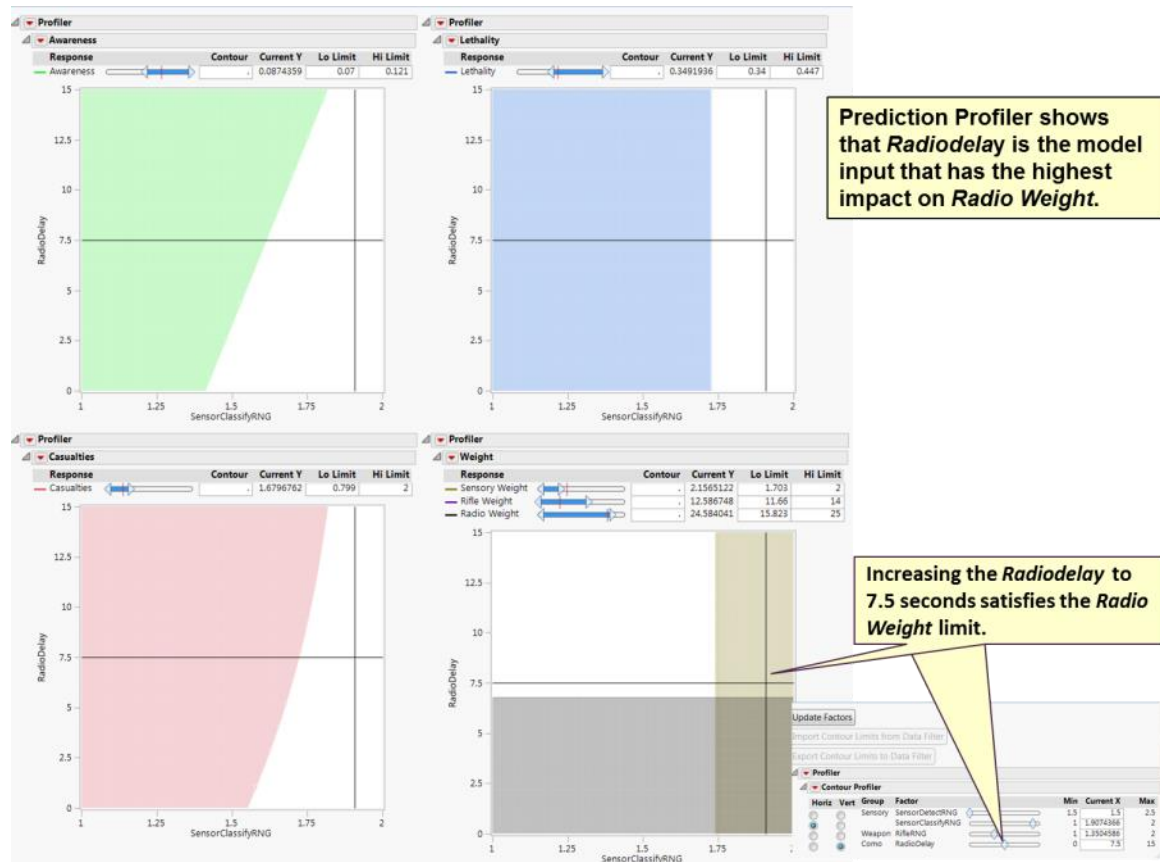


Figure 9.15. Contour profiler dashboard component after manipulating the *Radiodelay*.

The Prediction Profiler indicates that the *SensorDetectRNG* model input has the highest impact on *Sensor Weight* followed by *SensorClassifyRNG*. The contour profiler for the Weight domain indicates that increasing the *SensorDetectRNG* has no impact on the *Sensor Weight*. Changing the *SensorClassifyRNG* to 1.7 satisfies the *Sensor Weight* limit. Figure 9.16 shows these model input changes in the Contour Profiler dashboard component.

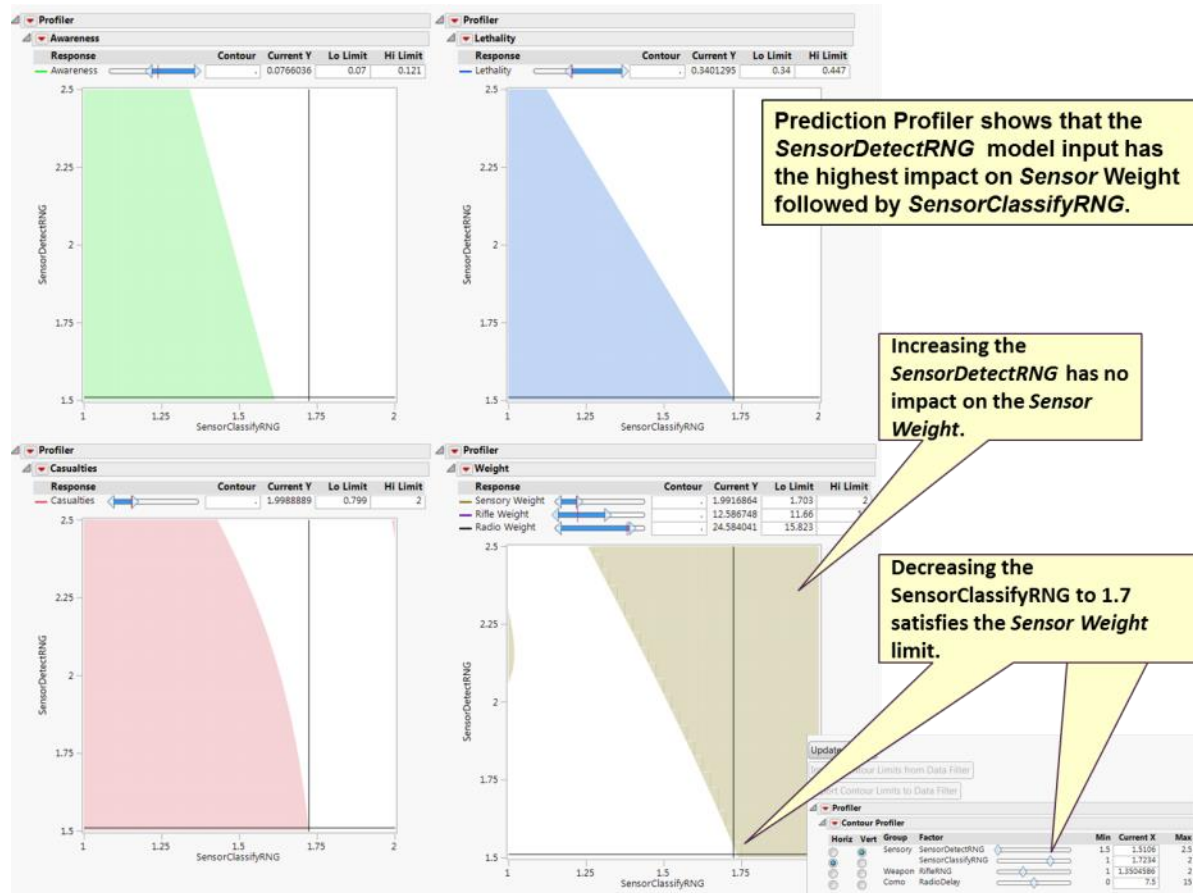


Figure 9.16. Contour profiler dashboard component after manipulating the *SensorClassifyRNG*.

Step 4: Tradeoff infeasible response limits. When no viable variant solution is found by changing model inputs, decide which shaded region response must be traded off to achieve a feasible solution where all contour profiler cross hairs are in the white region. Figure 9.16 shows a feasible solution and therefore we do not need to tradeoff responses. The end result is a white region within each contour profiler domain that represents a set of viable system variants that satisfy all desired effectiveness and constraints.

Step 5: Generate viable variant solution candidates using the Monte Carlo Filtering. In order to acquire the set of viable variants within the white region of the contour profilers we use the Monte Carlo Filtering component. Figure 9.17 shows two scatter plots, the one at the left shows the alternatives generated by the Monte Carlo simulations and the one to the right has the contour profiler response limits exported to the data filter.

Each point represents a simulation run approximation. Metamodels can generate a million alternatives in a few seconds.

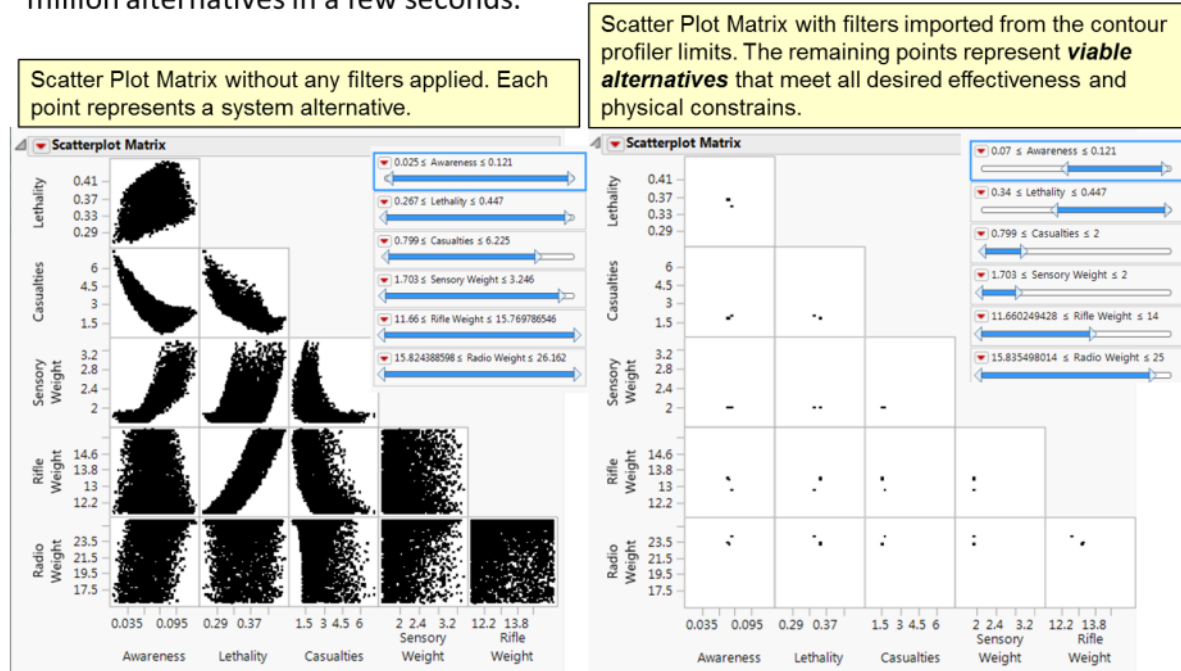


Figure 9.17. Monte Carlo Filtering component with response limit filters applied.

Out of the 5,000 simulations shown in Figure 9.17, 4 alternatives remain when the filters are applied. When we ran 100,000 simulations with the same response limits, 69 alternatives remained after applying the filters. The filtered set of alternative solutions provides the user a reduced set of viable system variants that satisfy the specified desired effectiveness and constraints of the responses. In order to narrow down the reduced set of solutions further, we can select the most affordable solution, the solution with the least amount of variation, or the original optimized solution that was modified to satisfy the response limits.

After completing the above five steps, the dashboard user found a solution based on the response limits and importance weights that were set in Step 1 and 2. In order to acquire additional viable variants, repeat steps 1-5 with a different set of response limits and importance weights from different stakeholder view perspectives and scenarios.

Figure 9.18 shows a screenshot of the Prediction Profiler dashboard component for the large squad enhancement use case. The Figure shows 38 model inputs and 43 responses. Because the model inputs are ordered from left to right based on their collective impact of all the responses, we can easily identify the ones that have the highest impact on the system design. The example shown in Figure 9.18 indicates that the rifle range of the soldier's weapon (*M4RNG*), the number of UAVs (*NOUAV*), and the sensor classification distance (*SDRClassRNG*) have the highest impact across all responses.

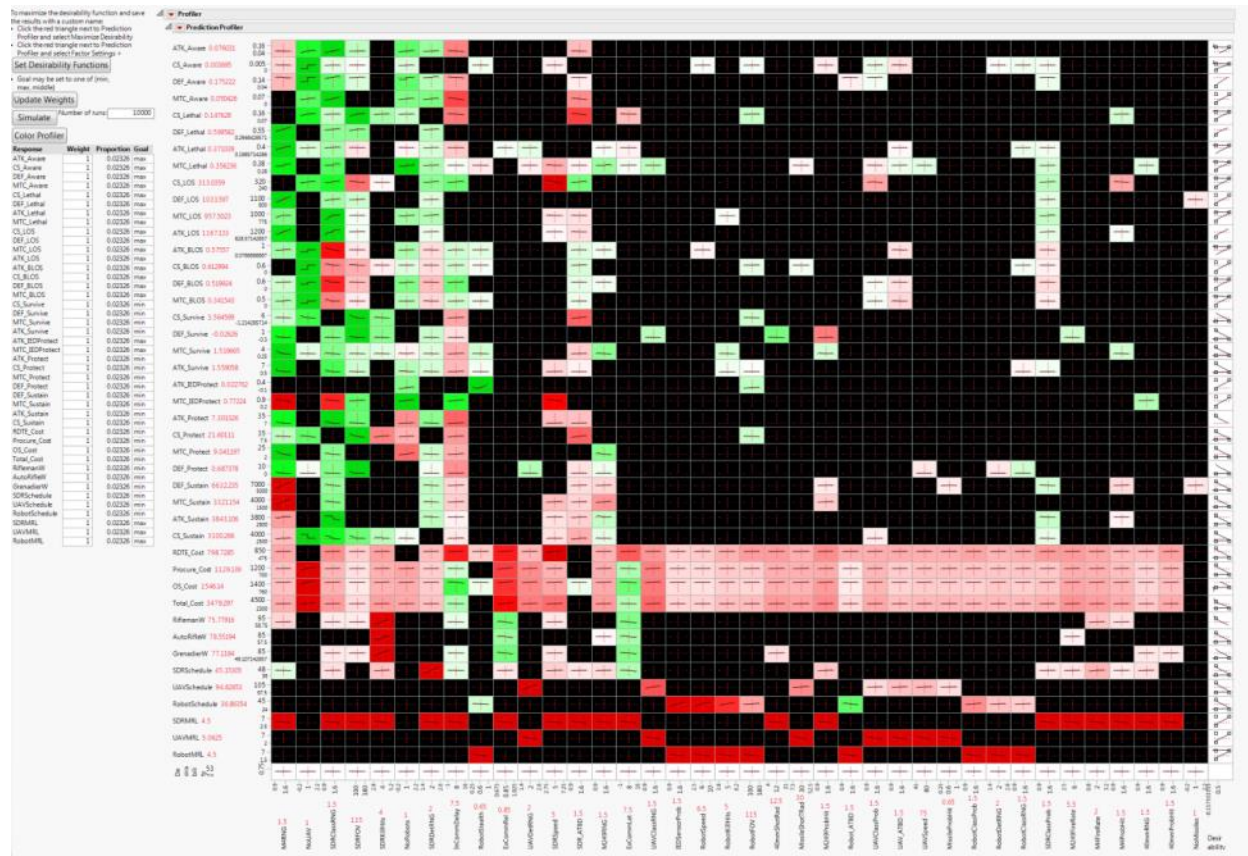


Figure 9.18. Prediction profiler dashboard component for the large squad enhancement uses case.

Identifying the model inputs that have the highest impact across several model domains provides important insights to the system stakeholders. Because model inputs are mapped to value properties, identifying the high impact model inputs help focus our system decisions on the most critical value properties.

The contour profiler dashboard component shown in Figure 9.19 allows decision makers to understand where a system alternative satisfies desired capabilities and constraints on a larger scale. Organizing each response within domains allows users to better understand where they need to make tradeoffs across domains. In addition, the contour profilers show where the system has room to improve the responses; when the cross hair does not reside along a shaded region's edge, there is an opportunity to improve a response.

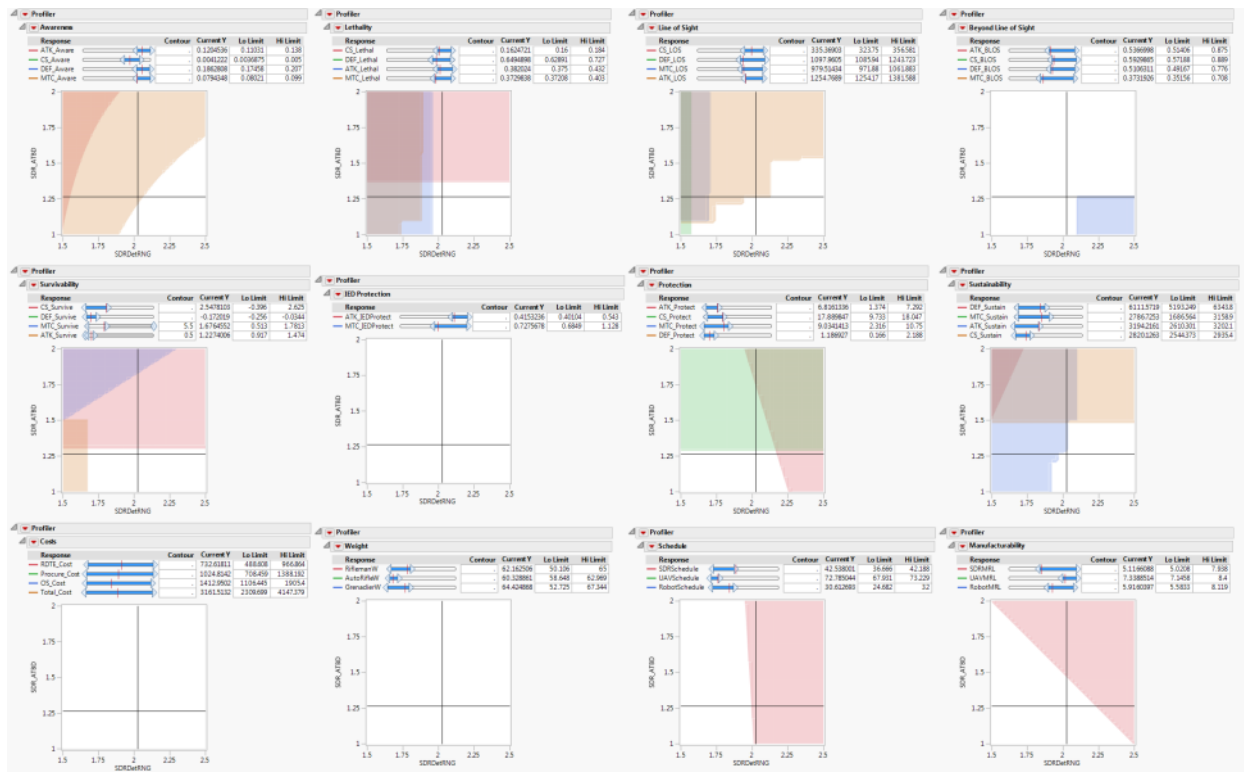


Figure 9.19. Contour profiler dashboard component for the large squad enhancement use case.

Figure 9.20 shows a screenshot of the Monte Carlo filtering dashboard component. The scatter plot matrix is capable of displaying a large number of responses with data filters that narrow in on viable sets of system variants.

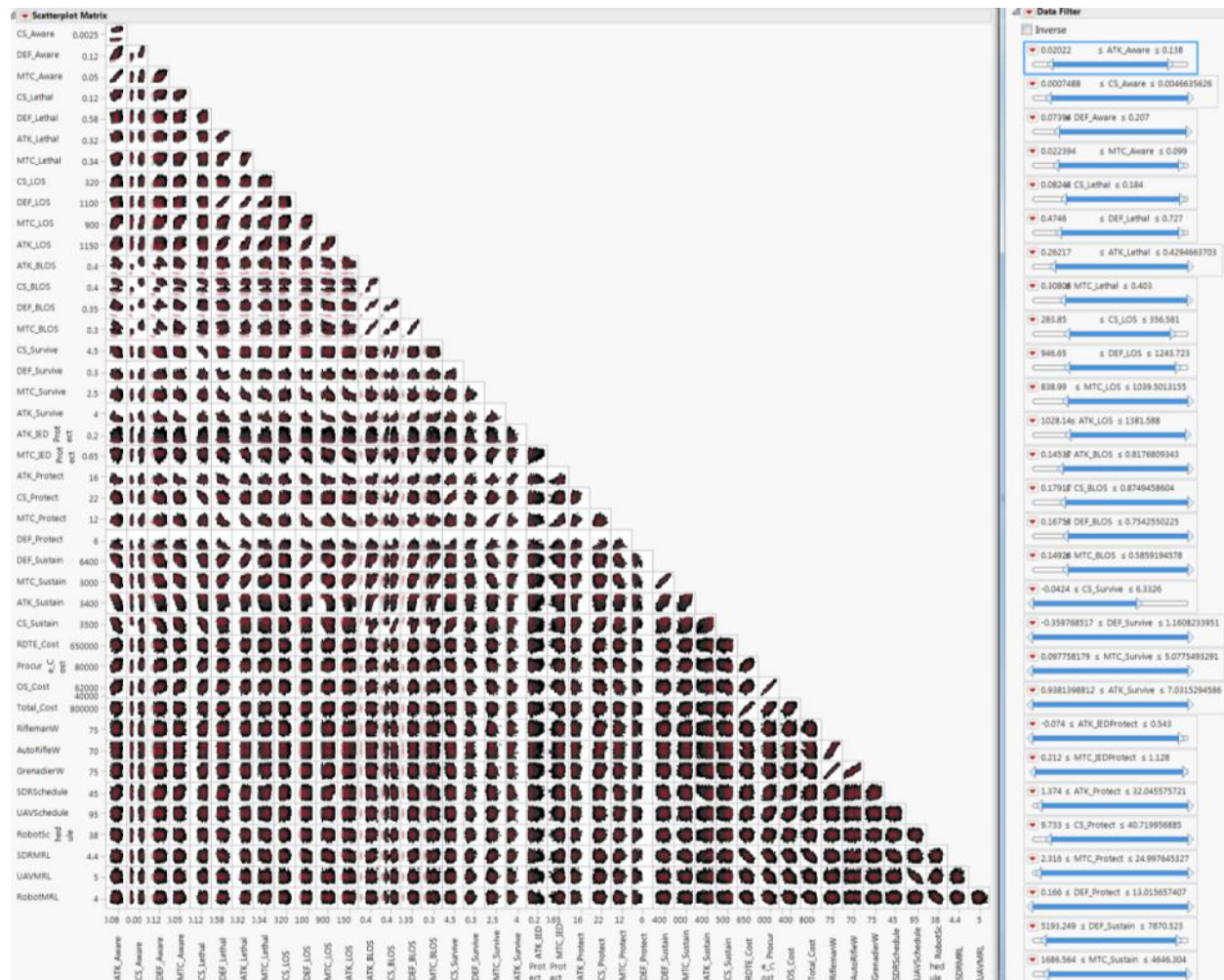









Figure 9.20. Monte Carlo filtering dashboard component for the large squad enhancement use case.

10. Multiple Objective Decision Analysis

In this section, we discuss the importance of evaluating the set of viable variants when there are multiple competing objectives and several different stakeholders. For the squad enhancement use case, we assume that there are seven viable variants found by the dashboard described in the previous section. Depending on where our system is in the lifecycle and who the key stakeholders are, the measures of effectiveness that will evaluate our viable variants will typically not solely rely on simulation model outputs. Our methodology advocates the use of multi-objective decision analysis (MODA) to evaluate each variant; we will refer to these viable variants as alternatives. Specifically, we use the mathematics of MODA and the philosophy of Value Focused Thinking (VFT) (Keeney 1992). The seven alternatives for our use case are shown in Table 10.1. The system features are the value properties for each of the blocks that are labeled as subsystems.

Table 10.1 Notional Viable Variant Alternatives.

Subsystem	System Feature	Baseline	Sustainable	Attack	LongRange	Survivable	Defendable	Performance
								
Subsystem	System Feature	Design Choice	Design Choice	Design Choice	Design Choice	Design Choice	Design Choice	Design Choice
Heads Up Display	Mount Type	NA	Helmet	Goggle	Helmet	Goggle	Integrated	Integrated
	Display Type	NA	2D Small	3D Small	2D Large	3D Medium	2D Medium	3D Large
Lenses	Magnification	0X	0.5X	1X	3X	1X	2X	4X
	Field of View	50 deg	60 deg	80 deg	60 deg	90 deg	100 deg	150 deg
	Range	300 m	900 m	800 m	1300 m	1000 m	1100 m	1400 m
Radio	LOS Range	1000 m	1200 m	1500 m	800 m	1400 m	1300 m	1600 m
	Bandwidth	5 mbps	10 mbps	9 mbps	13 mbps	12 mbps	6 mbps	12 mbps
	Security	Freq Hop	Freq Hop	TK400	TK650	KLM40	PX33	FH98
	Operating Time	12 hrs	14 hrs	10 hrs	24 hrs	16 hrs	10 hrs	18 hrs
	Power Source	5 watts	10 watts	40 watts	30 watts	38 watts	20 watts	25 watts
Rifle	Rate of Fire	700 rpm	700 rpm	1300 rpm	600 rpm	1100 rpm	800 rpm	1500 rpm
	Max Range	500 m	600 mm	700 mm	1200 mm	800 mm	1200 m	1500 m
	Ammo Type	5.56 mm	7.62 mm	12.7 mm	6.5 mm	10 mm	8 mm	12.7 mm
	Ammo Capacity	30 rds	50 rds	150 rds	50 rds	200 rds	300 rds	500 rds
	Muzzle Velocity	880 m/s	1000 m/s	1500 m/s	900 m/s	1100 m/s	950 m/s	1500 m/s
Exoskeleton	Max Speed	4 mph	4 mph	5 mph	5 mph	5.5 mph	6 mph	8 mph
	Power Source	NA	BA90	BA75	Tablet	BA550	Integrated	Integrated
	Frame Size	NA	Small	Medium	Medium	Medium	Large	Large
Body Armor	Protection Level	Type III	Type III	Type III	Type IV	Type III	Type V	Type V
	Coverage Area	Vital Only	Vital & Extreme	Full	Full	Full	Full	Full
	Thickness	2 in	1 in	2.5 in	3 in	2.5 in	3 in	3.5 in
	2.5 in	Kevlar	Kevlar	MX500	P4	LM900	Kevlar III	Telex
	Flexibility	Hard	Soft	Medium	Hard	Hard	Soft	Medium
UAV	Model	NA	Cardinal	Buzzard	Crow	Pigeon	Robin	Dove
Robot Type	Transport System	NA	Track	Track	Quad	Track	Mixed	Quad
	Max Speed	NA	2 mph	2.3 mph	4 mph	2.7 mph	5 mph	8 mph
	Operating Time	NA	24 hrs	8 hrs	20 hrs	5.5 hrs	10 hrs	18 hrs
	Range	NA	600 m	500 m	700 m	300 m	700 m	1000 m
Robot Sensor	Magnification	NA	1X	2X	1.5X	3X	2.5X	2X
	Field of View	NA	50 deg	360 deg	180 deg	80 deg	100 deg	75 deg
	Range	NA	300 m	400 m	200 m	500 m	4050 m	500 m

An important assumption we note is that our value model is developed prior to selecting our alternatives using the dashboard so that we do not bias the value model's development.

Too often when we are faced with a decision problem, we first develop a list of alternatives and limit ourselves to this set of choices; Keeney defines this as alternative focused thinking. Using alternative focused thinking causes us not to think about what we truly value. As a result, we may miss alternatives or we may not discover new decision opportunities that result in a better outcome. VFT emphasizes value development. Rather than focus on alternatives, VFT focuses on structuring the decision problem as a well-defined objective hierarchy that represents what is most important to decision makers. This objective hierarchy is known as the fundamental objective hierarchy, and is the foundation upon which all decisions are based; it is the most essential part of VFT because without a well-defined fundamental objective hierarchy, the results are meaningless (Keeney 1992).

10.1. Qualitative Functional Objective Value Hierarchy

For a systems design problem we use a modified form of the hierarchy known as the functional objective value hierarchy (Parnell et al. 2011). The functional objective value hierarchy has three

elements: the functions the system will perform, the fundamental objectives that define what the system tries to achieve, and the value measures that quantifies each objective. Keeney defines a fundamental objective as an objective that expresses what the decision maker values and qualitatively states what is important. We start with an over-arching fundamental objective and then decompose the objective into the functions and sub-functions that will accomplish the system purpose. We then decompose the functions into objectives that define value until we find a value measure that informs the decision maker how well an alternative achieves the associated objective. The value measure is the criteria used to measure the degree to which an objective is achieved. There is one or more value measure associated with each lowest-level objective. Therefore, the set of value measures comprise the evaluation criterion that measures how we value an alternative.

VFT makes a clear distinction between a fundamental and a means objective. Means objectives explain how we accomplish something essential to the problem. It is important that we emphasize the difference between these two types of objectives to fully understand what the fundamental objectives should be and how we determine them. To test whether an objective is a fundamental or a means objective we must ask the question “why is this important?” If the answer is because it helps us accomplish something fundamental to the problem then it is a means objective. If the answer is because it is one of the essential reasons why we care about this decision, then it is a fundamental objective. Identifying means objectives can help determine fundamental objectives because if the means objective helps accomplish something essential, this “something” could be a potential fundamental objective.

A well-defined functional objective value hierarchy has desirable properties that ensure our value measures can measure how we value alternatives. According to Kirkwood, these desirable properties are completeness, non-redundancy, independence, operability, and minimal size (Kirkwood 1997). This section discusses these properties and indicates the consequences of their violations. It is important to note that if we use fundamental objectives to structure our hierarchies then there is less chance that we violate any of the desired properties.

1. **Completeness:** In order for a functional objective value hierarchy to be complete, all the functions and objectives must exhaustively characterize every aspect of the higher-level function/objective. If we miss identifying a critical characteristic then we do not account for an important aspect of the problem; this leads to faulty results.
2. **Non-redundancy:** A non-redundant value model avoids double counting an alternative’s value. Double counting can occur when redundant functions/objectives or value measures exist. For example, if we include a value measure that is not a critical characteristic of an objective, then we have a redundant value measure that may over-emphasize an alternative. A non-redundant functional objective value hierarchy ensures that there are no unnecessary functions/objectives or value measures that may cause an alternative to be scored higher than intended.
3. **Independence:** An alternative’s achievement level obtained by one value measure must be independent from all the other value measure achievement levels. Specifically, if an alternative’s value score achieved by one value measure depends on the achievement level of

another value measure, then there is a violation of the independence property. Typically when we encounter a problem with independence, we have a means objective in our hierarchy or we have interactions between value measures. To fix this we must redefine the objective as a fundamental objective, or use a multi-dimensional value measure (Ewing et al. 2006).

4. **Operability:** An operable objective has the ability to obtain the data needed to assess an alternative's value. There may be a clear, well defined value measure that measures the achievement level of an objective but if the data are not available, we cannot use this value measure in the model.
5. **Minimal Size:** There is a balance between a high-resolution model with a large set of objectives and value measures and a low resolution, more compact model that only includes the most essential functions/objectives needed to evaluate an alternative. A smaller model can be communicated more easily to senior level decision makers and it is easier to obtain the necessary data to evaluate the alternatives (Kirkwood 1997). The decision regarding how far we should decompose the hierarchy depends on the availability of operable data that can inform the decision. The more we expand the size of the hierarchy, the more detailed the analysis becomes. A larger detailed model can provide meaningful insight but only if these details are available. The hierarchy must stop at the point where there are data available. Typically, a detailed hierarchy includes natural data value measures that have a measurable scale in terms of a quantitative number. When a natural value measure is not available, we use constructed value measures that aggregate categorical information to represent a qualitative measure.

The functional objective value model we used to assess our viable variant alternatives is shown in Figure 10.1. The model has functions, objectives, and value measures that capture every important aspect of the squad enhancement design problem.

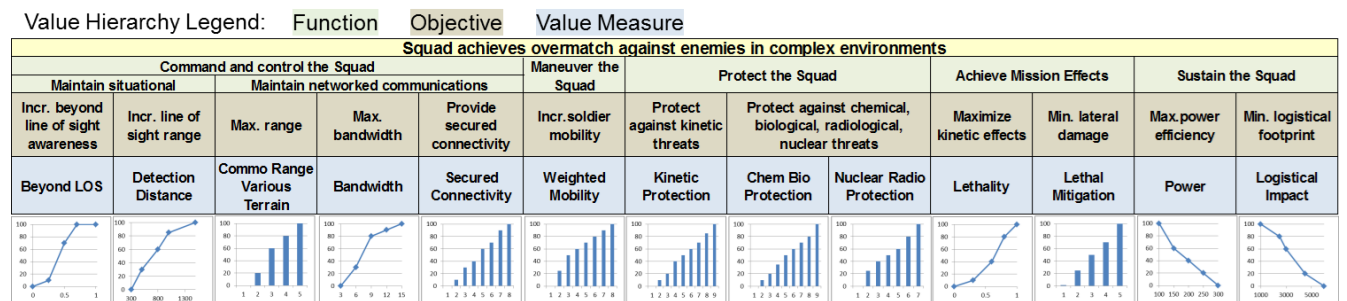


Figure 10.1. Functional Objective Value Hierarchy for the squad enhancement technology use case.

10.2. Quantitative Functional Objective Value Model

Once our functional objective value hierarchy is complete, we then develop our quantitative model by selecting the value measure types, developing value functions, and assigning swing weights to each measure. For each value measure we must transform the measurable scale into a value score. The

measurable scale is the value measure's quantitative unit of measure (miles per hour or probability of detection, for example). The value score is a number between 0 and 10 (which we choose arbitrarily) that represents the attribute's achievement of its associated objective (10 being the best and 0 being the worst). The single-dimensional value function transforms a value measure's scale into a value score. Because there are tradeoffs amongst the value measures, we must weigh their importance against each other. We use swing weights to assess these tradeoffs. Once we formulate both the single-dimensional value functions and assess the value measure tradeoffs, we utilize a multi-objective value function to determine the overall value of an alternative. The multi-objective value function outputs a total value that is a weighted sum of the value scores determined by each single-dimensional value function and swing weights. The total value indicates how much value each alternative attains with respect to the functional objective value hierarchy.

There are two types of value measures, natural and constructed. A natural value measure uses a number that has a common interpretation. A constructed value measure is a description of distinct impact levels that directly indicate the degree to which the associated objective is achieved. Each impact level has a unique category that an alternative can be classified. The development of both value measure types requires military judgments that convey what is important to the decision makers. One is not necessarily better (or worse) than the other. In practice, there are more constructed value measures in a well-defined functional value hierarchy than there are natural value measures. Inexperienced modelers typically place too many natural value measures that are ineffective at clearly measuring the impact level of an objective in the model. Our next two sections describe in more detail the construction of the natural and constructed value functions.

10.2.1. Natural Single-Dimensional Value Functions

Earlier we defined a value measure as the criteria used to measure an impact level. We use the term impact level to mean the degree to which an objective is achieved. Each attribute has a preferred direction of improvement; this usually means "more is better" or "less is better" (Kirkwood 1997). We associate the impact level with a measurable scale that represents the range of numerical values a value measure can assume from its worst to best impact levels. The numerical values typically have different units and ranges within the measurable scale. To ensure that we can compare these value measures relative to each other, we transform the different measurable scales into a common unit of measure called the value scale. The value scale range from 0 to 10, which we choose arbitrarily; we could have just as easily used value scales ranging from 0 to 1 or 0 to 100. The value scale of 0 and 10 represent the worst and best impact levels, respectively. We then examine how the impact level varies between its worst and best levels. Figure 10.2 shows a number of different types of value function shapes that define the returns to scale (RTS) in a number of different ways.

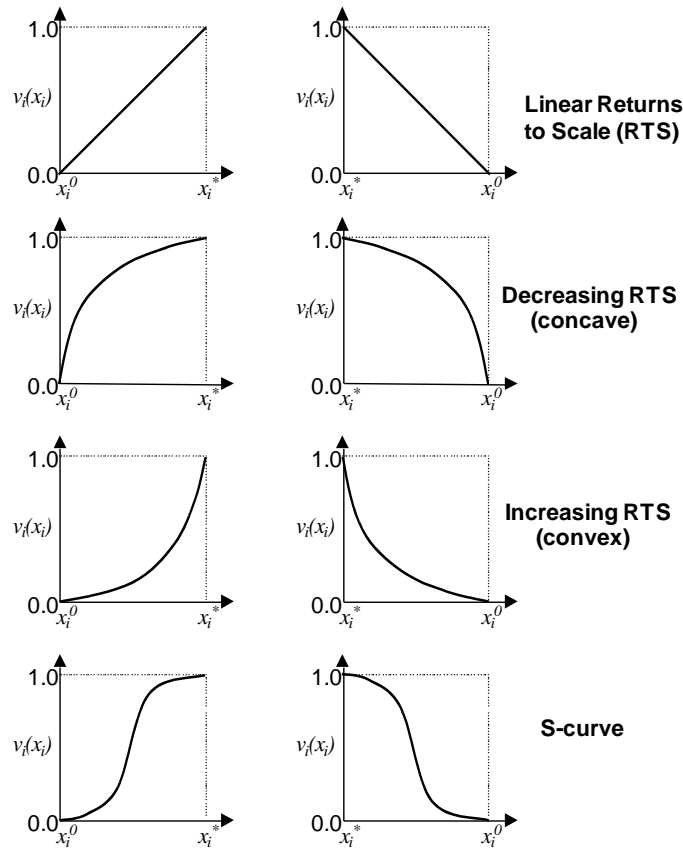


Figure 10.2 Value function curves that define the returns to scale (RTS).

The shape of the value function is often dictated by the requirements defined by the systems engineer as either a desired capability or a constraint. The constraints are what set the minimal acceptable level or walk away point while the desired capabilities help determine the threshold and objective value levels. The threshold level is the point along the value measure scale that we at least want to achieve but will accept a lower level when making trade-off decisions. The objective level is the point where we would hope to realistically achieve.

10.2.2. Constructed Single Dimensional Value Function

This section explains how to formulate and assess a constructed single-dimensional value function. Unlike natural value measure, constructed value measures do not have numbers with a common interpretation. Because there are no numbers that define the measurable scale, we develop categories to construct this scale. Each category has a narrative description of the impact level which indicates the achievement of an objective. These categories should contain all that we value with respect to the associated value measure and they should range between the worst and best cases.

Unlike the natural value measure, where we may have sets of ranges within a measurable scale with different value increments, constructed value measures have value increments between categories. To find the value increment between each category, we list them from worst to best. We assess the

constructed value function by finding the smallest value increment between any two neighboring categories. For example, in Table 10.2, categories 1 and 2 are neighbors, as are categories 2 and 3. We assign the smallest value increment to 1 and use this as a unit of measure to assess how much more we value any other two neighboring categories.

Table 10.2. Value increments and value scores for categories.

Category Number	Category Description	Value Increment	Value Score
1	Category 1		0
		1	
2	Category 2		3.33
		2	
3	Category 3		10.00

Among all neighboring categories in Table 10.2, the smallest value increment is between Category 1 and Category 2. The value increment between Category 2 and Category 3 is two times more valuable than the smallest value increment. The value scores are a function of the established value increments.

To demonstrate how to calculate the value scores as a function of the value increments, we add the products of each value increment and a variable v , set the expression equal to 10 and solve for v ($v + 2v = 10$). We use the value of v to calculate the value score for each category. The term $Score(Category_n)$ represents the value score for category n where n ranges from 1 to the total number of categories. $Score(Category_1)$, the worst case category, always equals 0. The remaining attribute values are the cumulative sum product of the value increment between a category's preceding neighbor and the current category, and v . Equations 5 through 7 shows the value score calculations for the example in Table 10.2.

$$Score(Category_1) = 0.00 \quad (5)$$

$$Score(Category_2) = Score(Category_1) + v = 0.00 + 3.33 = 3.33 \quad (6)$$

$$Score(Category_3) = Score(Category_2) + 2v = 3.33 + 3(3.33) = 10.00 \quad (7)$$

While developing the categories for each of the constructed value measures we ensure that each alternative can only be classified into one of the existing categories. If we use ambiguous definitions then the value model may return inconsistent results because different people may categorize an alternative differently. Clear category definitions ensure reproducible results because they eliminate ambiguity. The two possible pitfalls that arise from ambiguous definitions are the following:

1. An alternative receives no value towards an objective because the constructed value measure's categories are not an exhaustive representation of all significant impacts.
2. The constructed value measure categories are not mutually exclusive. Therefore, an alternative may fit into more than one category.

For the squad enhancement use case we developed 13 value measures with a mix of natural and constructed value functions. Table 10.3 lists each of the value measures and their types.

Table 10.3. Value measures for the squad enhancement problem.

Function	Objective	Value Measure	Type	Minimal Acceptable Value	Ideal Value
Maintain situational awareness	Increase beyond line of sight awareness	Beyond LOS (% detected)	Natural	0	1
	Increase line of sight range	Detection Distance (meters)	Natural	300	1500
Maintain networked communications	Maximize range	Communication Range in Various Terrain	Constructed	1	5
	Maximize bandwidth	Bandwidth (mbps)	Natural	3	15
	Provide secured connectivity	Secured Connectivity	Constructed	1	8
Maneuver the Squad	Increase soldier mobility	Weighted Mobility	Multi-Dimensional Constructed	1	8
Protect the Squad	Protect against kinetic threats	Kinetic Protection	Constructed	1	9
	Protect against chemical, biological, radiological, nuclear threats	Chemical Biological Protection	Constructed	1	9
		Nuclear Radiological Protection	Constructed	1	7
Achieve mission effects	Maximize kinetic effects	Lethality (% enemy killed)	Natural	0	1
	Minimize lateral damage	Lethal Mitigation	Constructed	1	5
Sustain the Squad	Maximize power efficiency	Power (kw/h)	Natural	300	100
	Minimize logistical footprint	Logistical Impact (rounds fired)	Natural	6000	1000

10.2.3. Swing Weights for Value Measure Tradeoffs

Because we have multiple and competing objectives, each assessed by one or more value measures, we must determine the tradeoffs between these value measures. We do this by assigning global weights to each value measure so that we can utilize the multi-objective value function. There are several weight-assessment approaches available to determine the tradeoffs between value measures. We use an approach called the Swing Weight Matrix (Parnell, Bresnick, Tani, & Johnson, 2013).

We assign weights to each value measure by using swing weights. A swing weight assesses a value measure based on how important it is to swing from the measure's worst impact level to its best impact level along the measurable scale (Clemens 2001). Our selection has nothing to do with the magnitude of the scales; we base our choice on a subjective determination that considers swinging from the worst to the best impact levels. The swing weight we assign is based on where it is placed inside the swing weight matrix. The columns in the matrix allow us to define categories of subjective importance that express the impact on the overall fundamental objective. The rows of the matrix classify the value measure's impact on capability and are based on the range of the value measure scale (large, medium, or small

capability gap between the walkaway and the ideal levels). For the squad enhancement use case, the subjective importance depends on whether the value measure assesses an objective that is mission critical, enables, or enhances the system’s capabilities. Typically, we prioritize value measures differently depending on the scenario the system is used in. For example, we would prioritize value measures that assess mobility and lethality more for an attack scenario than we would for a defense scenario. Tables 10.4 and 10.5 show the swing weight matrices for the squad enhancement attack and defense scenarios where the columns categorize the importance while the rows bin the impact on the capability increase based on the value measure’s range from the minimum acceptable level and the ideal level.

Table 10.4. Swing weight matrix for the attack scenario.

	Mission Critical			Enables Capability			Enhances Capability		
Capability Impact		Matrix Weight	Swing Weight		Matrix Weight	Swing Weight		Matrix Weight	Swing Weight
Significant Impact	Lethality	100	0.14	Weighted Mobility	70	0.10	Power	20	0.03
	Lethal Mitigation	90	0.13				Logistical Impact	20	0.03
	Beyond LOS	90	0.13						
	Kinetic Protection	70	0.10						
Medium Impact				Bandwidth	50	0.07	Chemical Bio Protection	15	0.02
	Secured Connectivity	65	0.09						
	Communication Range	60	0.09	Detection Distance	50	0.06			
Minimal Impact							Nuclear Radio Protection	5	0.01

Table 10.5. Swing weight matrix for the defense scenario.

	Mission Critical			Enables Capability			Enhances Capability		
Capability Impact		Matrix Weight	Swing Weight		Matric Weight	Swing Weight		Matrix Weight	Swing Weight
Significant Impact	Kinetic Protection	100	0.15	Beyond LOS	70	0.11	Lethality	30	0.05
	Lethal Mitigation	90	0.14	Lethality			Power	20	0.03
	Detection Distance	90	0.14				Weighted Mobility	20	0.03
							Logistical Impact	20	.03
Medium Impact	Chemical Bio Protection	65	0.10	Secured Connectivity	50	0.08	Communication Range	10	0.02
				Bandwidth	40	0.06			
Minimal Impact	Nuclear Radio Protection	60	.09						

To obtain the value matrix swing weights, we first use matrix weights that are placed in each cell of the swing weight matrix. To ensure there is a proper range of weights between the highest and lowest value measure, we use matrix weights that range from 5 to 100. The lowest weight is placed in the lower right cell and the highest weight is placed in the upper left cell. The value measures that are placed in a cell get assigned the matrix weight in that cell. We use these matrix weights to calculate a value measure's *global weight*. Global weights represent the importance of a value measure relative to the others and are values between 0 and 1; the sum of all the global weights is 1. After placing the value measures inside the matrix, we now can calculate the global weights by dividing a value measure's matrix weight by the sum of all the value measures' matrix weights.

10.2.4. Multi-Objective Value Function

To obtain the total value score of an alternative, we use the additive value model (Equations 8 and 9). Each single dimensional value function can be normalized and then weighted by each swing weight contribution, w_i . The total value is the weighted sum of the single dimensional value. The swing weights sum to 1.

$$v(x_j) = \sum_{i=1}^n w_i v_i(x_{ij}) \quad (8)$$

$$\sum_{i=1}^n w_i = 1 \quad (9)$$

i = index on the value measures, $1, \dots, n$

j = index on the alternatives, $1, \dots, m$

$v(x_j)$ = the multidimensional value of value measure i

x_{ji} = alternative's score in the i^{th} value measure

$v_i(x_{ji})$ = normalized single-dimensional value of the score of x_{ji}

w_i = normalized swing weight of i^{th} value measure

The multi-objective value function is an additive function because it has no multiplicative terms in the expression. The independence property of the functional objective value hierarchy allows us to use an additive function. There are other functional forms in decision theory that try to account for dependence between value measures. These functions are difficult to implement and rarely used in practice. Therefore, to ensure we have an additive model, we use functional objective values hierarchies that we measure with independent value measures. Because of the independence property, we can sum the weighted attribute values to find the total value of an alternative (Keeney 1992).

10.3. Value and Cost Tradeoff Analysis

Our next section discusses how to integrate the alternative total value analysis with cost analysis. We first will describe how trade-off analysis is normally conducted deterministically and then discuss the importance of incorporating uncertainties. Prior to our analysis, we assumed that we have already performed additional simulation runs on the viable variant alternatives found using the dashboard in the previous section. These additional runs allow us to confirm the results obtained from the surrogate metamodels and provide the distribution of alternative outcomes for the value measures that use simulation data.

One way to understand how each alternative achieves each of the objectives in the value hierarchy is to use a value component chart. Figure 10.3 shows the value components of each alternative as a stacked bar chart. To the right we see the *Ideal* alternative that represents the maximum possible value score, derived from the swing weights for each value measure. The *Hypothetical Best* alternative is the maximum value achieved for each value measure in the set of alternatives. The difference between the *Ideal* and *Hypothetical Best* is the value gap that the set of alternatives cannot achieve with existing technologies. Depending on its size, we may want to consider including other new alternatives that close the gap. In addition, we may have an opportunity to combine components from the existing alternatives to create new alternative. When we reconfigure system components to create new alternatives we must consider the system integration challenges that may result.

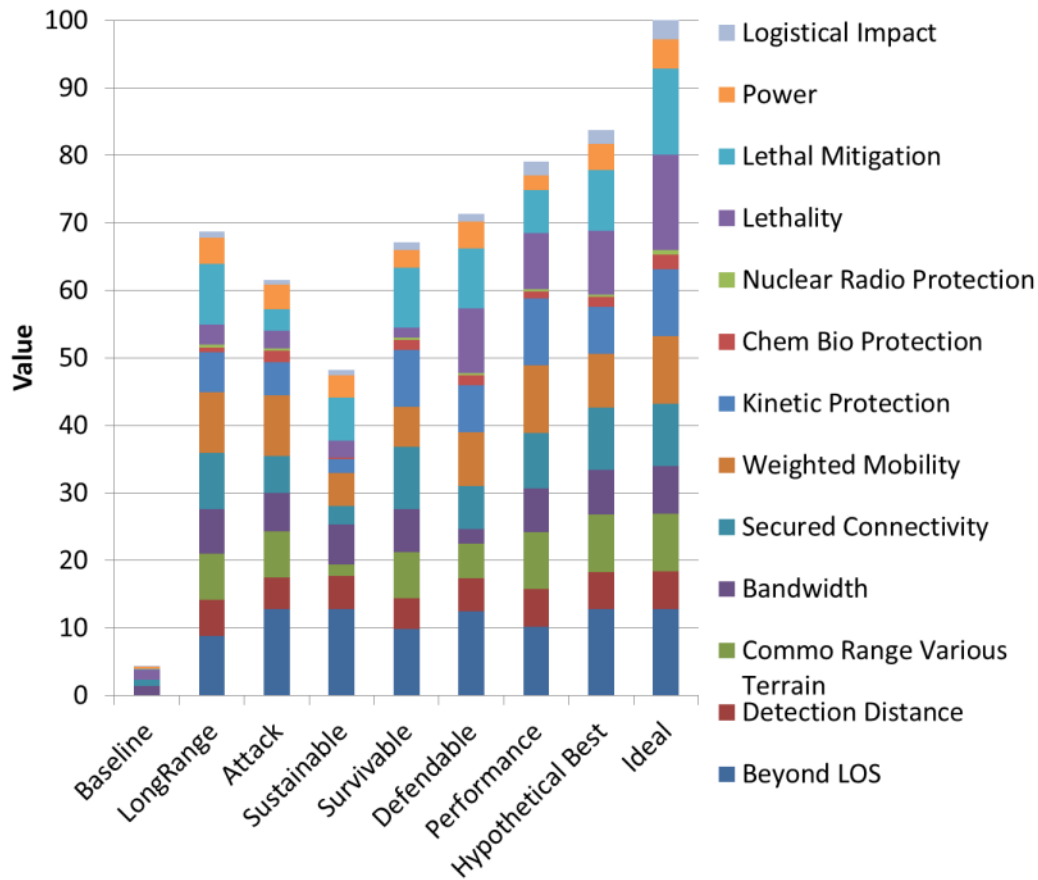


Figure 10.3. Value component chart.

Because lifecycle cost is such an important aspect of any systems decision, we often exclude cost as a value measure and treat it as an independent variable. An effective way to understand the trade-offs between value and cost is to use a Pareto chart. The Pareto chart is a scatter plot with cost on the horizontal axis and value on the vertical axis; each dot represents an alternative's total lifecycle cost and total value score. Figure 10.4 shows a deterministic cost versus value Pareto chart using the average costs and value scores. We can see that the *Sustainable* and *Survivable* alternatives are deterministically dominated by all the others. It makes no sense to select a dominated alternative when we can select non-dominated alternative with a higher value for less cost.

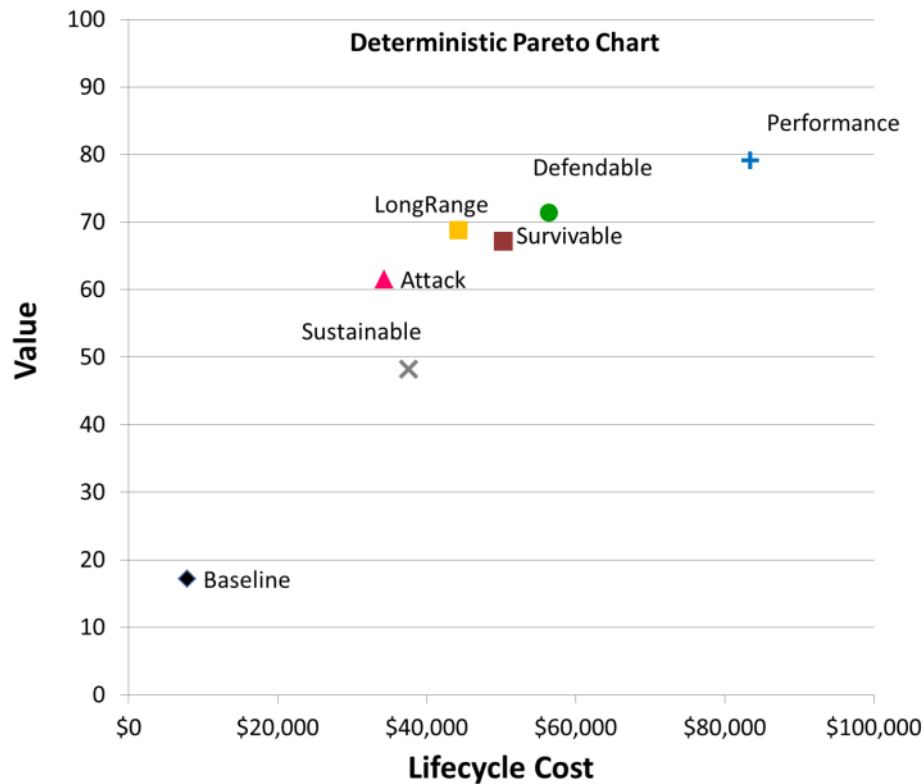


Figure 10.4. Value component charts for the attack and defense scenarios.

However, Figure 10.4 does not consider the uncertainty associated with each alternative's value and cost and does not show the risk, the probability of a lower value. The majority of value trade-off studies in the literature are deterministic. Eliminating deterministically dominated alternatives without considering risk may lead to the wrong decision. Cost estimation techniques already incorporate uncertainties using Monte Carlo simulations and other methods. A major contribution of our approach is that we simultaneously integrate value trade-off uncertainties and cost uncertainties in order to facilitate better value and risk identification. If we do not consider how much variation there is in the consequences of our decision, we may end up making the wrong decision.

When faced with an uncertain system decision, we can leverage three types of analytical charts that help the systems engineer understand the risk associated with a decision and identify what drives the uncertainties in each alternative. First, stochastic Pareto charts identify non-stochastically dominated alternatives with respect to value and cost; second, cumulative distribution function charts (S-curves) compare the alternative risk profiles; and third, tornado charts identify the value measures and cost components that have the highest impact on the alternative uncertainties. We will now describe each of these charts separately and use the squad enhancement problem to provide examples of the insights we can obtain from them with respect to value and cost.

Stochastic Pareto Chart. In order to address the limitations of the deterministic Pareto chart, we create a stochastic Pareto chart, shown in Figure 10.5, by displaying a two dimensional box plot for each alternative's cost and value. The boxes along each axis represent the 2nd and 3rd quartiles while the

lines represent the 1st and 4th quartiles of the vector output data from the value and cost models. We can create these box plots in Microsoft Excel using a scatter plot with a combination of four data series for each alternative, two for the cost axis and two for the value axis. The lines, otherwise known as whiskers, are created from the vector data using the maximum and minimum data points. The boxes are created using the 3rd quartile, mean, and 1st quartile; to create the box, increase the line style width of the data series. The box plots allow us to understand the uncertainty associated with each alternative's cost and value simultaneously.

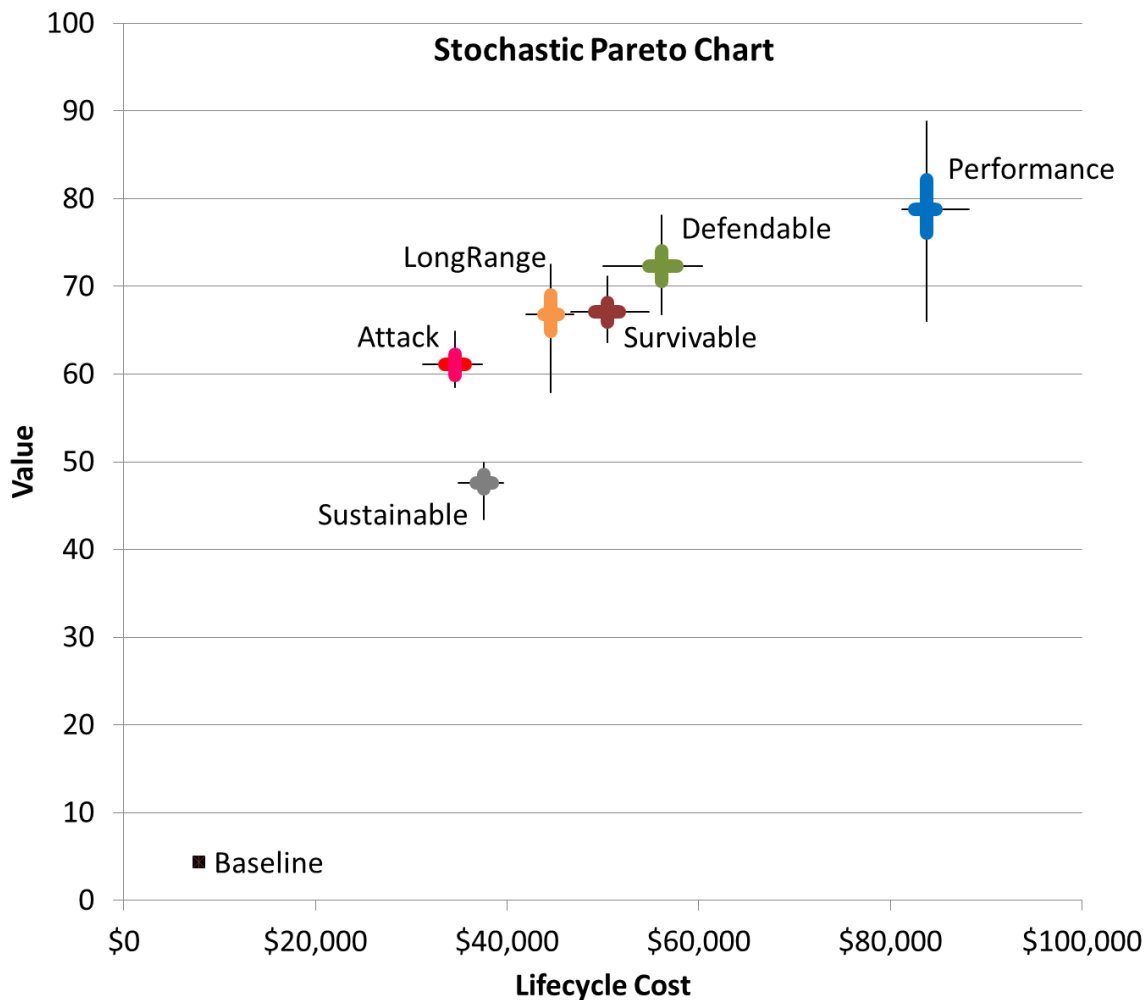


Figure 10.5. Stochastic Pareto Chart

The stochastic Pareto chart allows us to consider value, risk, and dominance simultaneously. In addition, it provides important information for affordability analyses. We can see in Figure 10.5 that *Sustainable* is stochastically dominated by all other alternatives and can be eliminated from consideration. If *Performance* or *Defendable* are affordable we then focus on understanding what drives their uncertainty to mitigate risk. If *Defendable* is not affordable, we then consider either *LongRange* or *Survivable*. If we used the deterministic Pareto chart from Figure 10.4 to eliminate the *Survivable* alternative as a dominated solution, we would have missed an important trade-off consideration. We

can see in Figure 10.5 that *LongRange* has a higher risk in value (probability of lower value) compared to *Survivable*. We may want to accept a higher cost by choosing *Survivable* to mitigate the risk associated with *LongRange*. Of course, an alternative is to choose *LongRange* to reduce the risk. In order to better understand the risk implications, we can use cumulative distribution function charts to compare alternative risk profiles.

Cumulative Distribution Function (S-Curve) Charts. The cumulative distribution function (cdf) chart displays an alternative's potential outcomes by accumulating the area under the outcome's probability mass functions for discrete data and the probability density functions for continuous data. Typically, the shape of the line in the chart is an S-curve that depicts the probability that the outcome will be at or below a given value. The horizontal axis has the outcome scale, either value or cost, while the vertical axis has the probability. Figure 10.6 shows a cdf chart with six S-curves that represents the uncertain alternative value outcomes, otherwise known as the risk profiles. Sustainable is deterministically dominated by the other five alternatives. Attack is deterministically dominated by LongRange, Defendable, and Performance. *Survivable* is deterministically dominated by *Defendable* and *Performance*. The *Performance* and *Defendable* alternatives stochastically dominate all others because their S-curves are positioned completely to the right of all others. The risk profiles of *Survivable* and *LongRange* value cross indicating that there is no clear winner between the two; we may want to accept a higher cost by choosing *Survivable* to mitigate the risk associated with *LongRange*. The cdf chart tells us that there is a 43 percent chance *Survivable* will outperform *LongRange* and that *Survivable* has less risk due to its steeper risk profile. In general, when the risk profiles of alternatives cross we then consider risk preference (risk averse, neutral, or risk taking) during our system decision. A risk adverse decision would spend more for *Survivable* to guarantee a higher value while a risk taking decision would select *LongRange* to save money with the risk of achieving less value.

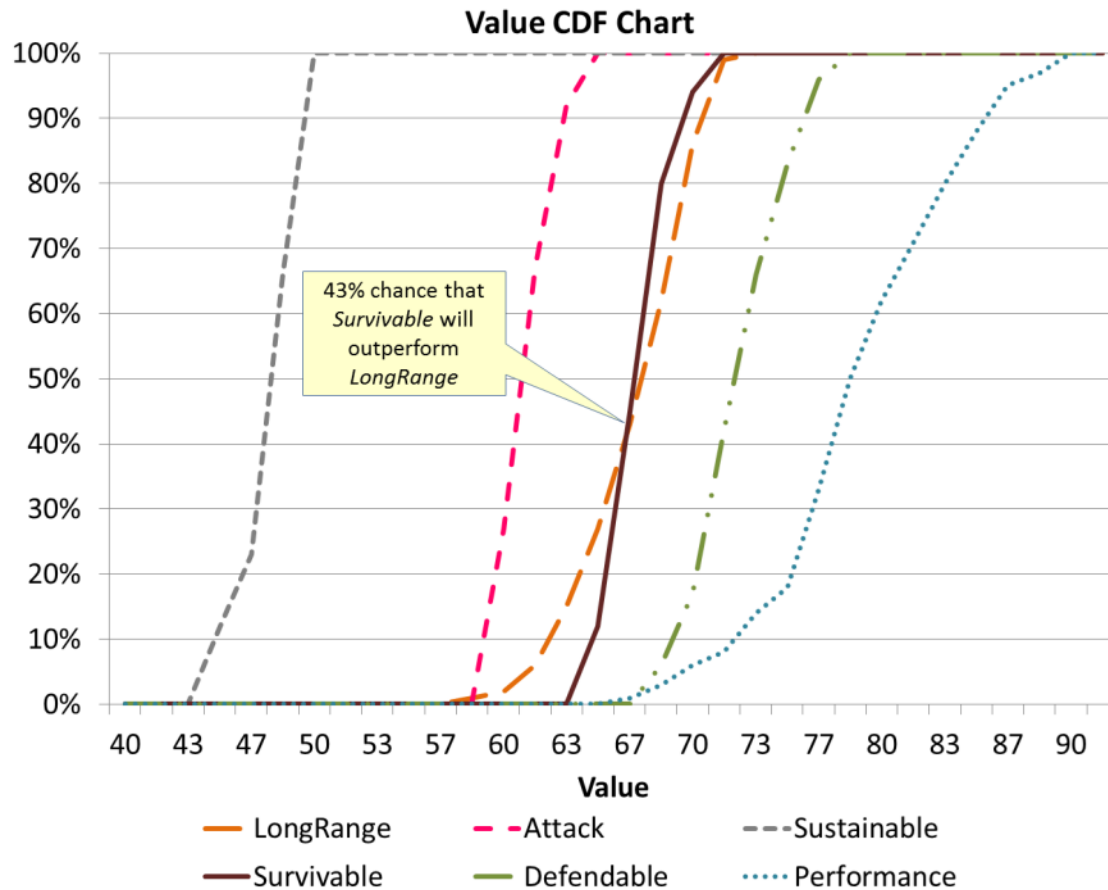


Figure 10.6. Value cumulative distribution chart.

When we want to understand how to best mitigate an alternative's risk, we can use tornado diagrams to identify the value measures and cost components that have the highest impact on value and cost respectively.

Tornado Diagrams. An effective way to identify the impact of uncertainty is to perform sensitivity analysis using tornado diagrams. Tornado diagrams allow us to compare the relative importance of each uncertain input variable with horizontal bars; the longer the bar the higher the impact on the output variable's variation. The bars are sorted so that the longest bars are at the top; sorting the bars in this way makes the diagram look like a tornado. The length of the bars depends on the type of tornado diagram. Deterministic tornado diagrams vary each input variable using low, base, and high settings while all other input variables are held constant. A stochastic tornado diagram uses the vectors of input and output variable trials from a Monte Carlo simulation (Parnell, Bresnick, Tani, & Johnson, 2013). The low end of the bar is the average output variable from the subset of trials where the input is less than a specified lower percentile. Similarly, the high end of the bar is the average output variable from the subset of trials where the input is greater than a specified higher percentile. For the squad enhancement problem, we use stochastic tornado diagrams with a low percentile of 0.3 and a high percentile of 0.7. Figure 10.7 shows the value and cost tornado diagrams for the *Performance* alternative. The input

variables for value are the value measures and the input variables for cost are the cost components. The horizontal axis shows each input variable’s impact on the total variation for the value and cost. We can see in Figure 10.7 that the LethalMitigation value measure has the highest impact on the value’s variation while the Lenses cost component has the highest impact on the cost’s variation.

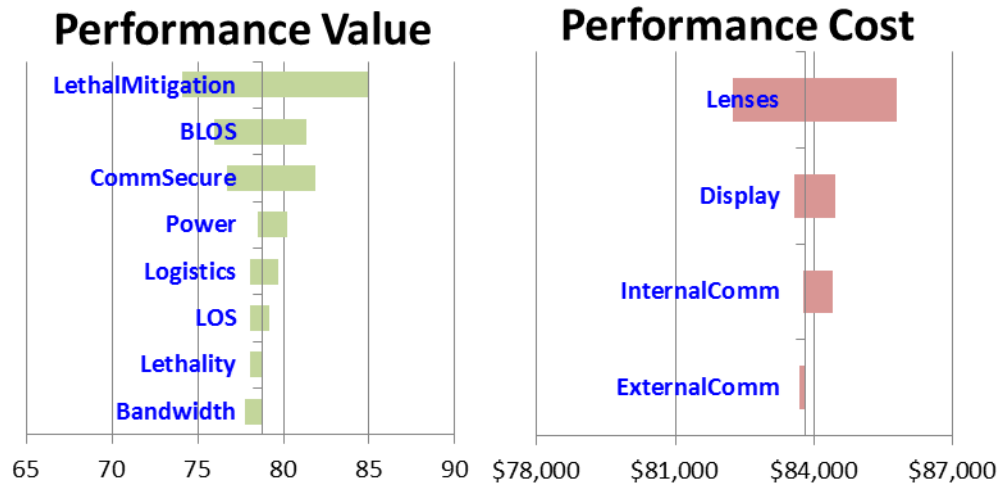


Figure 10.7 Tornado diagrams.

Prior to performing the integrated trade-off analysis, we allocated functions and cost components to subsystems. Our value hierarchy contains objectives that assess the performance of functions and value measures that define how well an alternative achieves the objectives. As a result, the value measures are indirectly allocated to system features through the objectives and functions. Cost components are generally allocated directly to subsystems. Because of these indirect and direct allocations, we can use tornado diagrams to identify the system features that have the highest impact on the system decision. Figure 10.8 illustrates how we use tornado diagrams to trace high impact value measures and cost components to system features.

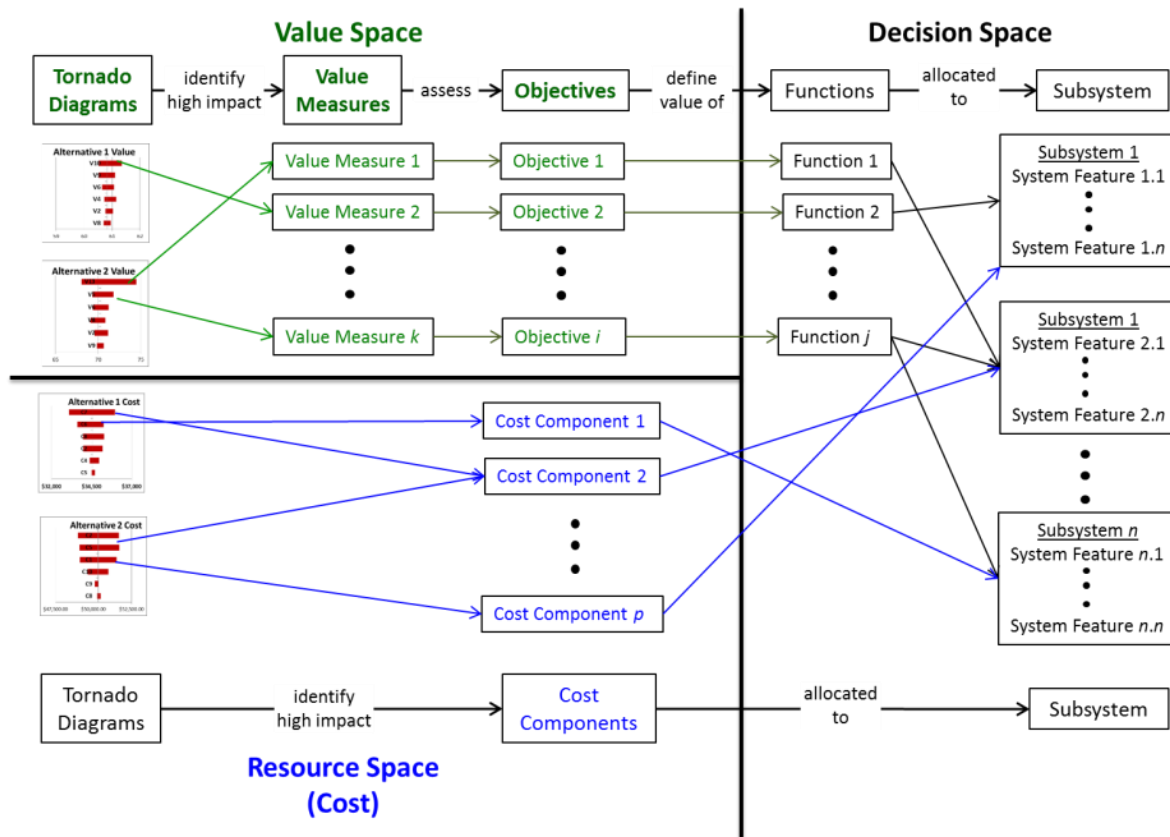


Figure 10.8 Value measure and cost component linkage to system features.

As we learned from our stochastic Pareto chart and cdf charts, we do not have a clear winner between the *Survivable* and *LongRange* alternatives. To help understand how these alternative uncertainties impact the system decision, we can use tornado diagrams to identify the system features that are driving risk. Figure 10.9 shows the value measures and cost components' indirect and direct allocations to the subsystems and their system features.

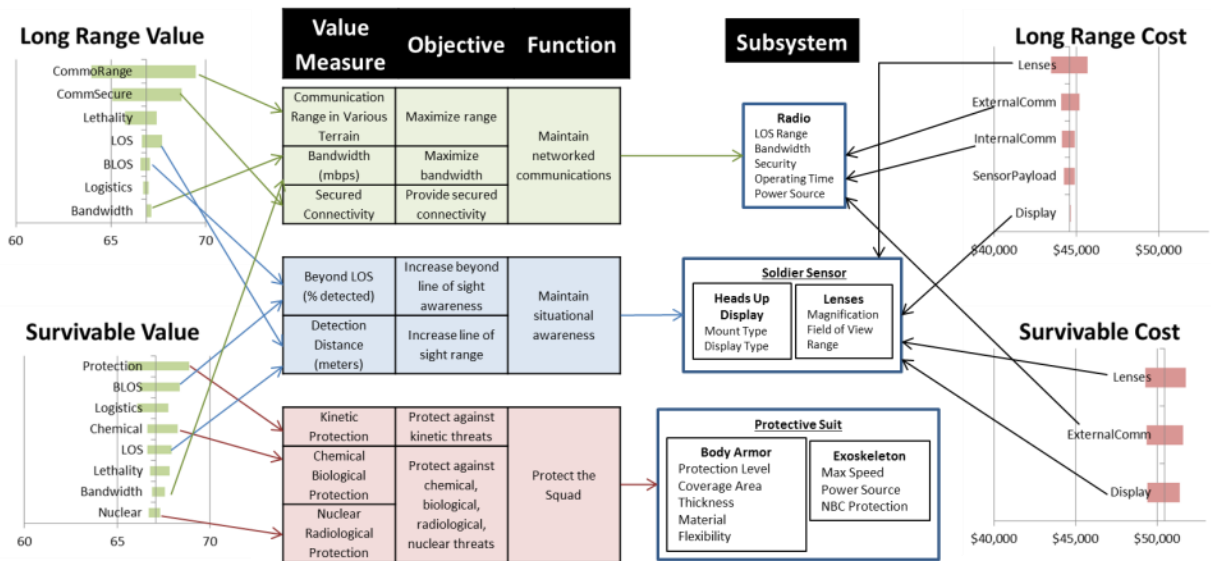


Figure 10.9. Example of value measure and cost component linkages to system features.

We can see from the top bar of the tornado diagrams in Figure 10.9 that the radio drives the majority of the risk for the *LongRange* value while soldier sensor drives the majority of its cost. The protective suit drives the majority of the risk for the *Survivable* value while the soldier sensor drives the majority of its cost. These subsystems and the system features that characterize them have the highest impact on the system decision. These insights provide a clearer understanding of what drives the alternatives' risk and how to prioritize system feature refinements. For example, the systems engineer can reduce the risk of the *LongRange* alternative by investing more resources into improving the radio's range, security, and bandwidth features.

11. Conclusions and Future Work

Our final section reviews the steps in our proposed MBSE methodology and how it fills the ERDC tradespace technical gaps. We conclude with some closing remarks and propose future research that will further advance how we can illuminate trade decisions within the ERS Architecture.

11.1. MBSE Methodology Review

The pre-existing models needed prior to starting the procedural workflow of the methodology is an MBSE integrated system model, a collection of external models and simulations that the systems engineers will use to analyze a system across multiple domain, and a value model that will evaluate the system's effectiveness. The following outlines the procedural steps of our proposed MBSE methodology and the sections within the technical report that describes their details:

1. **Select system model element design variables (Sections 5 and 6).** Based on the problem domain and research questions, select the structural elements and design variables from the MBSE integrated model and define the experimental design region that will be explored with the external models.
2. **Select models, develop basecase scenarios, map design variables to model inputs (Section 6s and 8).** Select the collection of external models that evaluate systems within the variety of domains that are relevant to the stakeholders. For each external model, develop a baseline scenario with the desired model input settings that represent the system baseline. Map the value properties from structural blocks of the system model to the simulation model inputs that directly represent them. When necessary, use translators that link value properties to a model input when there is not a direct mapping. These linkages are established with either lookup tables, analytical equations, or other models.
3. **Create the experimental design (Sections 7 and 8).** Select the experimental design type that will explore each model and decide the appropriate amount of experimental runs needed to perform the analysis. Create three experimental designs with a mix of continuous, discrete, and categorical columns that will represent each of the simulation model inputs using the custom design builder. The data from the first design will be the training set that fits the surrogate metamodels. The second will be the validation set used to select the metamodel complexity and size. The third design will test the predictive performance of the metamodel with a new set of data not used during the training and validation.
4. **Generate the study file that specifies which model input parameters to change (Section 8).** Use a software mechanism to find, select, and modify the model input parameters of the basecase scenario to create a set of excursions, one for each experiment or row in the design matrix.
5. **Generate excursion files for each experiment (row) in the design matrix (Section 8).** Use a software mechanism that generates the excursion files for each experimental run (row in the design matrix). For stochastic simulations, ensure that an adequate number of replications are performed.
6. **Execute HPC simulation runs (Section 8).** Perform a design of experiment on each of the external simulation models using High Performance Computing Clusters. Use a software simulation run queuing mechanism that enforces a scheduling policy and priority scheme while monitoring the computer resources that will complete the simulation runs.
7. **Post-process output files to extract MOEs (Section 8).** Post-process the output data in order to acquire the desired output measure responses needed to gain insights and populate the value model. These responses are also known as measures of effectiveness. Extract, aggregate and append the responses to the experimental design.
8. **Perform statistical metamodeling (Sections 7 and 9).** Use a statistical software package and the experimental design with response model output columns appended, to perform the following tasks:
 - a. Fit polynomial regression models to identify the model inputs that are the system drivers, where synergies/interactions exist, and where there are points of diminishing or increasing rates of return. Populate the MBSE system model with derived requirements developed from these experimental design insights.

- b. Fit a variety of surrogate metamodels of different types using the cross validation technique. Compare metamodel fits and select the highest performing model for each simulation output.
9. **ERS tradespace visualization (Sections 9 and 10).** Use a Prediction Profiler dashboard component to identify the key tradeable variables that have the highest impact on the system design decisions. To identify and analyze a set of viable system variants, perform the following steps:
- a. Set the minimum desired effectiveness and feasibility constraints to the contour profilers as low and high limits as needed.
 - b. Set the importance weights for each simulation model output and run the optimization algorithm to find a system alternative.
 - c. Explore changes to model input settings to find feasible solutions. Examine the contour profilers to identify where the alternative does not meet the desired effectiveness or feasibility constraints; this occurs where the contour profiler crosshair is in a shaded region. Examine the prediction profilers to find the model inputs that have the highest impact on the shaded regions that represent the model output limits. Explore changes to these model inputs to find a viable alternative. When appropriate, increase the desired effectiveness for all responses so that the shaded region is positioned at the edge of the cross hair.
 - d. Tradeoff infeasible response limits. If no viable alternative is found, decide which shaded region model output must be traded off to achieve a feasible solution where all contour profiler cross hairs are in the white region. Name the alternative.
 - e. Generate viable variant solution candidates using the Monte Carlo Filtering. To acquire a reduced collection of viable system variants that satisfy the desired effectiveness and constraints set in the contour profilers, use the Monte Carlo Filtering component to fill the contour profiler white region with feasible solutions. In order to narrow down the reduced set of solutions further, we can select the most affordable solution, the solution with the least amount of variation, or the original optimized solution that was modified to satisfy the response limits.
 - f. Change the importance weights of the model outputs to represent different stakeholder views and repeat the previous steps until there are enough viable system variants.
 - g. Rerun the simulation models for each of the selected viable variants to confirm the surrogate metamodel results and generate the data needed to populate the measures in the value model.
 - h. Utilize the value model to evaluate the alternatives.

11.2. Technical Gap Bridges

Section 4 outlines the technical gaps within the ERS tradespace areas. This section reviews these gaps and discusses how our methodology addresses them.

Tradespace analytics. Current computational tools limit the exploration to only a few variables and produce static visualization diagrams. Our methodology leverages experimental space-filling designs suited for complex simulation models that allow the analyst to explore a high-dimensional problem. Traditional designs used for physical experiments were limited to only a few variables at a time. The new state-of-the-art nearly-orthogonal Latin hypercubes combined with nearly balanced designs allow system engineers to explore the interior of the experimental region while minimizing correlations between columns; in the past, experimenters had to choose a design that had only one of these properties. In addition, we can construct custom made space-filling designs for a very large dimension problem that have all experimental factor types (continuous, discrete, and categorical). These designs allow us to create surrogate metamodels that approximate the input/output behavior of the simulation models. We can use these metamodels to illuminate the key tradable variables with a dynamic dashboard created in JMP that combines statistical analytic artifacts with visualization features.

Decompose tradespace. The set of design driving variables is often unknown when designing a system. Additionally, current methods for conceptualizing design alternatives overlook or prematurely eliminate feasible designs. To address these technical gaps, we demonstrated an MBSE methodology that maps SysML elements to experimental design factors and incorporated the DOE analytical insights as derived requirements within our MBSE integrated system model. SysML allows us to express the conceptual model of a design with a wide variety of value properties that define the structural system characteristics. We can explore a several design alternatives by establishing an experimental region defined by the low and high setting of the value properties that define the system alternative. In order to identify the most significant design drivers we can use the polynomial linear regression metamodel with easily interpretable model coefficients that express the nature of the model input's impact on the response and the synergies that exist between them.

Tradespace search. Methods of searching the tradespace are slow due to the increase in system complexity, the growth of the number of potential tradable variables, and the need for high fidelity models for more accurate results. We can address these challenges by leveraging state-of-the-art space-filling experimental designs that efficiently span the entire design space region for a large number of variables. Because we can effectively explore the interior of the design space, we can analyze highly complicated response surface landscapes. In order to identify a narrow set of viable system alternatives we can use the collection of metamodels that we fit after executing the experiments to create a dynamic dashboard. The dashboard visualizes the multi-dimensional response surface using horizontal and vertical cross sections. We use the JMP™ 12 optimization heuristic to find solutions that balance multiple competing output responses and refine them when they are not feasible.

Evaluating tradespace results. Every system design problem has multiple stakeholders with different perspectives. In addition, attribute weighing schemes are often subjective making it difficult to build consensus. We address these gaps by integrating the philosophy of value focused thinking and the mathematics of multi-objective decision analysis to incorporate multiple stakeholder viewpoints. We create a functional objective value model that represents the composite perspective of multiple

stakeholders. We use the swing weight matrix to simultaneously incorporate subjective importance and objective levels of capability impacts.

11.3. Concluding Remarks

In today's complex environment, the DoD needs systems that are resilient to change and are effective across a wide variety of uncertain futures. The ERS architecture provides the means to apply a data driven approach using MBSE, incorporate previous design successes, integrate models, generate the data needed to visualize the tradespace, and create a shared digital thread of design decisions accessible to a community of users throughout the system lifecycle. Currently, there is a technical gap with regard to our ability to untangle the system design drivers when there is a high volume of multi-dimensional data. We propose an MBSE methodology that addresses this gap by leveraging the methods of experimental design in order to clearly identify tradable variables and narrow the search for viable system variants.

The general state-of-practice is to perform brute force simulation runs on a small set of baseline and excursions that do not effectively explore the system alternative design space. There is a lot of time, money, and resources devoted to building complicated simulation models and we do not use them to the maximum extent possible if we only compare a few excursions from the baseline. DOE provides a number of benefits that can assist in the design of a system. We can clearly identify the model inputs that affect the output responses, identify interactions that may exist between model inputs, uncover detailed insight into the model's behavior, examine the modeling assumption implications, frame the questions when we do not know what to ask, challenge or confirm expectation of directional model input effects and their relative importance, and uncover problems with simulation program logic.

In order to untangle the system design drivers across several different domain models, our methodology uses statistical metamodeling to approximate the simulations' behavior. We use these metamodels to capture insights as derived requirements that further refine a local property requirement and as satisfy relationships that identify value properties and structural blocks that satisfy emergent property requirements. In addition, we create a dynamic dashboard using the collection of metamodels to help visualize multi-dimensional response surfaces using horizontal and vertical cross sections. These cross sections allow us to clearly identify the tradable variables and find viable system variants that met the desired capabilities across multiple viewpoints and are physically feasible.

11.4. Future Research

The model integration challenge is a significant limitation that needs further research in order to effectively execute our proposed MBSE methodology. Translating system element value properties to simulation model inputs is a research area that is very specific to each domain and the types of models we use. The MBSE paradigm provides a means of managing the model integration requirements. Model fidelity differences create an even more challenging problem when we want to integrate a collection of models that represent one system design alternative. Addressing the model integration challenge will require additional modeling and practical application of system modeling efforts that are cross-coupled across different domains.

The statistical metamodels we created were fit using the mean of the response data. Therefore, the functional form of the metamodel only approximated the mean performance. The dashboard steps we proposed in Section 9 that identify viable variants are completing based on mean performance without regard to the alternative variability. We address this limitation by rerunning the selected viable system variant alternatives and using the vector of outcomes within our multiple-objective decision analysis model. Further research is necessary to apply our proposed methodology on the variance of the response data. Additionally, we can apply the methods of robust design in order to incorporate both the mean and variance using a loss function. Analyzing these loss functions allows us to directly incorporate noise variables along with our decision variables in order to find robust solutions that perform well across an uncertain environment. Finding robust solutions aligns very well with the ERS effort to identify resilient systems that are effective in a wide variety of uncertain environments. Rather than optimizing on the mean decision space, we can find robust solutions that provide a new class of viable system variants.

A final area of research is to create the physical architecture within the MBSE integrated model of the viable system variants identified using our dashboard. This task involves translating the model input settings of the identified solution back into the value properties of the system elements.

References

- Barton R. (1998). Simulation metamodels (D.J. Medeiros, E.F. Watson, J.S. Carson, and M.S. Manivannan, Eds.). Proc 1998 Winter Simul Conf Piscataway NJ IEEE. 1:167–74.
- BKCASE Editorial Board. (2015). The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 1.3.2 R.D.Adcock (EIC). Hoboken, NJ: The Trustees of the Stevens Institute of Technology. Accessed May 15, 2015. www.sebokwiki.org.
- Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. and Yergeau, F. (2008), Extensible Markup Language (XML) 1.0 (Fifth Edition) W3C Recommendation 26 November 2008.
- Clemen, R. T., & Reilly, T. (2001). Making hard decision with decision tools. *South-Western Cengage Learning, Mason, Ohio*.
- Delligatti L. (2013) SysML Distilled: A Brief Guide to the Systems Modeling Language. 1st edition. Upper Saddle River, NJ: Addison-Wesley Professional.
- Ewing Jr, P. L., Tarantino, W., & Parnell, G. S. (2006). Use of decision analysis in the army base realignment and closure (BRAC) 2005 military value analysis. *Decision Analysis*, 3(1), 33-49.
- Fisher RA. (1925) Statistical Methods for Research Workers. Biol Monogr Man Ser Edinb Scotl Oliver Boyd.
- Friedenthal S, Moore A, Steiner R. (2011) A Practical Guide to SysML, Second Edition: The Systems Modeling Language. 2nd ed. Morgan Kaufmann.
- Grayson, J., and S. Gardner. (2015). Building Better Models with JMP Pro. Cary, North Carolina: SAS Institute Inc.
- Hastie, T., Tibshirani, and J. Friedman. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed. New York: Springer-Verlag.
- INCOSE. (2015). Systems Engineering Handbook, A Guide for System Life Cycle Processes and Activities. 4th edition. San Diego, CA: Wiley.
- Keeney, R. L. (1992). Value-Focused Thinking: A Path to Creative Decision Making. Cambridge, MA: Harvard University Press.
- Kim H, Fried D, Menegay P, Soremekun G, Oster C. (2013) Application of Integrated Modeling and Analysis to Development of Complex Systems. *Conf Syst Eng Res*. 2013;16(0):98–107.
- Kirkwood, C. W. (1996). Strategic decision making. Wadsworth Publ. Co.
- Kleijnen JP., Sanchez SM, Lucas TW, Cioppa TM. (2005) A user's guide to the brave new world of designing simulation experiments. *Inf J Comput*;17(3):263–89.

Kleijnen, J. P. C. (2015). *Design and Analysis of Simulation Experiments*. 2nd ed. New York: Springer-Verlag.

Kuhn, M., and K. Johnson. (2013). *Applied Predictive Modeling*. New York: Springer-Verlag.

Hernandez, A. S., T. W. LUCAS, AND M. CARLYLE. (2012). Constructing nearly orthogonal Latin hypercubes for any nonsaturated run-variable combination. *ACM Transactions on Modeling and Computer Simulation*, 22, 4, No. 20.

Loh W-Y. (2011) Classification and regression trees. *Wiley Interdiscip Rev Data Min Knowl Discov*. Jan; 1(1):14–23.

MacCalman, A. D. (2013). *Flexible space-filling designs for complex system simulations*. Doctoral dissertation. Naval Postgraduate School, Monterey, CA. DTIC Document; 2013. Available from:

<http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA585718>

McIntosh, G.C., Galligan, D.P., Anderson, M.A., & Lauren, M.K. (2007). *MANA (Map Aware Non-Uniform Automata) Version 4 User's Manual*, Defense Technology Agency, New Zealand.

Montgomery DC. (2012). *Design and Analysis of Experiments*. 7th ed. Wiley.

Myers, R. H., D. C. Montgomery, and C. M. Anderson-Cook. (2009). *Response surface methodology: Process and product optimization using designed experiments*. 3rd ed. Wiley, January 14.

NATO Science and Technology Organization. (2014). *Data Farming in Support of NATO*. STO Technical Report TR-MSG-088.

NDIA Systems Engineering Division. (2011) *Final Report of the Model Based Engineering (MBE) Subcommittee*. Arlington, VA: NDIA.

Neches, R. (2011) *Engineered Resilient Systems (ERS), S&T Priority Description and Roadmap*.

Rao, C. R. (1945). Finite geometries and certain derived results in number theory. *Proceedings of the National Institute of Sciences of India*, 11, 136–149.

Sanchez, S.M. and H. Wan. (2009). [Better than a petaflop: The power of efficient experimental design](#). *Proceedings of the 2009 Winter Simulation Conference*, 60–74.

Sanchez, S. M. (2015). "Simulation experiments: Better Data, Not Just Big Data." In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz, W. K V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti. Piscataway, New Jersey: Institute of Electrical and Electronic Engineers, Inc.,

Sanchez, S. M., P. J. Sanchez, and H. Wan. (2014). "Simulation Experiments: Better Insights by Design." In *Proceedings of the 2014 Summer Simulation Conference*. San Diego, California: The Society for Modeling & Simulation International.

Sanchez, S. M., and H. Wan. (2012). "Work Smarter, Not Harder: A Tutorial on Designing and Conducting Simulation Experiments." In *Proceedings of the 2012 Winter Simulation Conference*, edited by C. Laroque, J. Himmelspace, R. Pasupathy, O. Rose, and A. M. Urmacher, 1929–1943. Piscataway, New Jersey: Institute of Electrical and Electronic Engineers, Inc

SAS Institute. 2015. JMP 12 Profilers. Cary, North Carolina: SAS Institute Inc.

Vieira, Jr., H., Sanchez, S., Kienitz, K. H., & Belderrain, M. C. N. (2011). Generating and improving orthogonal designs by using mixed integer programming. *European Journal of Operational Research*, 215(3), 629–638.

Appendix A: MANA Data Farming Manual



UNITED STATES MILITARY ACADEMY

ORCEN, Department of Systems Engineering

MANA Data Farming Manual

OPERATIONS RESEARCH CENTER, DSE

MANA Data Farming Manual

2LT Hyangshim Kwak
LTC Alexander MacCalman
February 2015

Table of Contents

Introduction	112
1.1 Required Programs	112
1.2 HPC Labs in the DSE.....	113
1.3 Overview	113
Design of Experiments	4
2.1 Scenario File	5
2.2 Analysis File	6
2.3 Translation File.....	10
2.4 Executable File	11
Generate Study File.....	123
3.1 Map Parameters.....	123
3.2 Navigating the Structured Tree	126
3.3 Designate the XPath	128
Execute Cluster Runs	23
4.1 Folder Hierarchy Orientation	23
4.2 Cluster Running Prerequisites.....	33
4.3 Launch OldMcData	36
4.4 Monitor Progress using Condor	37
Post-Process Output.....	38
5.1 Create Summary File	38
5.2 Post-Processing Setup.....	39
5.3 Concurrent Processing of Design Points.....	47
5.4 Sequential Processing of Design Points	48
5.5 Contents of Output Folder.....	49
Appendices.....	50
Appendix A: References	50
Appendix B: Glossary.....	51
Appendix C: OldMcData Folder Organization Layout	52
Appendix D: Installing XStudy	53
Appendix E: Downloading Condor	53
Appendix F: OldMcData Troubleshooting Options	54
Appendix G: Modifying the “condor_config” file.....	56
Appendix H: Points of Contact for Additional Help	57

Introduction

This manual provides instruction on how to setup, start, and manage data farming runs on the West Point server. It outlines the steps for creating a design of experiment, running simulations on a local distributed computer cluster, and post-processing data. The process was first used in a Tradespace Exploration project headed by LTC Alexander MacCalman at the Operations Research Center within the Department of Systems Engineering.

ICON KEY

[[key]]: Button on the keyboard

Command invoked in Command Prompt

1.1 Required Programs

The following instructions document the data farming procedures as was carried out in the Tradespace Exploration project. In this case, the analysts used the Map Aware Non-Uniform Automata as the simulation program and the “Design Creator” tool for the Design of Experiments portion; however, all programs can be substituted with other software that is better suited for the project at hand as long as it fulfills the required functionalities. The list below is a set of programs that have been implemented in the Tradespace Exploration project for data farming purposes. The ensuing instructions will be geared towards the following programs:

- Map Aware Non-Uniform Automata (MANA): Operational simulation
- “Design Creator” tool: Custom design creator or statistical package
- Microsoft Office Excel
- XStudy: A software mechanism with a graphical front-end to map model input parameter settings to columns in a design matrix
- Microsoft Visual Studio: an XML reading program
- OldMcData: A software that accepts the design matrix and specification study file as an input and produces the set of excursions needed to perform the simulation experiments

- Condor: A software management system that interfaces with the HPC and excursion files and distributes jobs across available computer resources while managing the transfer of files
- RStudio: A statistical software

1.2 HPC Labs in the DSE

Data cluster runs are implemented using High Performance Computer (HPC) cluster systems, which is a series of mapped computers that communicate with one another. Usually, HPCs are organized so that one master computer communicates with other remote machines. There are two labs in the Department of Systems Engineering that have HPC cluster capabilities. The table below lists the location of the labs and the associated master computer of the cluster.

Lab Name	Location	Computer ID
The Cave	Room 406a	08
Information Visualization (IV) Lab	Room 408	001

Table 1. HPC Labs in USMA, DSE

Chapters 2 and 3 can be completed with any logon credentials in your personal computer. However, you will need to log in to the master computer of a lab as a student in order to use the data farming and post-processing capabilities of Condor. When executing chapters 4 and 5, make sure you work in the lab.

To login to the master computer, enter the following credentials:

ID: cadet1se

Password: G0Systems123456

1.3 Overview

DATA

FARMING

METHODOLOGY

OGY

The Data Farming methodology can be broken down into the six steps listed below.

Step 1. Create a **Scenario file** (.xml) on MANA

Step 2. Create the analysis matrix using the “Design Creator” tool. Then, create the **Analysis** (.csv), **Translation** (.csv) and **Executable** (.csv) files on Microsoft Excel that designate the design variables and their values for the **design of experiment** (DOE).

Step 3. Use XStudy to map the design variables of the **Executable file** (.csv) and the elements of a MANA **Scenario file**. The output is an **Experiment Study file** (.xml).

- Step 4. Launch OldMcData to create individual scenario **excursion files** (.xml) for each design point. OldMcData also submits the runs to Condor, which distributes the simulation over a local distributed computer cluster.
- Step 5. Monitor the progress of the runs using Condor commands.
- Step 6. Post-process the output. Either invoke the standard MANA post-processor to create a **Summary file** (manaPP.bat) that concatenates all the MANA outputs, or invoke the manaminer post-processor that allows for personalized outputs.

Ensure that ALL file names (scenario, excursion, image, etc. files) do not contain spaces or special characters

Design of Experiments

Creating Analysis, Translation, and Executable files.

This section provides instruction on the Design of Experiments (DOE) portion of the project. The field of DOE allows us to efficiently explore a high-dimensional tradespace problem at the right system configuration settings within the design region in a feasible amount of time.

When using MANA models, there is a need to create three matrices, as manifested in the Analysis, Translation, and Executable files. Matrices are formed so that columns represent design variables and rows represent the input parameter specifications for each model experiment. Often times, MANA input parameters have multiple entries (see Figure 8). Therefore, lone design variables are not directly importable as MANA input parameters. Instead, the entries of the input parameter must be scaled by a single factor, which necessitates the creation of multiple files.

To create the executable matrix, there must be an intermediary file, known as the Translation file, to translate the analysis matrix into an execution matrix. The translation matrix scales all of the elements according to the desired amount based on the experimental design.

An overview of the four types of files is as follows:

- **Scenario File** (.xml): A MANA file that contains the baseline parameters for all variables that will be manipulated in the design of experiments. It contains all the details for the scenario.
- **Analysis File** (.csv): The experimental design the analysts will use to perform linear regression after appending the output data to it.
- **Translation File** (.csv): An intermediary file that translates the Analysis file into the Executable file.
- **Executable File** (.csv): A Microsoft Excel file where each column is a variable in the design of experiment and each row is a design point. The file is uploaded on XStudy to map the variables in the design and the elements of a MANA Scenario file.

2.1 Scenario File

Construct the baseline scenario model on MANA, which is the file that contains the baseline parameter settings for all the design variables that will get modified during the HPC cluster runs.

Step 1. Open MANA

- a. See “MANA (Map Aware Non-Uniform Automata) Version 4 User Manual” (May 2007) by McIntosh, Calligan, Anderson, and Lauren and “MANA-V (Map Aware Non-Uniform Automata- Vector) Supplementary Manual” (September 2009) by McIntosh for guidance on how to navigate and create a Scenario file (.xml).

Step 2. Establish baseline values for all parameters in the “Edit Squad” window, as shown in Figure 1. These values will later be manipulated in the Executable file (.csv).

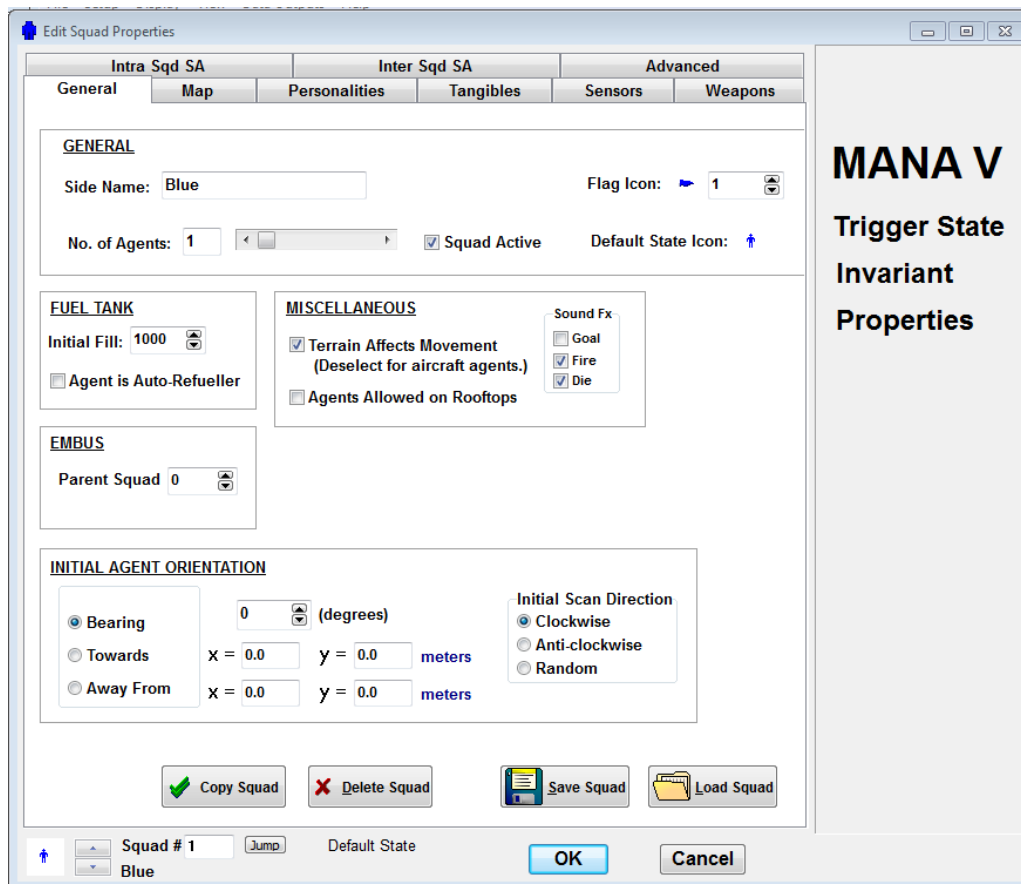


FIGURE 1. “Edit Squad Properties” window in MANA

Step 3. Check the desired output options in the “Data Outputs” tab. If you will be using the unique post-processor as instructed in Chapter 5, ensure that the following options are checked (see Figure 2):

- Record Casualty Location Data
- Record Agent State Data
- Record Multi-Contact Detections

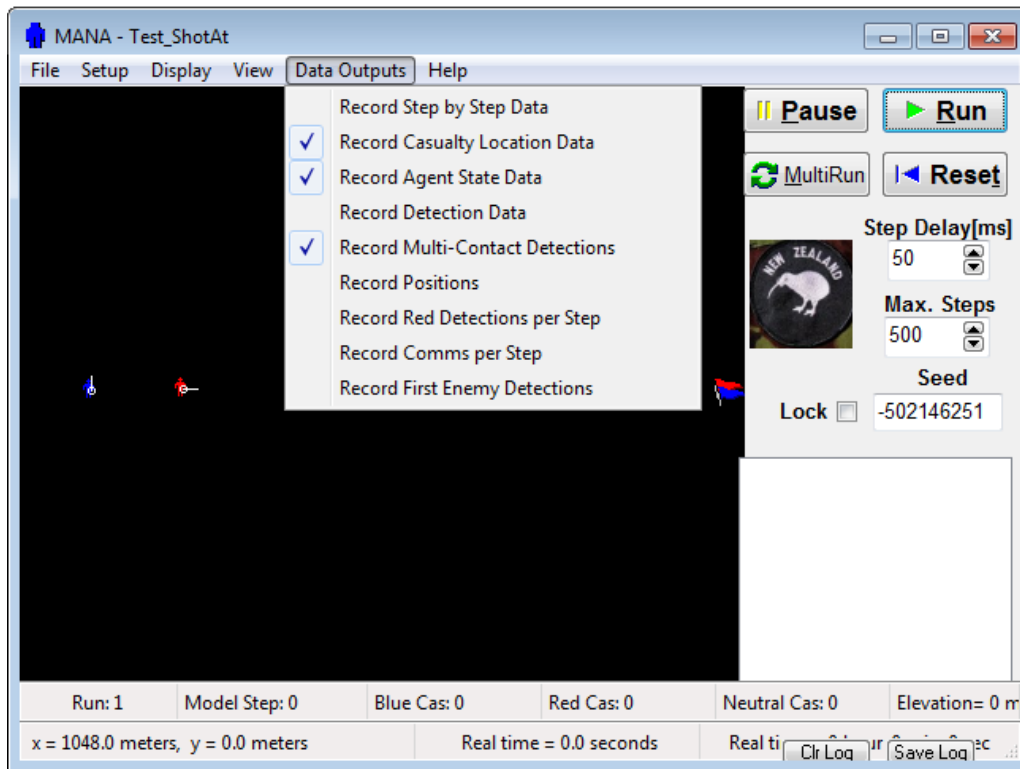


FIGURE 2. “Data Outputs” tab in MANA

Step 4. Save the file. Do not include spaces in the file name. In the manual, this MANA file will be referred to as the Scenario file (.xml).

2.2 Analysis File

The Analysis file (.csv) is the initial compilation of input parameters that lays out the individual MANA runs, otherwise referred to as design points, for the DOE. The Translation file (.csv) is derived from the Analysis file (.csv).

The “Design Creator” VBA file uses an algorithm that creates the experimental design. This manual will provide step by step instructions for creating an Analysis file through a use case. For more specific directions, reference the “Design Creator User Manual.”

Step 1. Open the “Design Creator” on Microsoft Excel and enable Macros.

Step 2. In the “Front End” tab, enter the number of factors and other required information in the blue cells. In the example scenario, there are five continuous factors. The number of levels is set to “39” in order to match the “Number of Design Points” parameter in the green-colored entry area:

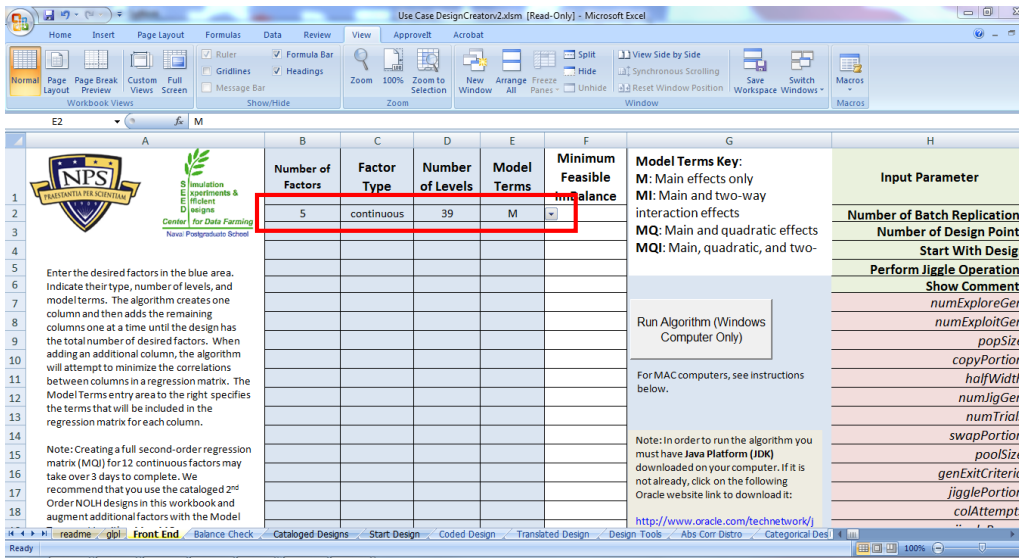


FIGURE 3. “Front End” tab of the “Design Creator” tool

	A	B	C	D	E	F
1	continuous	continuous	continuous	continuous	continuous	
2	39	39	39	39	39	
3	MQI	MQI	MQI	MQI	MQI	
4	x1	x2	x3	x4	x5	
5	20.5	22.5	22.5	26.49	17.43	
6	18.5	36.5	13.5	1	22.29	
7	29.89	30.39	4.5	36.5	31.68	
8	18.05	3.5	34.5	22.49	25.69	
9	9.5	1.5	5.46	6.5	22.5	
10	6.51	23.5	8.5	30.5	18	
11	5.87	29.21	3.31	12.5	37.03	
12	10.5	13.06	15.53	13.5	30.93	
13	1	20.5	24.66	1.58	15.76	
14	4.71	39	12.08	20.5	5.84	
15	36.54	34.87	25.62	28.5	5.2	
16	15.5	7.19	32.5	34.5	9.03	
17	22.04	21.66	24.13	24.5	28.5	
18	10.52	16.5	27.5	16.5	1	
19	16.58	18.5	1	22.5	20.88	
20	34.03	2.5	31.5	9.5	12.64	
21	28.5	1	8.13	30.5	20.32	
22	22.5	12.5	7.35	6.03	2.04	
23	36.25	9.92	15.5	32.5	6.94	
24	14.5	30.66	36.51	33.18	8.19	
25	30.75	11.5	11.5	10.53	3.5	

FIGURE 4. Output Design generated in Microsoft Excel

Step 3. After entering all of the factors, select the “Run Algorithm” button. A command window will open that shows the running processes. Once the algorithm is complete, an output design will be generated in Microsoft Excel. The total number of columns pertains to the total number of factors. Excluding the first four rows of headers, there should also be 39 rows of design points (see Figure 4).

Step 4. Next, copy the entire content of the output design in Microsoft Excel.

Step 5. Toggle back to the “Design Creator” and select the “Start Design” tab. If there is another design already inputted, select the “Clear Design Area” button to delete the previous work.

Step 6. Paste the contents of the design output in cell B1. (See Figure 3 for a screenshot of the “Start Design” tab). Ensure that the content is aligned with its respective label in column A.

Step 7. Press the “Create Translation Worksheet” button (see Figure 5). The “Design Creator” will then open the “Translated Design” tab, as shown in Figure 6.

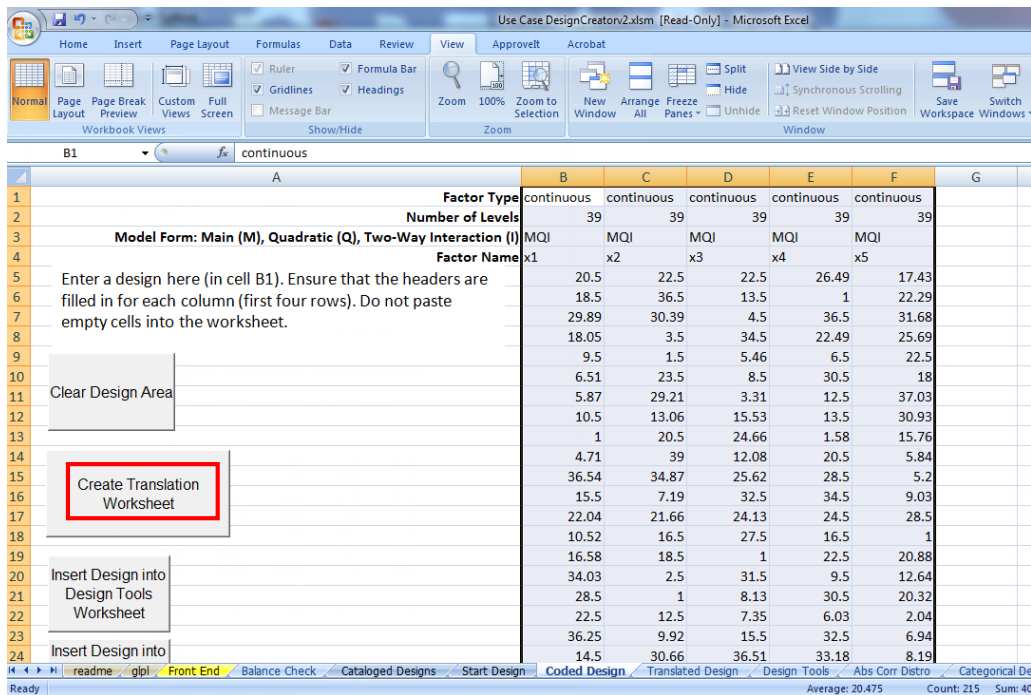
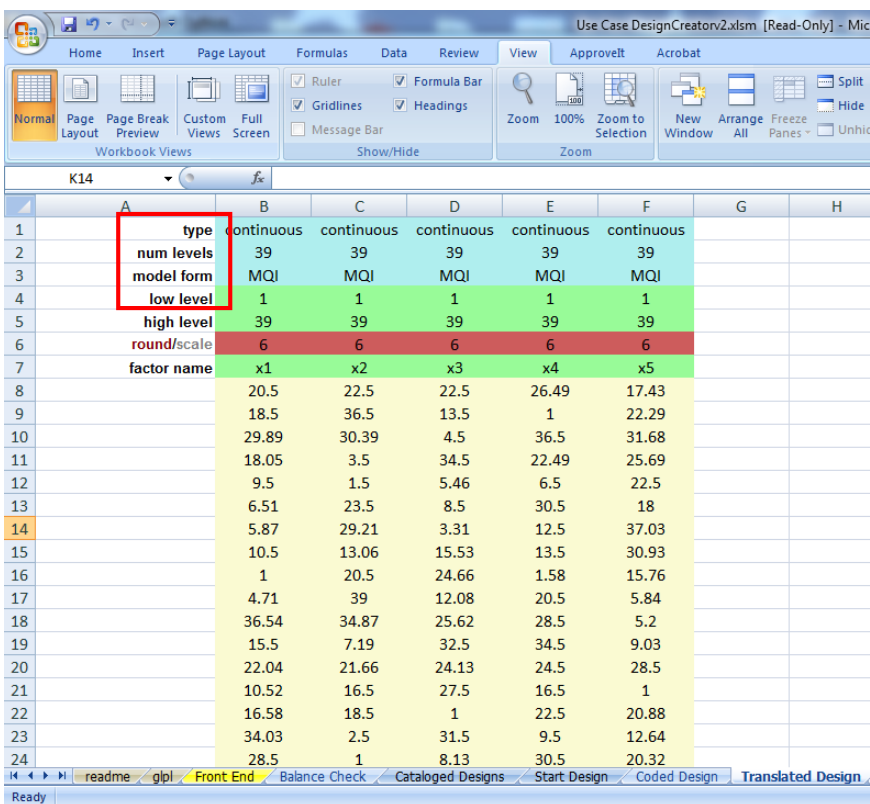


FIGURE 5. Press the “Create Translation Worksheet” button



Step 8. Update the “factor name” in row 7.

Step 9. Enter the low and high levels for each individual design variable in rows 4 and 5.

Step 10. Specify the desired number of digits in which to round the values in row 6. The “Design Creator” will automatically adjust the values accordingly. The image in Figure 7 shows the updated “Translated Design.”

Step 11.

Copy the “factor names” and design points (from cell B7 onwards).

Step 12.

Open a new Microsoft Excel file and select cell A1.

FIGURE 6. “Translated Design” tab within the Design Creator

	A	B	C	D	E	F	G	H
1	type	continuous	continuous	continuous	continuous	continuous		
2	num levels	39	39	39	39	39		
3	model form	MQI	MQI	MQI	MQI	MQI		
4	low level	6	1	60	2	0.9		
5	high level	15	5	100	7	2		
6	round/scale	6	0	6	6	6		
7	factor name	SDRSpeed	SDRKilHits	SDRStealth	SDRVisHeight	SDRDetRange		
8		10.618421	3	82.631579	5.353947	1.375605		
9		10.144737	5	73.157895	2	1.516289		
10		12.842368	4	63.684211	6.671053	1.788105		
11		10.038158	1	95.263158	4.827632	1.614711		
12		8.013158	1	64.694737	2.723684	1.522368		
13		7.305	3	67.894737	5.881579	1.392105		
14		7.153421	4	62.431579	3.513158	1.942974		
15		8.25	2	75.294737	3.644737	1.766395		
16		6	3	84.905263	2.076316	1.327263		
17		6.878684	5	71.663158	4.565789	1.040105		
18		14.417368	5	85.915789	5.618421	1.021579		
19		9.434211	2	93.157895	6.407895	1.132447		
20		10.983158	3	84.347368	5.092105	1.696053		
21		8.254737	3	87.894737	4.039474	0.9		
22		9.69	3	60	4.828947	1.475474		
23		13.822895	1	92.105263	3.118421	1.236947		
24		12.513158	1	67.505263	5.881579	1.459263		

FIGURE 8. “Sensor” tab in the “Edit Squad Properties” window on MANA

Step 13. Press [[Alt]], [[Ctrl]], and [[v]] simultaneously to special paste the design

points.

Step 14. Label the tab as “Analysis.” This is your Analysis file. Continue reading below for further orientation on the Analysis file and formatting suggestions. See Figure 9 for an example of a formatted Analysis file.

- ✓ The first row lists each design variable that will be altered in the **DOE**.
 - Each design variable pertains to a setting on the MANA input parameters that will be manipulated in the DOE
 - Ensure that variable headings do not contain spaces
 - Choose headings that are easily recognizable and unique to the setting
- ✓ Formatting:
 - Suggested background color: yellow
 - Bold the variables that require exactly one input.

- For example, bold all the variables in the use case except for “SDRDetRange”

FIGURE 9. Final “Analysis File” on Microsoft Excel

2.3 Translation File

Create a Translation file.

As seen in Figure 6, an input parameter may need a scaling factor to apply to multiple inputs. The **Translation file** (.csv) is an expanded version of the Analysis file that lays out each component that will be manipulated in the DOE. As such, the Translation file is necessary to create the **Executable file** (.csv).

Step 1. Copy and paste the variable headings and values in the second row of the newly created “Translation” tab.

Step 2. For the headings that are not bolded, note the number of inputs for the corresponding parameter in MANA and create additional headings as necessary.

- Bold the newly created headings and color the corresponding columns gray.
- Do not reformat the unbolted variable headings.

** Note: The bolded variable headings contain the values that will be directly inputted into MANA when running each simulation.*

Step 3. Fill in the values for the unbolted variable headings. Unlike the values for the bolded variable headings, the values for the unbolted headings are NOT the figures that will be inputted into MANA. Instead, they are the numbers that will be multiplied with the baseline value already determined in the scenario file (.xml)

- In row 1, write the baseline values as designated in the **Scenario file** (.xml) above each corresponding variable heading.
- Multiply the baseline values with the design points for each respective row to obtain the number which will be inputted.

<

2.4 Executable File

The Executable file (.csv) is derived from the Translation file (.csv) and will be uploaded in XStudy.

Step 1. Open the Translation file (.csv)

Step 2. Copy the bolded variable headings and the corresponding values. These are the design points that will be manipulated within MANA. These are the bolded variable headings in the Translation file. In other words, omit the columns that pertain to the scaling factors.

Step 3. Press [[Alt]], [[Ctrl]], and [[v]] simultaneously to special paste the values in a new sheet.

Step 4. Save the file as your Executable file (.csv) in csv format (see Figure 11).

	A	B	C	D	E	F	G	H	I	J
1	SDRSPEED	SDRKILLHIT	SDRSTEALT	SDRVISHE	SDRDR1	SDRDR2	SDRDR3	SDRDR4	SDRDR5	
2	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	
3	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	
4	12.84237	4	63.68421	6.671053	17.88105	536.4315	894.0525	1788.105	2682.158	
5	10.03816	1	95.26316	4.827632	16.14711	484.4133	807.3555	1614.711	2422.067	
6	8.013158	1	64.69474	2.723684	15.22368	456.7104	761.184	1522.368	2283.552	
7	7.305	3	67.89474	5.881579	13.92105	417.6315	696.0525	1392.105	2088.158	
8	7.153421	4	62.43158	3.513158	19.42974	582.8922	971.487	1942.974	2914.461	
9	8.25	2	75.29474	3.644737	17.66395	529.9185	883.1975	1766.395	2649.593	
10	6	3	84.90526	2.076316	13.27263	398.1789	663.6315	1327.263	1990.895	
11	6.878684	5	71.66316	4.565789	10.40105	312.0315	520.0525	1040.105	1560.158	
12	14.41737	5	85.91579	5.618421	10.21579	306.4737	510.7895	1021.579	1532.369	
13	9.434211	2	93.1579	6.407895	11.32447	339.7341	566.2235	1132.447	1698.671	
14	10.98316	3	84.34737	5.092105	16.96053	508.8159	848.0265	1696.053	2544.08	
15	8.254737	3	87.89474	4.039474	9	270	450	900	1350	
16	9.69	3	60	4.828947	14.75474	442.6422	737.737	1475.474	2213.211	
17	13.8229	1	92.10526	3.118421	12.36947	371.0841	618.4735	1236.947	1855.421	
18	12.51316	1	67.50526	5.881579	14.59263	437.7789	729.6315	1459.263	2188.895	
19	11.09211	2	66.68421	2.661842	9.30105	279.0315	465.0525	930.105	1395.158	
20	14.34868	2	75.26316	6.144737	10.71947	321.5841	535.9735	1071.947	1607.921	
21	9.197368	4	97.37895	6.234211	11.08132	332.4396	554.066	1108.132	1662.198	
22	13.04605	2	71.05263	3.253947	9.72368	291.7104	486.184	972.368	1458.552	
23	12.03474	4	76.31579	5.486842	12.10895	363.2685	605.4475	1210.895	1816.343	
24	10.5829	4	81.89474	2.984211	12.61842	378.5526	630.921	1261.842	1892.763	
25	8.723684	5	80.41053	7	15.60289	468.0867	780.1445	1560.289	2340.434	

FIGURE 11. “Executable File” in Microsoft Excel

Generate Study File

Use XStudy to generate an Experiment Study file that specifies the model input parameters within MANA to change

3.1 Map Parameters

XStudy is a mechanism that finds, selects, and modifies the model input parameters of the base case scenario to create a set of excursions, one for each experiment or row in the design matrix. We will use the software to map the input parameters of a MANA Scenario file to the columns in the Executable file.

Ensure that the xstudy.bat file is saved in your C drive.

Step 1. Navigate to the C directory and locate the XStudy folder:

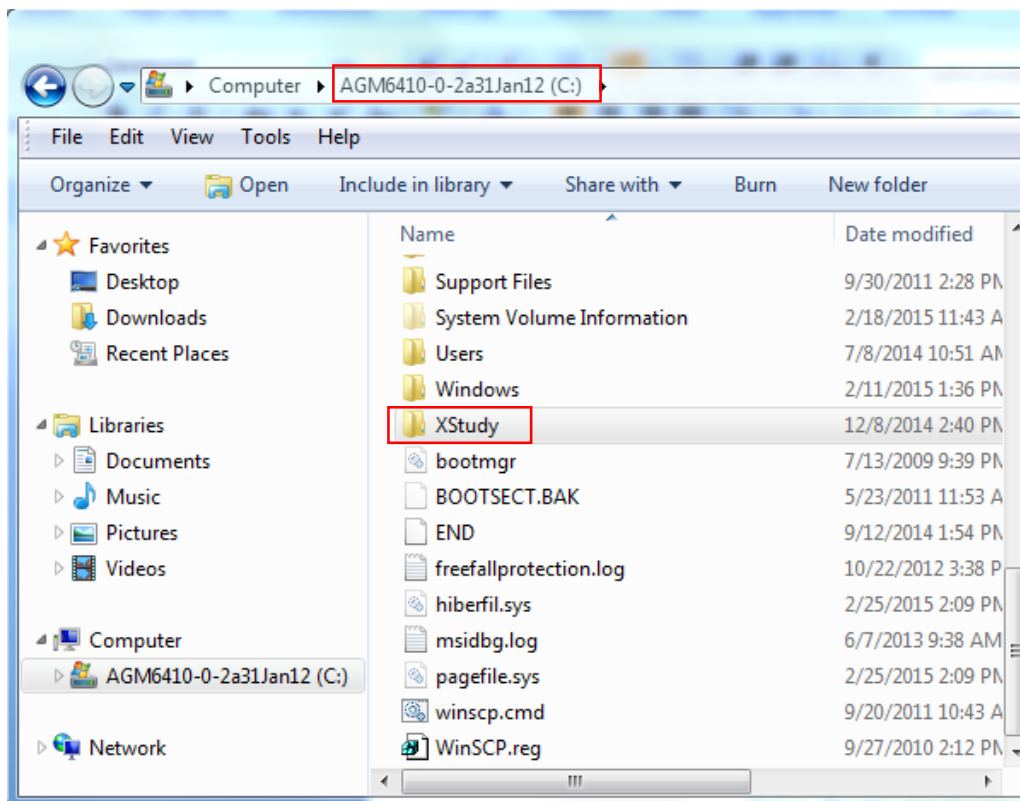


FIGURE 12. “XStudy” folder

Step 2. Open the XStudy folder (Path: C drive → “XStudy” folder) and locate the xstudy.bat file (see Figure 13).

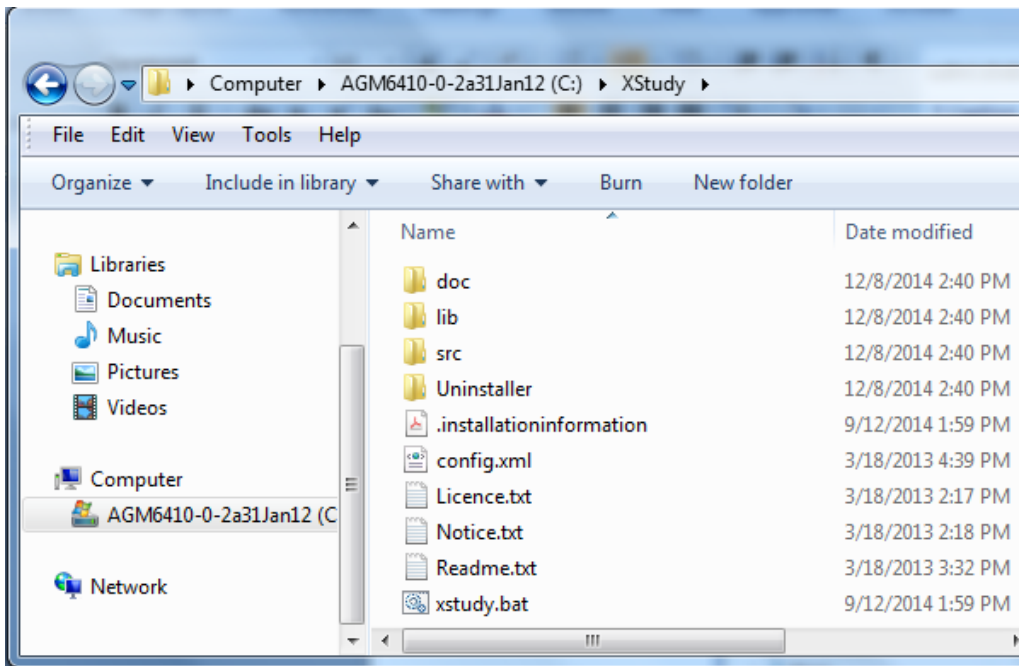
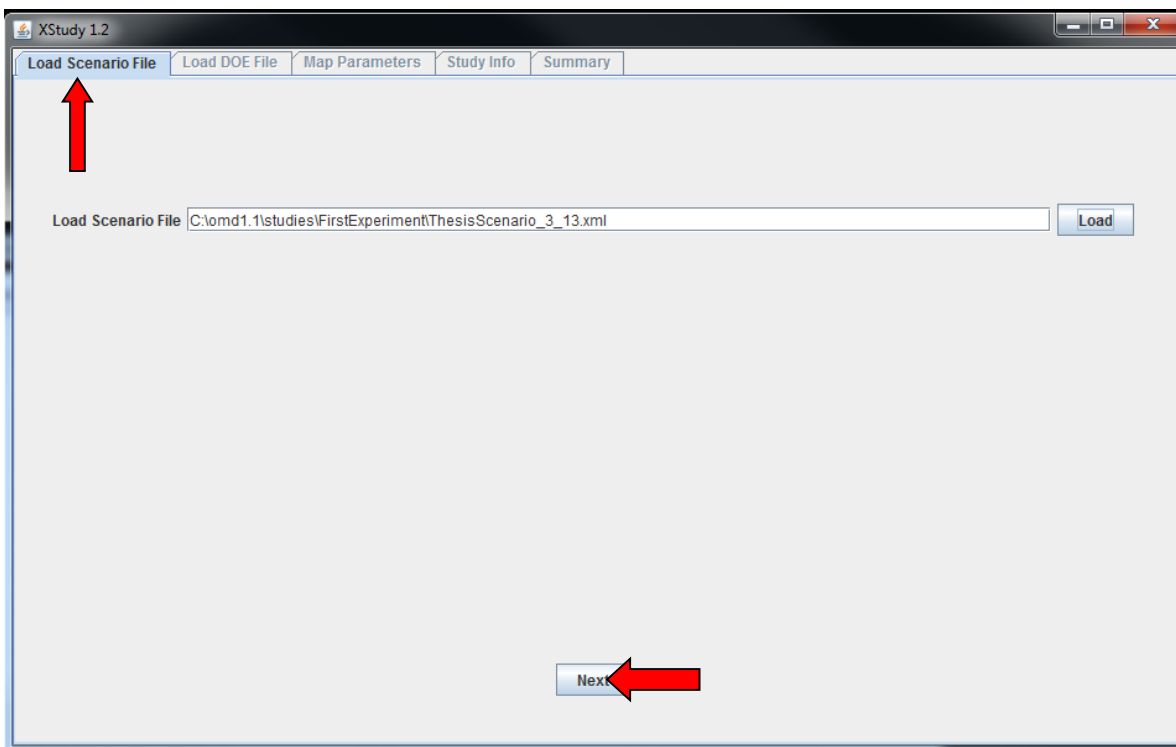


FIGURE 13. “xstudy.bat” file within the “XStudy” folder

Step 3. Double click the xstudy.bat file then load the MANA Scenario file (.xml) by navigating to its file location.



Step 4. In the “Load Scenario File” tab, select “Next” as depicted in Figure 14.

Step 5. In the “Load DOE File” tab, load the Executable file (.csv) by clicking on “browse” then navigating to its file location. Check to see that the proper file name has appeared as the “DOE File Name.” Then select “Load DOE”:

FIGURE 14. “Load Scenario File” tab in XStudy

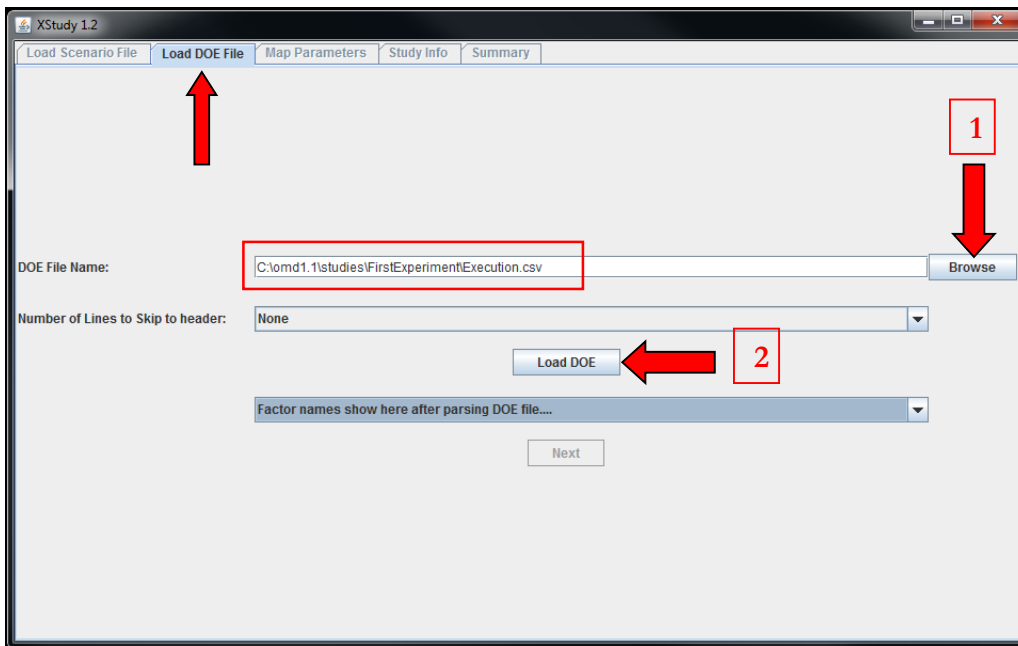


FIGURE 15. “Load DOE File” tab in XStudy

Step 6. After loading the Executable file (.csv), confirm the correct number of lines to skip. Then check to ensure the column headings are properly identified by using the drop-down arrow:

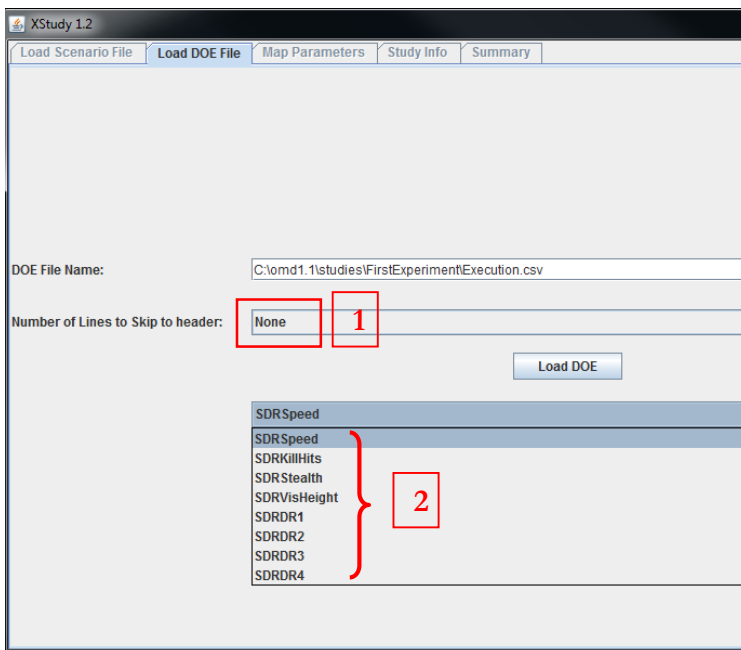


FIGURE 16. Checking the Headings in the “Load DOE File” tab

Step 7. Select “Next.” You will be taken to the “Map Parameters” tab as shown below. The structured tree on the left corresponds to the MANA Scenario file (.xml) input parameters. The items under “Parameter Groups” are the column headings of the Executable file (.csv).

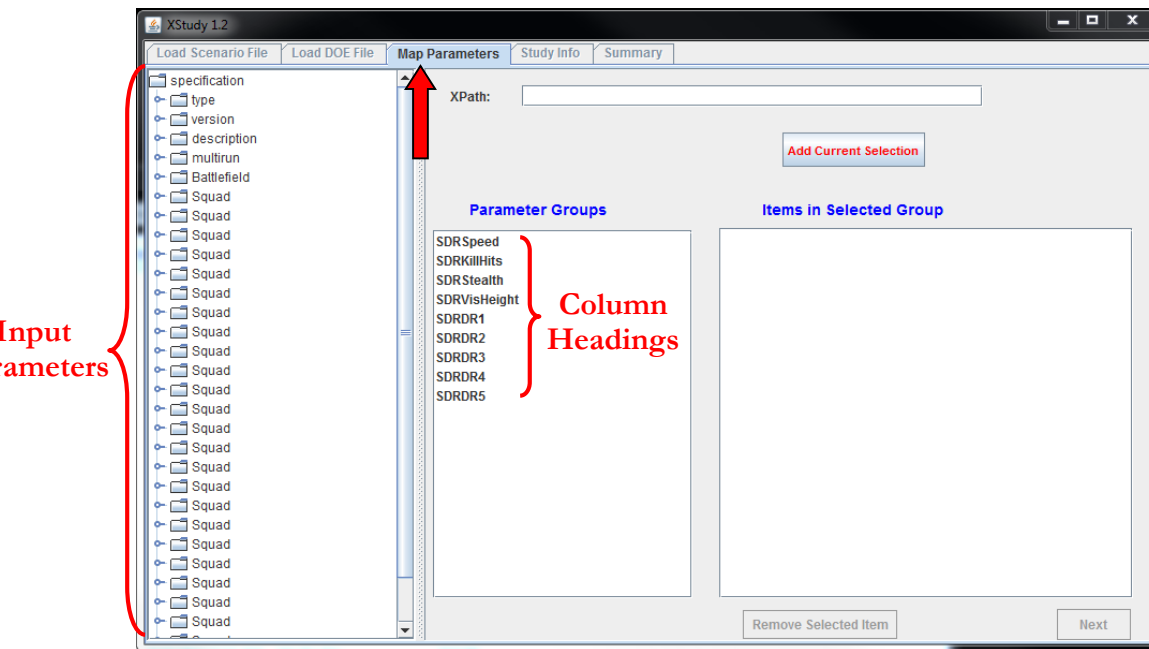
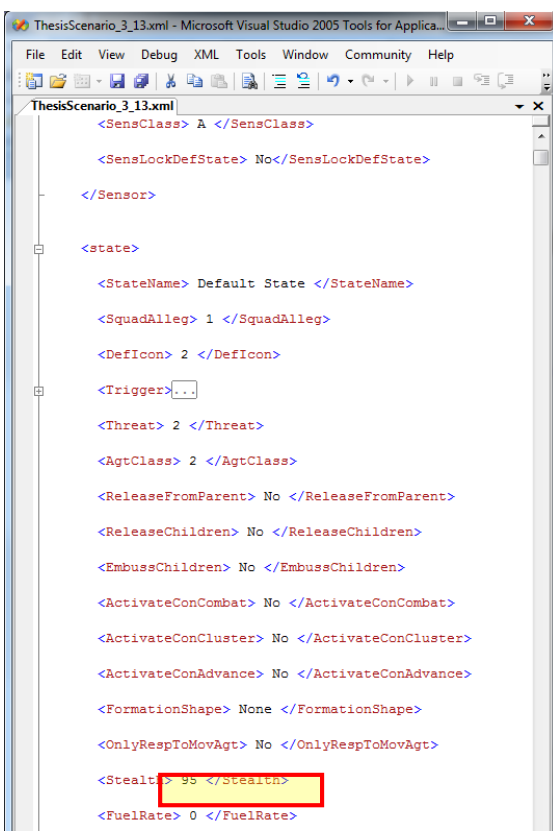


FIGURE 17. “Map Parameters” tab in XStudy

In this tab of XStudy, you will need to link all the column headings of the executable file (.csv) to its respective input parameter in the MANA scenario file (.xml). Before we describe the details on mapping the two items, we will learn how to locate the input parameters within the structured tree in the next section, entitled “Navigating the Structured Tree.”

3.2 Navigating the Structured Tree



It may be difficult to identify the input parameter that you need to reference in the “Map Parameters” tab of XStudy, especially if you have an extensive and complex Scenario file (.xml). In this section, we will demonstrate two methods on how to locate the specific parameter more effectively through an example use case.

Use case: We want to locate the “Stealth” MANA parameter that pertains to the heading “SDR Stealth” for a friendly Bradley Infantry squad.

Method 1: Browse the XML text file of the Scenario file.

Step 1. Open up the MANA Scenario file (.xml) on any XML viewer.

Step 2. Locate the input parameter in the file using the search function by pressing [[Control]] + [[F]]. The key aspect to keep in mind is if whether

the input parameter is a state-dependent or state-invariant parameter. If the property is the former, make sure to look for it in the appropriate <state> section. Some key elements to record are the squad number and state name which contain the parameter. It may be easier to print out a hard copy or screen capture selected parts of the scenario file (.xml) to manually number the ranges, which XStudy does not label.

By viewing the XML script of the Scenario file (.xml), we can see that the “Stealth” MANA parameter lies within the “Default” state with a value of 95 (see Figure 18).

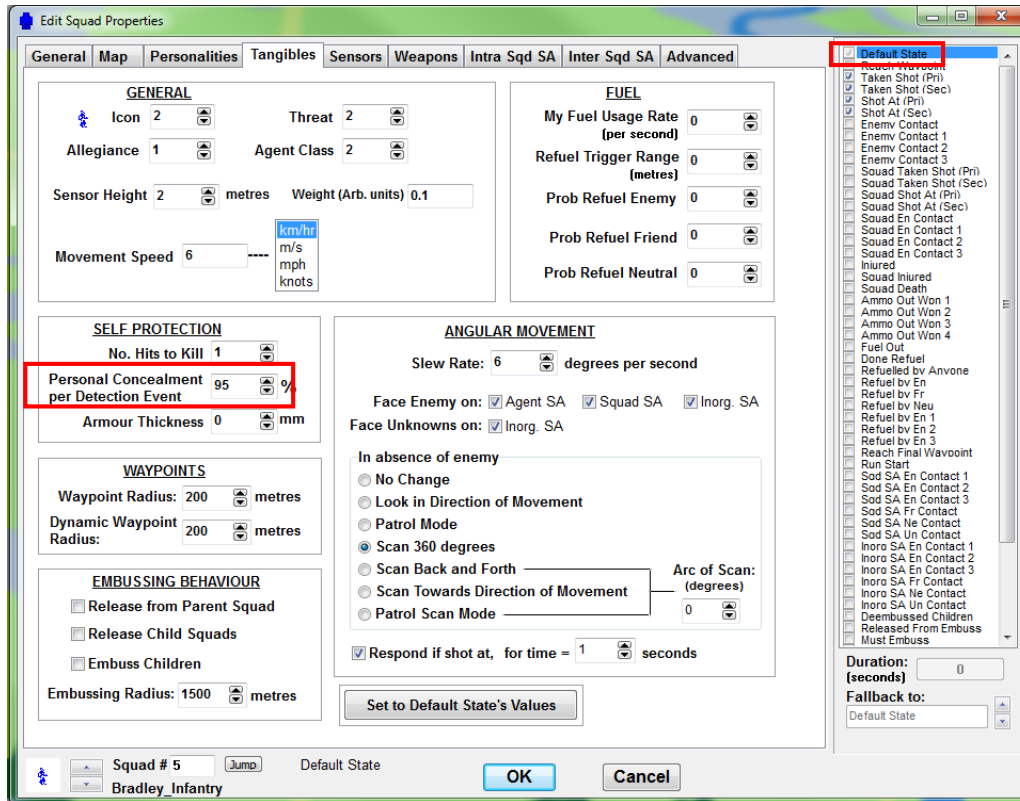
- Step 3. Navigate back to the XStudy GUI. Count the number of squads to find the desired squad name. Then locate the state name. Finally, find the desired input parameter within the structured tree.
- Step 4. Check to make sure the value of the input parameter is consistent with what is inputted on MANA to ensure that you have selected the correct one.

Method 2: Locate the input parameter in the MANA dashboard.

Another method of narrowing the location of the input parameter of interest is to look within your MANA dashboard.

- Step 1. Open the MANA dashboard
- Step 2. Toggle to the location of the input parameter in MANA and notes the states to which the parameter is applied. For example, the “Stealth” parameter, designated as “Personal Concealment per Detection Event” in MANA is located in the “Tangibles” tab and applies to the “Default,” “Taken Shot,” and “Shot At” states. It also is consistent with a value of 95 (see Figure 19)
- Step 3. Navigate back to the XStudy GUI and find the corresponding location of the MANA input parameter by choosing the appropriate squad number and state.

Step 4. Check to make sure the value of the input parameter is consistent with what is inputted on MANA to ensure that you have selected the correct one. In this case, “Stealth” was given a value of 95 in MANA. The selected input parameter entitled “Stealth” in XStudy has the same value. This is one more check to ensure correct mapping.



3.3 Designate the XPath

Now that we are familiar with how to navigate the structured tree in XStudy, we will proceed with instructions on how to map the MANA input parameters with

the column headings of the Executable file (.csv).

FIGURE 19. “Tangibles” tab in “Edit Squad Properties” Window of MANA (Method 2)

Step 1. Select a column heading in the “Parameter Groups” section:

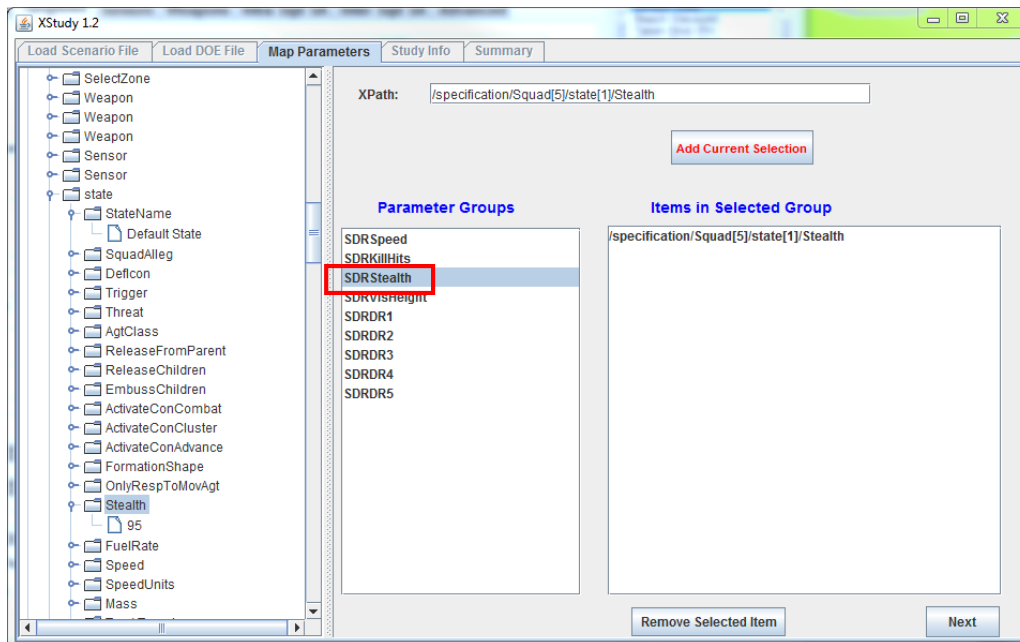


FIGURE 20. “Map Parameters” tab in XStudy

Step 2. Navigate to and select the corresponding MANA input parameter in the structured tree. (Refer to the previous section for guidance on how to navigate the structured tree) Once selected, the path to the input parameter will appear as the “XPath,” as shown below:

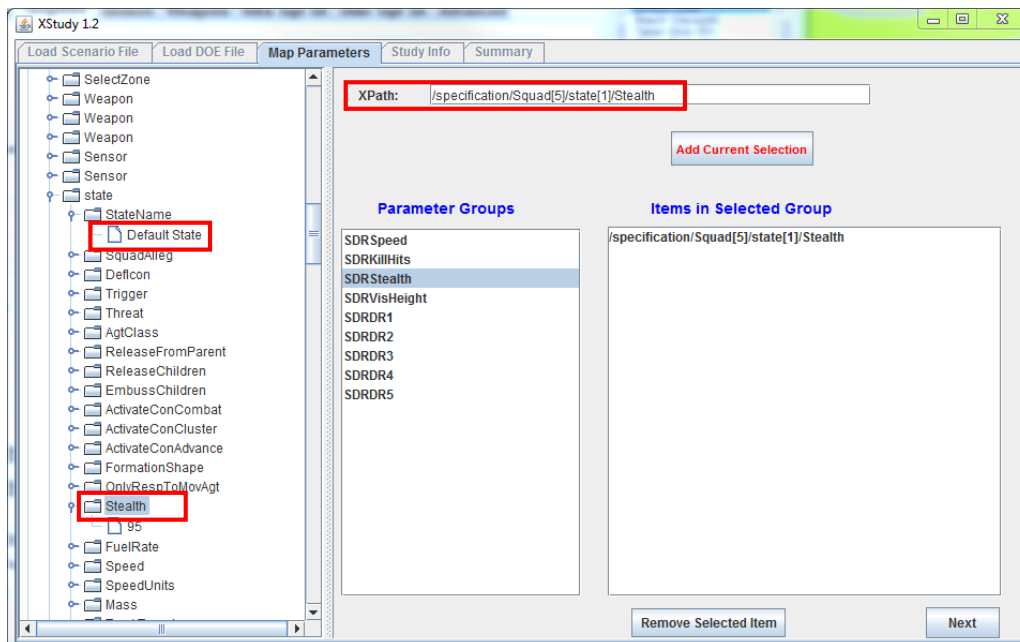


FIGURE 21. Selecting the XPath

Step 3. Click on “Add Current Selection” and check that the XPath has appeared in the “Items in Selected Group” block and the appropriate input parameter is highlighted. Populate the XPath(s) for all the input parameters:

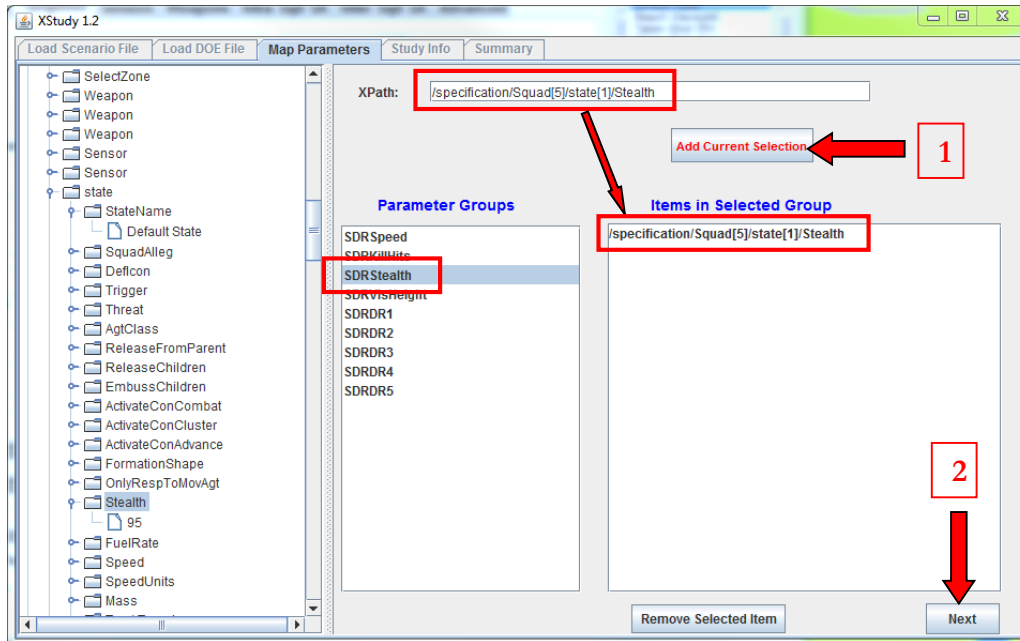


FIGURE 22. Populating XPaths

Step 4. For the column headings that have multiple inputs, map each individual input. Once all the input parameters are mapped, select “Next” to proceed to the “Study Info” tab. See the following screenshot depicting the XStudy GUI:

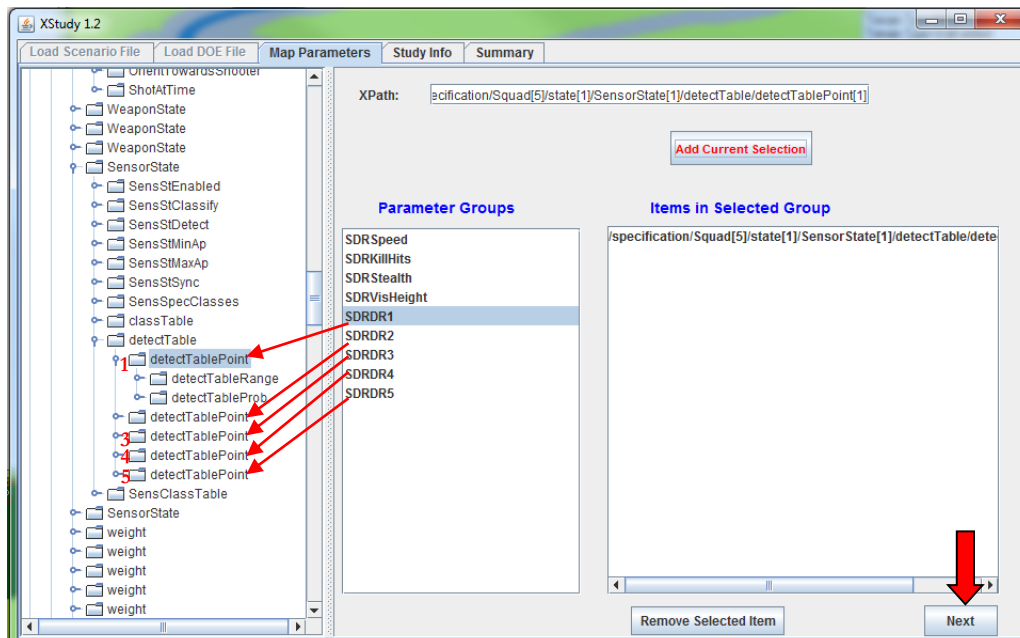


FIGURE 23. Mapping an Parameter with Multiple Inputs

Step 5. In the “Study Info” tab, enter the “user” and “study” information. In addition, edit the model information so that it matches your version of MANA. Finally, select “Next” to proceed to the “Summary” tab. See the snapshot of the XStudy GUI below:

USER INFO:

Name:

Phone:

Email:

STUDY INFO

Study Name:

Description:

MODEL INFO:

Name:

Major Ver:

Minor Ver:

MODEL RUN INFO

Initial Seed:

of Replicates:

Next

Edit the model information so that it matches your version of MANA. It can also be changed in an XML viewer (see ???)

FIGURE 24. “Study Info” tab in XStudy

Step 6. In the “Summary” tab, look over the XPaths to ensure proper mapping. The visual below shows one XPath per input parameter but there can be multiple XPaths. Finally, click “Make Study .xml File” to generate the Experiment Study file (.xml):

NAME: SDRDR1

XPATHS:

NAME: SDRDR2

XPATHS:

NAME: SDRDR3

XPATHS:

NAME: SDRDR4

XPATHS:

NAME: SDRDR5

XPATHS:

Make Study.xml file

Exit

FIGURE 25. “Summary” tab in XStudy

**Note: There can be multiple linkages for each heading. I.e. The heading “SDRStealth” can pertain to the “Stealth” parameter for multiple squads and will therefore be linked with multiple paths.*

Step 7. Finally, exit XStudy by selecting “Exit,” as shown in the image below:

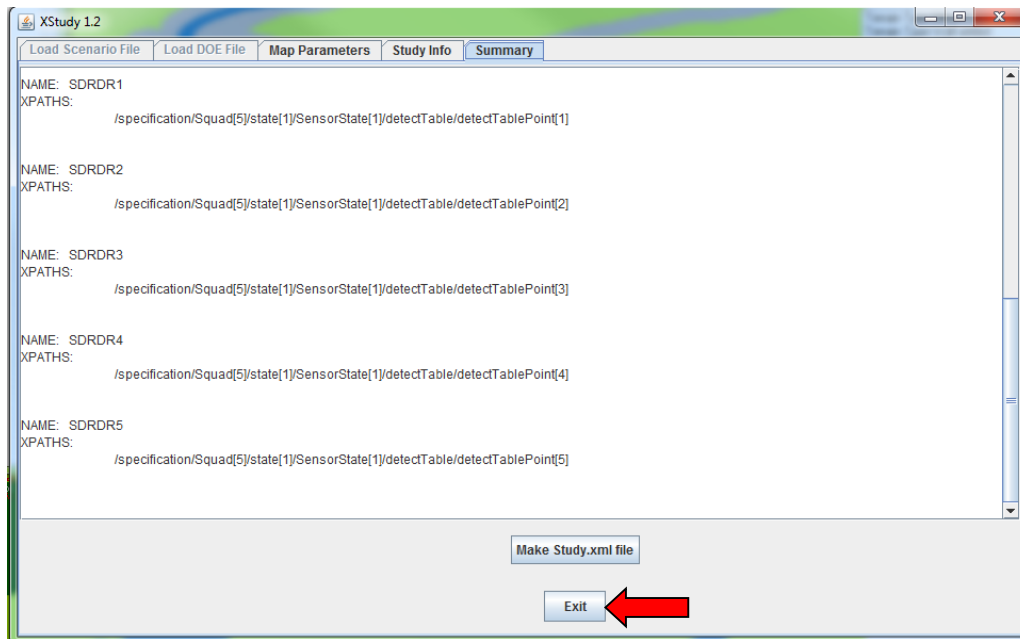


FIGURE 26. Exiting out of XStudy

Execute HPC Simulation Runs

*Launch OldMcData to create individual scenario **excursion files** (.xml) for each **design point**. OldMcData also submits the runs to Condor, which distributes the tasks amongst the computers in the cluster.*

OldMcData is a mechanism that accepts the Executable file and Experiment Study file as an input and produces the set of excursions needed to perform the simulation experiments. It creates individual excursion files with the new model input parameters set to the values specified in the design matrix. For each experiment or design point in the design matrix, there should be one excursion file.

Condor is a job queuing mechanism that enforces a scheduling policy and priority scheme while monitoring the computer resources that will complete the jobs. A job is one or more runs of the simulation model excursion. Condor interfaces with the HPC and excursion files and distributes jobs across available computer resources while managing the transfer of files.

OldMcData and Condor in conjunction allow for the HPC cluster runs of a simulation.

4.1 Folder Hierarchy Orientation

For the data farming procedure to properly execute, certain files must be saved in specific folders within the C drive, otherwise known as the C directory. As such, it is crucial to gain a firm understanding of the folder hierarchy for navigational and organizational purposes. In the following steps, we will orient ourselves to the layout of the omd1.1 folder within the C directory and create obsolete folders.

First, log in to the master computer of a lab.

Step 1. For West Point analysts, log in to the master computer of a lab with the following credentials:

ID: cadet1se

Password: G0Systems123456

West Point analysts must log in using the Student ID in the computer labs.

We must use the student ID to bypass the Condor credentials constraint. Condor requires new users to store their credentials in the system for security purposes; however, the procedure necessitates administrative privileges. Unlike individual accounts, the

student ID account is already recognized by Condor.

Become familiar with the folder hierarchy.

Step 2. Open up “My Computer” and locate the C directory, as shown in the image below:

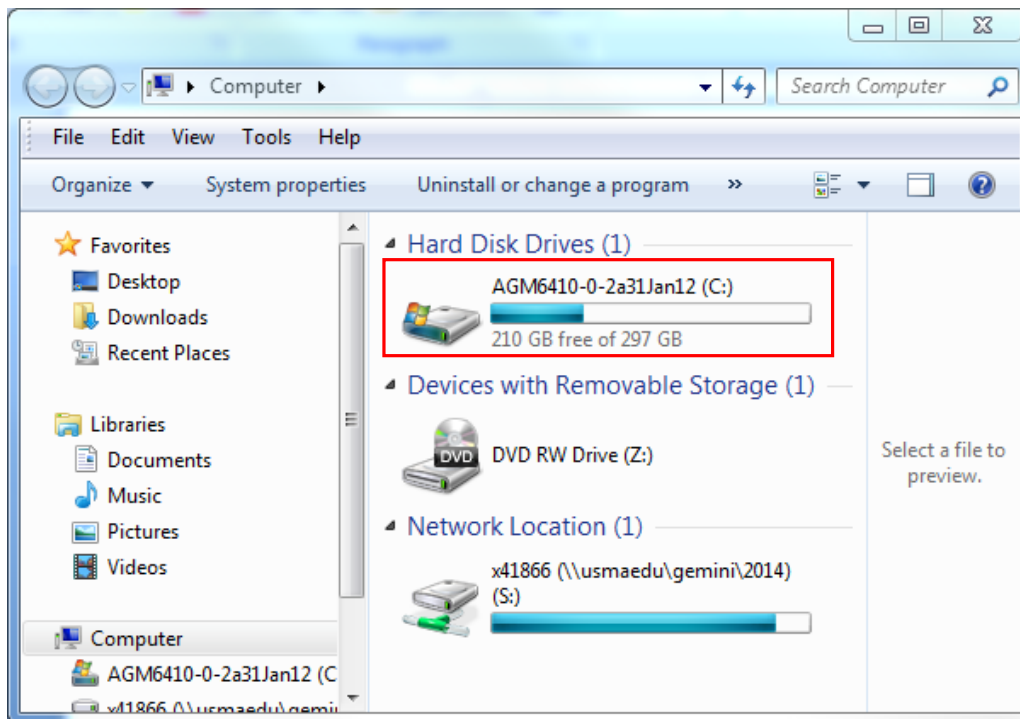


FIGURE 27. Locate the C Directory in “My Computer”

Step 3. Open up the C directory and locate the “omd1.1” folder, as shown in Figure 28.

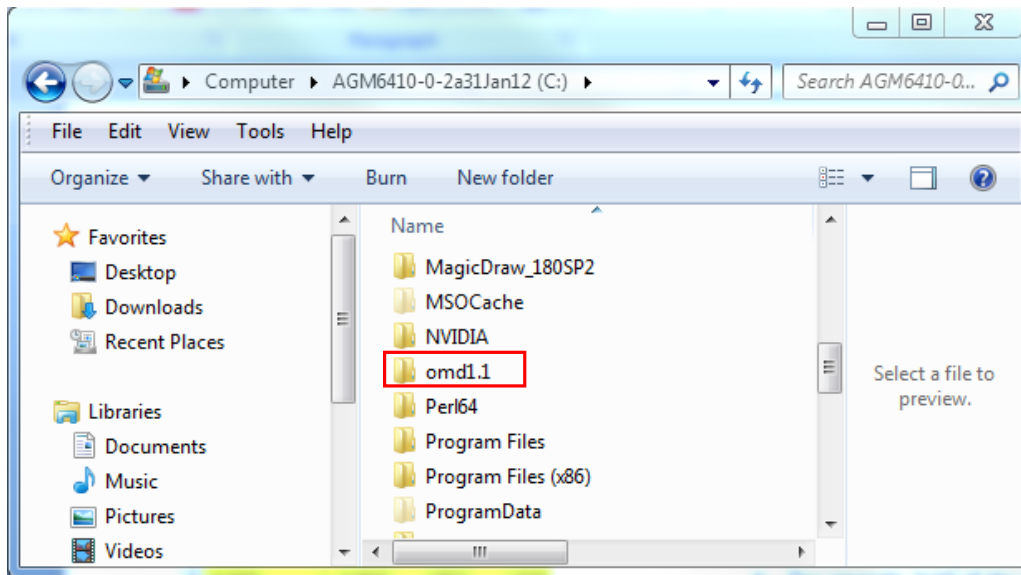


FIGURE 28. Locate the “omd1.1” folder in the C drive

The “omd1.1” folder pertains to the OldMcData program and is the central folder that contains all the folders of interest. An overview of the folder layout is shown below:

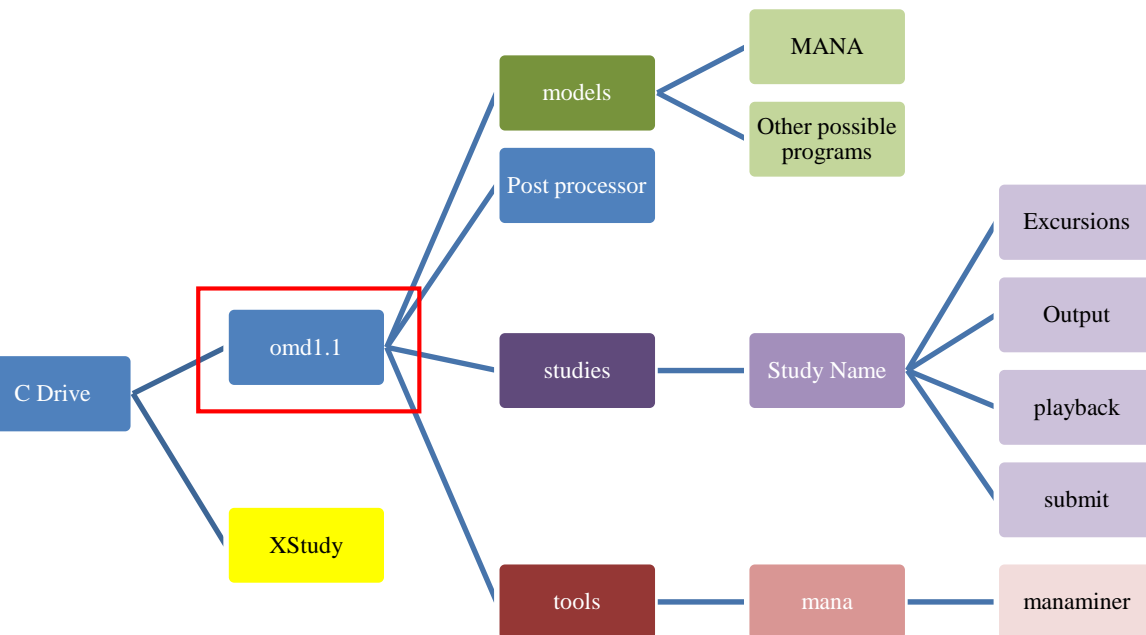


Chart 1. “omd1.1” folder in Folder Hierarchy

Step 4. Open the “omd1.1” folder and locate the “models” folder as seen in the image below:

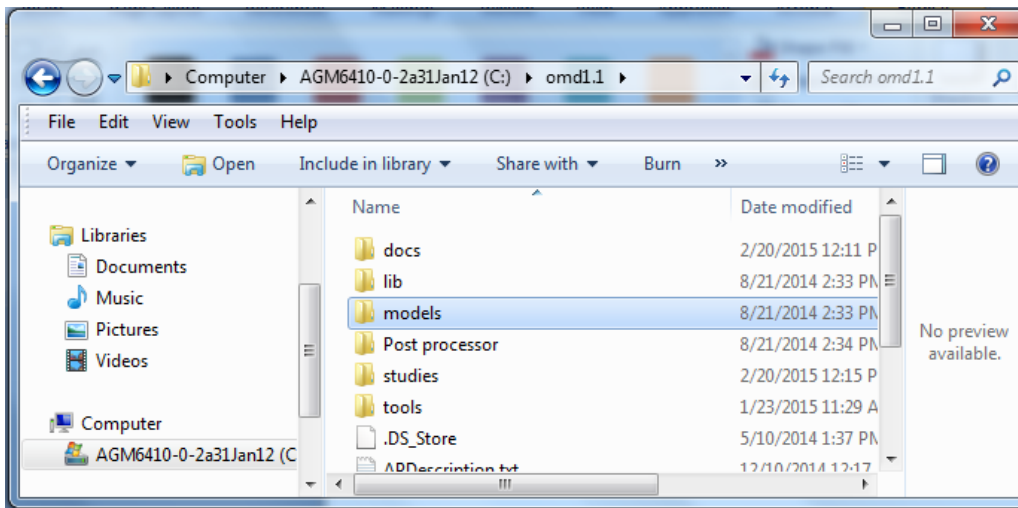


FIGURE 29. Locate the “models” folder in the “omd1.1” folder

Step 5. Create a new “MANA” folder within the “models” folder. Each simulation program requires a unique folder.

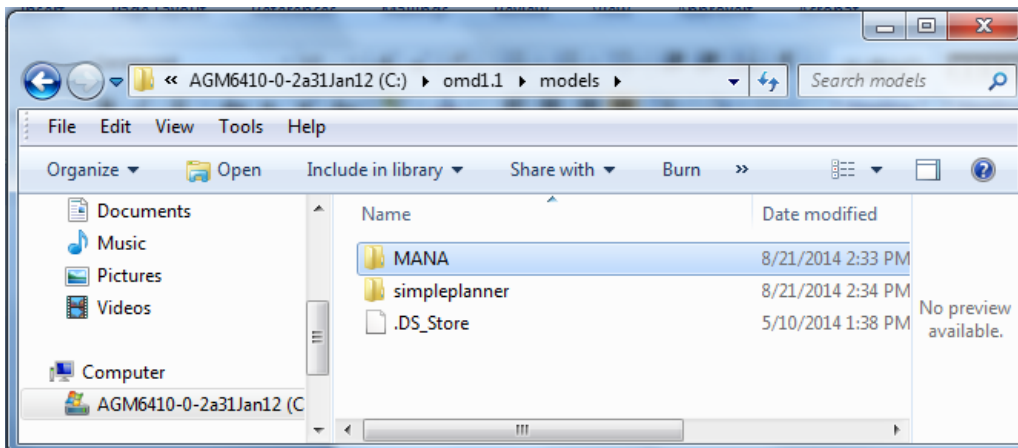


FIGURE 30. Locate the “MANA” folder in the “models” folder

Chart 2 highlights the newly created “MANA” folder in the folder hierarchy.

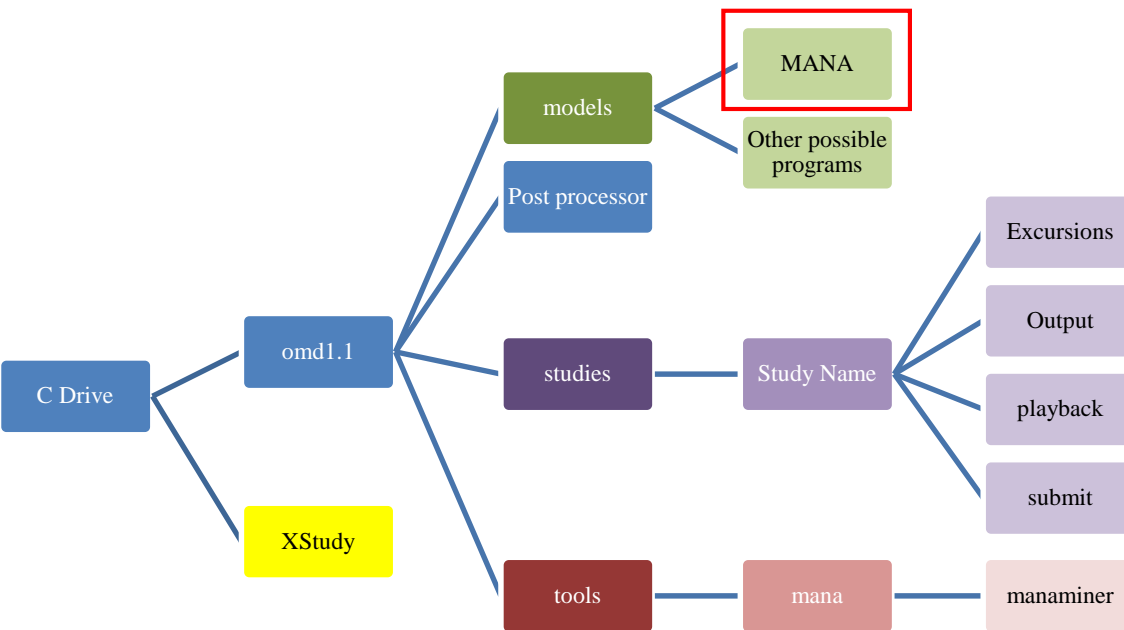


Chart 2. “MANA” folder in Folder Hierarchy

Create a new bat. file and save it to the “MANA” folder.

Step 6. Open up Notepad. Type in:

MANAC.exe -f%1 -n%2 -m%3 -e%4 (See Figure 31)

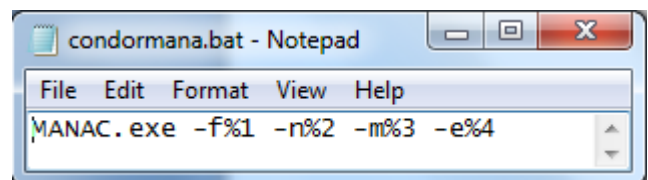
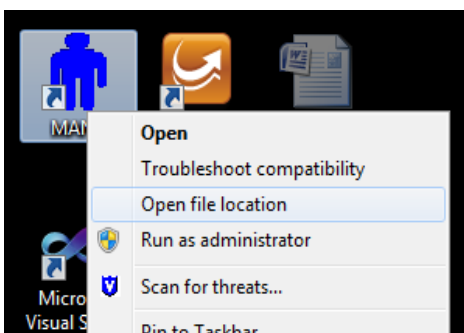


FIGURE 31. “condormanana.bat” file in Notepad



Step 7. Save the file as “condormanana.bat” in the newly created “MANA” folder.

Copy two files into the “MANA” folder: “mana.reg” and “MANAC.exe”

Step 8. Right click on the “MANA.exe” shortcut on your desktop and select “Open file location” as seen in Figure 32.

Step 9. Copy and paste the “mana.reg” and “MANAC.exe” files

FIGURE 32. Open MANA file location

from their original location into the “MANA” folder (Path: C drive → “omd1.1” folder → “models” folder → “MANA” folder). The “MANA” folder should now contain three files, as shown in Figure 33.

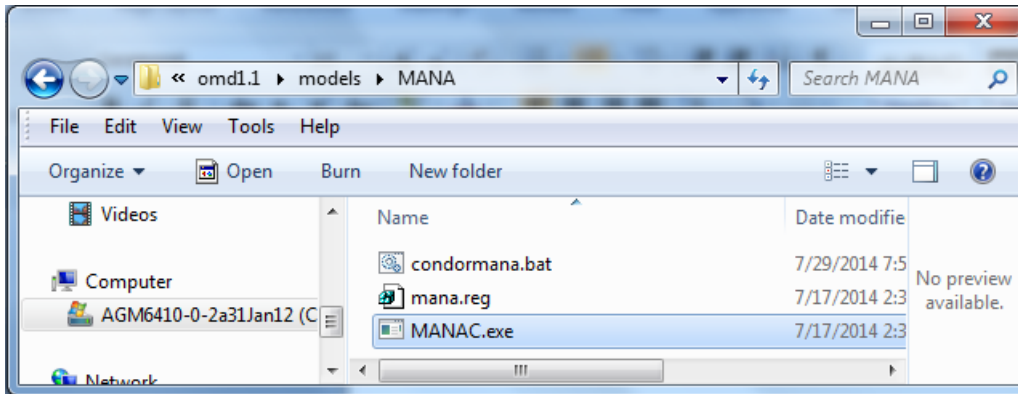


FIGURE 33. Contents of “MANA” folder.

SETTING UP THE MANAMINER POST- PROCESSOR

Create a folder that contains the files necessary to post-processes the output.

First, check to see if the manaminer folder already exists in the C drive. The path to the folder is: C drive → “omd1.1” folder → “tools” folder → “mana” folder → “manaminer” folder. If the “manaminer” folder is absent, follow the directions in this section. Otherwise, skip to “Create the ‘Study Name’ Folder” section.

Step 1. Navigate back to the “omd1.1” folder. Create a new “tools” folder, as shown below:

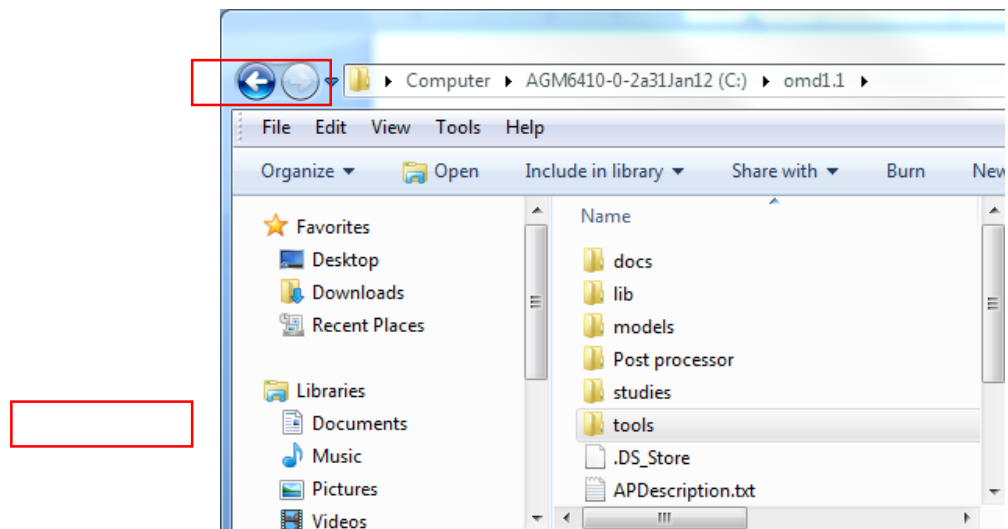


FIGURE 34. Locate “tools” folder within “omd1.1” folder

Step 2. Open the “tools” folder and create a “mana” folder within it (see Figure 35).

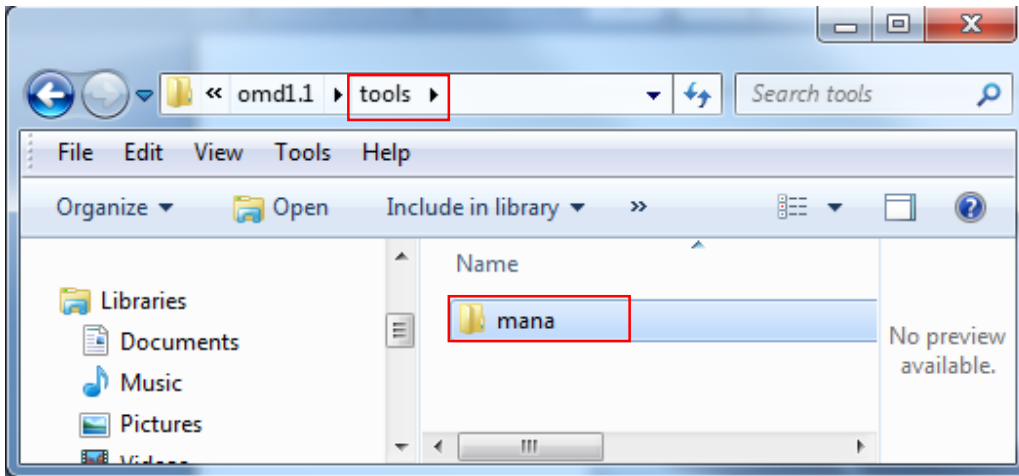


FIGURE 35. Create a “mana” folder within the “tools” folder

The chart below highlights the newly created “mana” folder in the folder hierarchy:

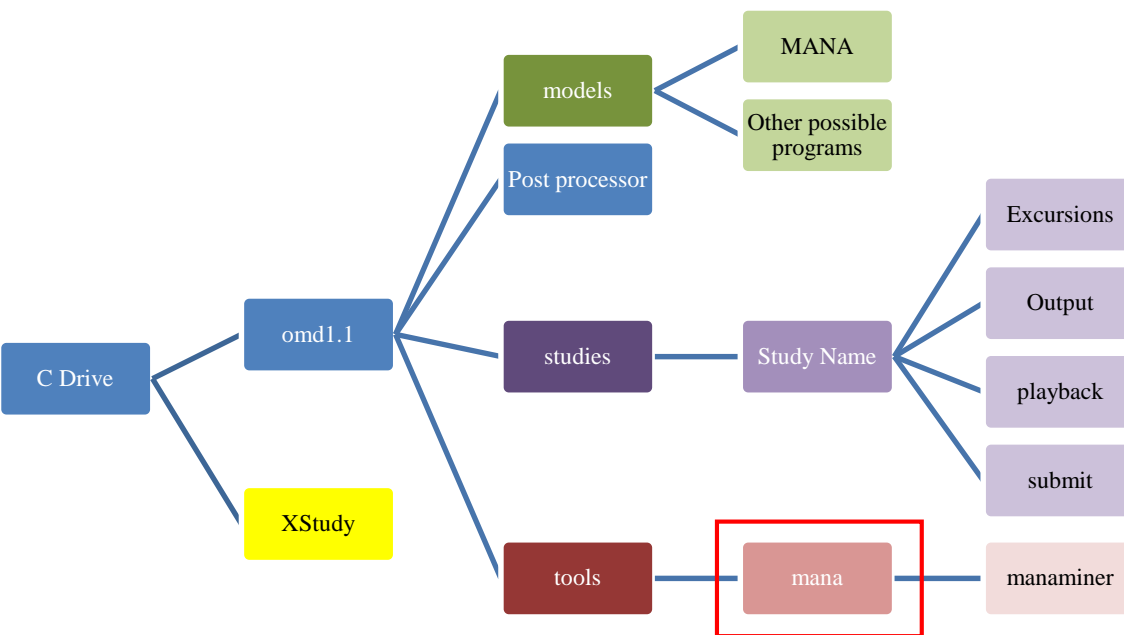


Chart 3. “mana” folder in Folder Hierarchy

We will now copy the ManaMiner post-processing folder into the newly created “mana” folder.

Step 3. Locate the “manaminer” folder. You may have to attach the files from another computer to transfer the files.

Step 4. Copy the “manaminer” folder and all of its contents into the “mana” folder, as shown below:

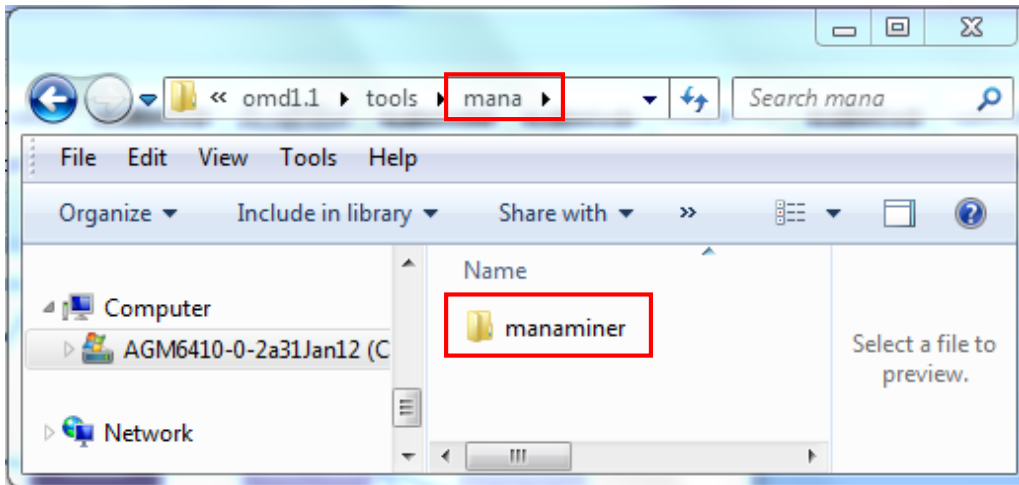


FIGURE 36. Paste the “manaminer” folder into the “mana” folder

The chart below highlights the newly created “mana” folder in the folder hierarchy:

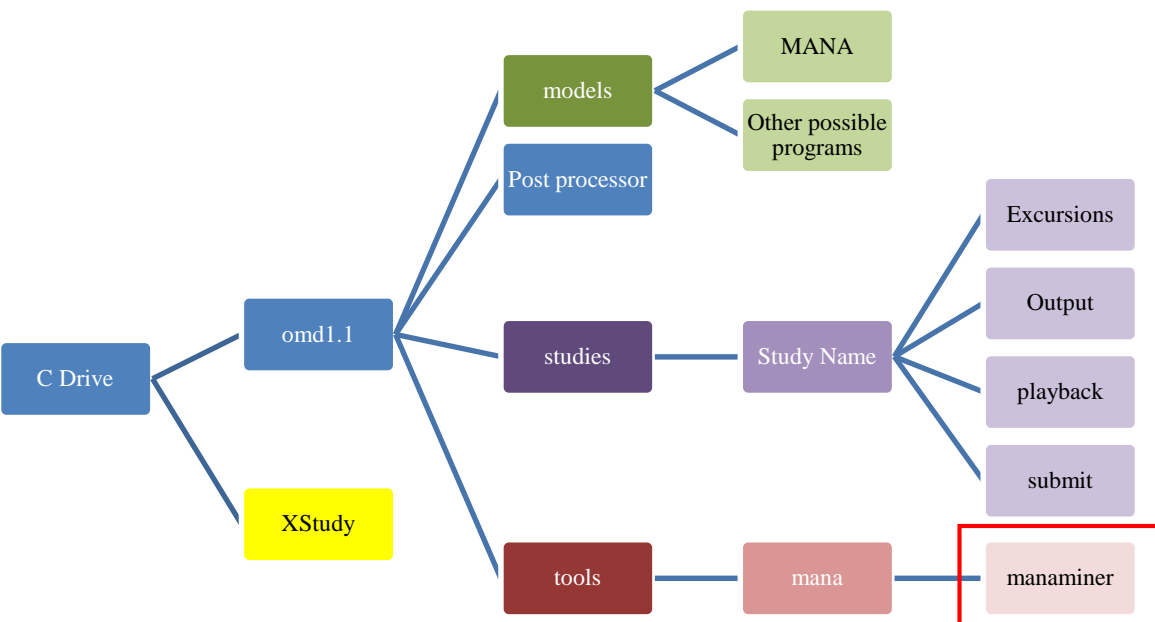


Chart 4. “manaminer” folder in Folder Hierarchy

CREATE THE “STUDY NAME” FOLDER

Create the “Study Name” folder that contains the files specific to your project.

Step 1. Toggle back to the “omd1.1” folder and locate the “studies” folder:

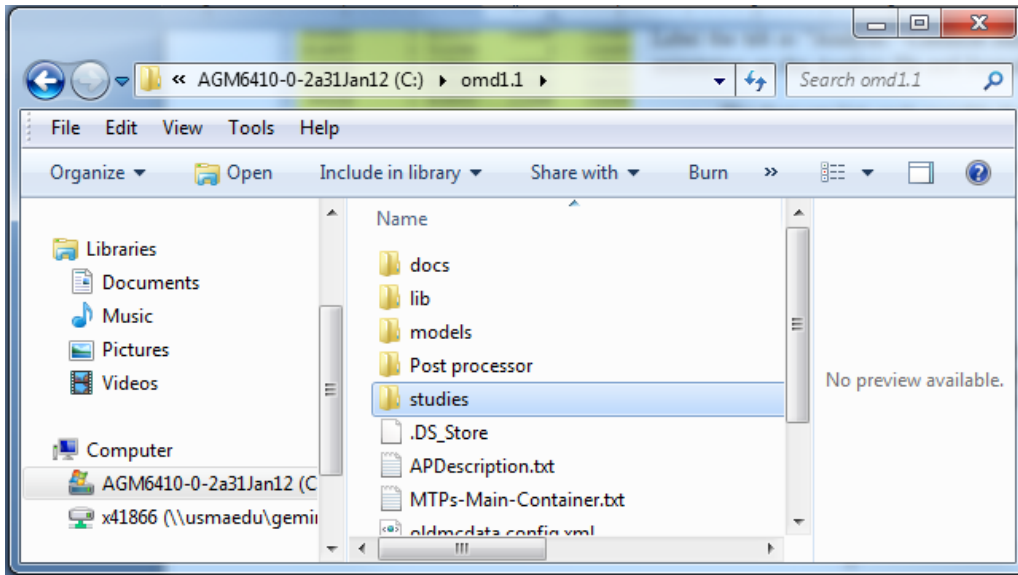


FIGURE 37. Locate the “studies” folder in the “omd1.1” folder

The chart below points out the location “studies” folder in the folder hierarchy:

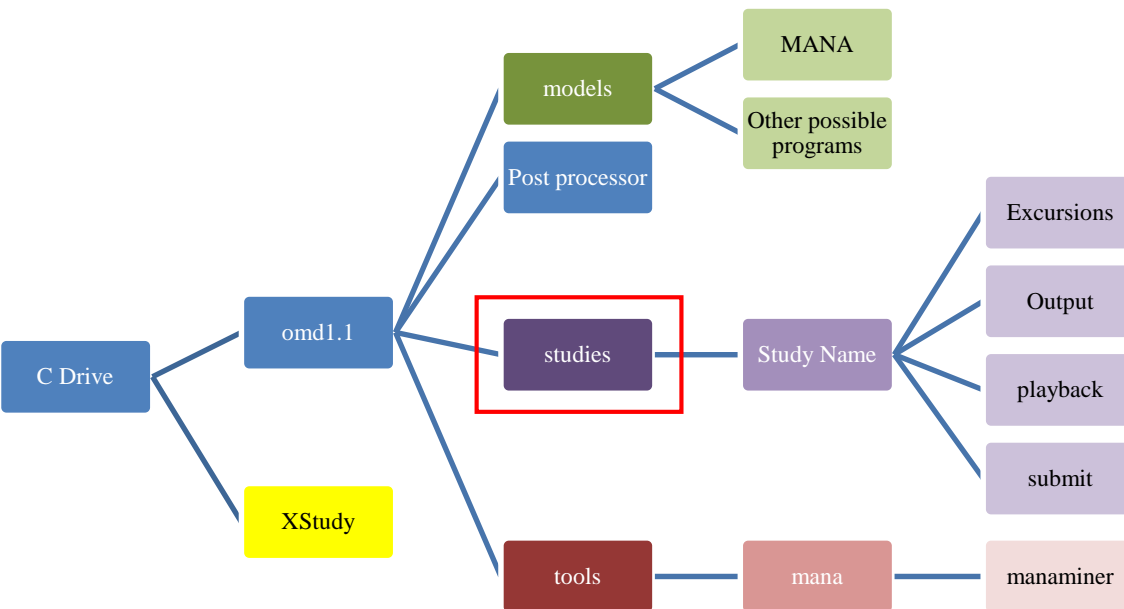


Chart 5. “studies” folder in Folder Hierarchy

Step 2. Open the “studies” folder and create a new folder. This is the “Study Name” folder. Label it as your project name. Ensure the label does not have spaces. In the example below, the Study Name is “Test10DEC.”

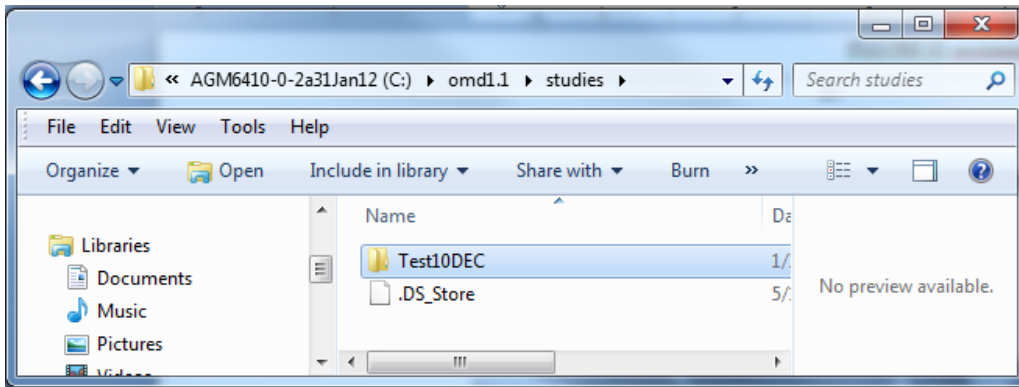


FIGURE 38. Create the “Study Name” folder

Step 3. Navigate to the following files and copy them into your newly created “Study Name” folder:

- a. MANA **Scenario file** (.xml)
- b. MANA terrain and/or elevation maps (if being used) (.bmp).

**Note: it is not necessary to transfer the background map.*

- c. **Executable file** (.csv)
- d. **Analysis file** (.csv)
- e. **Experiment Study file** (.xml)
- f. **moes.dat file**

- i. Path: C drive → “omd1.1” folder → “tools” folder → “mana” folder → “manaminer” folder

The completed “Study Name” folder should resemble the figure below:

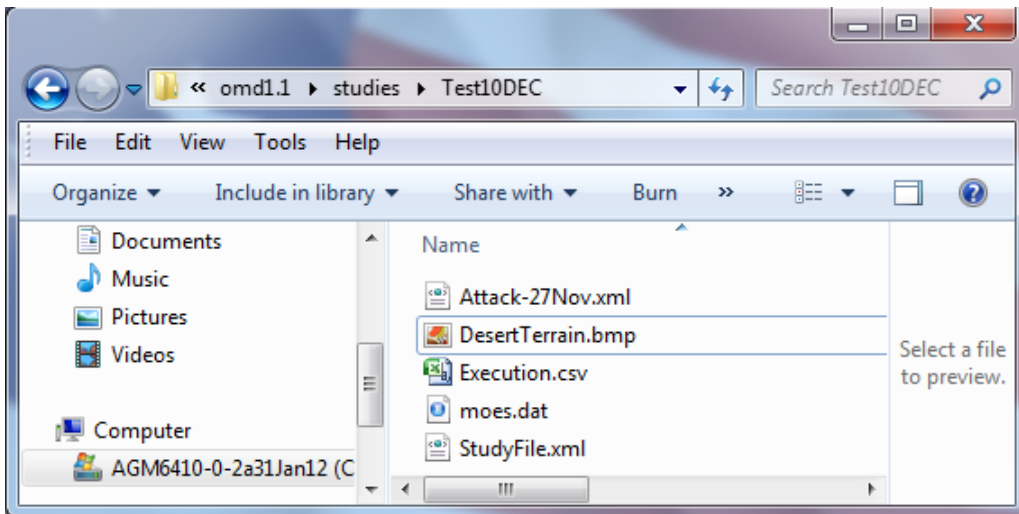


FIGURE 39. Contents of the “Study Name” folder

4.2 Cluster Running Prerequisites

The following is a checklist that should be completed before initiating cluster runs on Condor:

- ✓ Ensure the “MANA” folder (Path to “models” folder: C drive → “omd1.1” → “models” → “MANA”) has the following files:
 - mana.reg
 - MANAC.exe
 - condormana.bat
- ✓ Configure the oldmcdata.config.xml file in Microsoft Visual Studio or another XML viewer
 - Step 1. Navigate to the “omd1.1” folder (Path: C drive → “omd1.1”) and locate the “oldmcdata.config.xml” file as shown below:

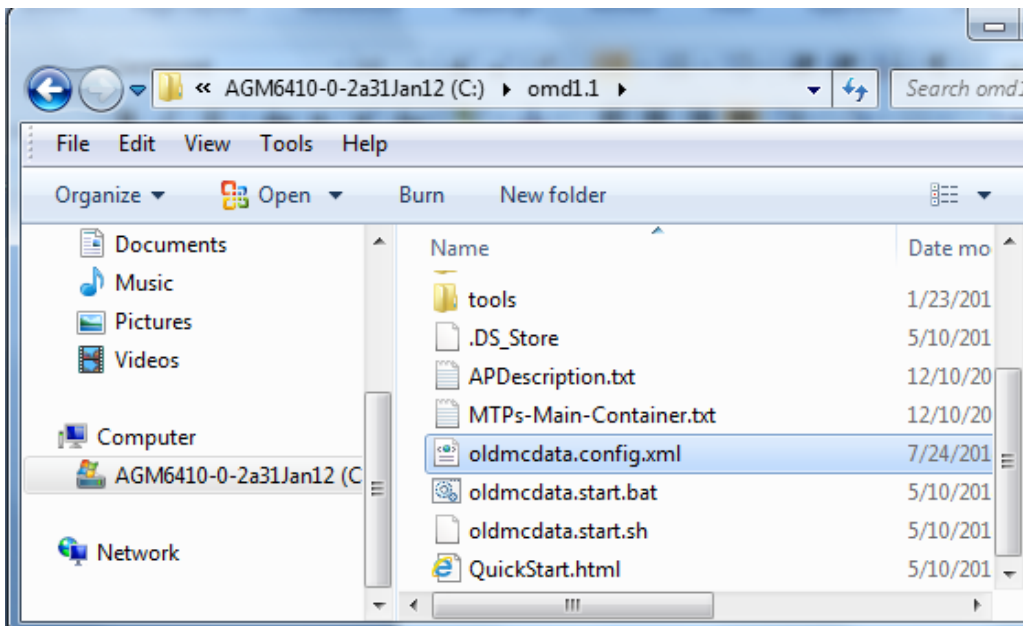


FIGURE 40. Locate the “oldmcdata.config.xml” file within the “omd1.1” folder

- Step 2. Open the “oldmcdata.config.xml” file in Microsoft Visual Studio or any XML viewer. Change the “oldmcdata.config.xml” file in the areas depicted in Figure 41. The file is case sensitive so be sure to check the information is replicated exactly Ensure the major and minor model versions correspond to the versions specified in the Experiment Study file.. Additionally, the slashes in the path location must be forward facing.

The slashes in the path location must be forward slashes.

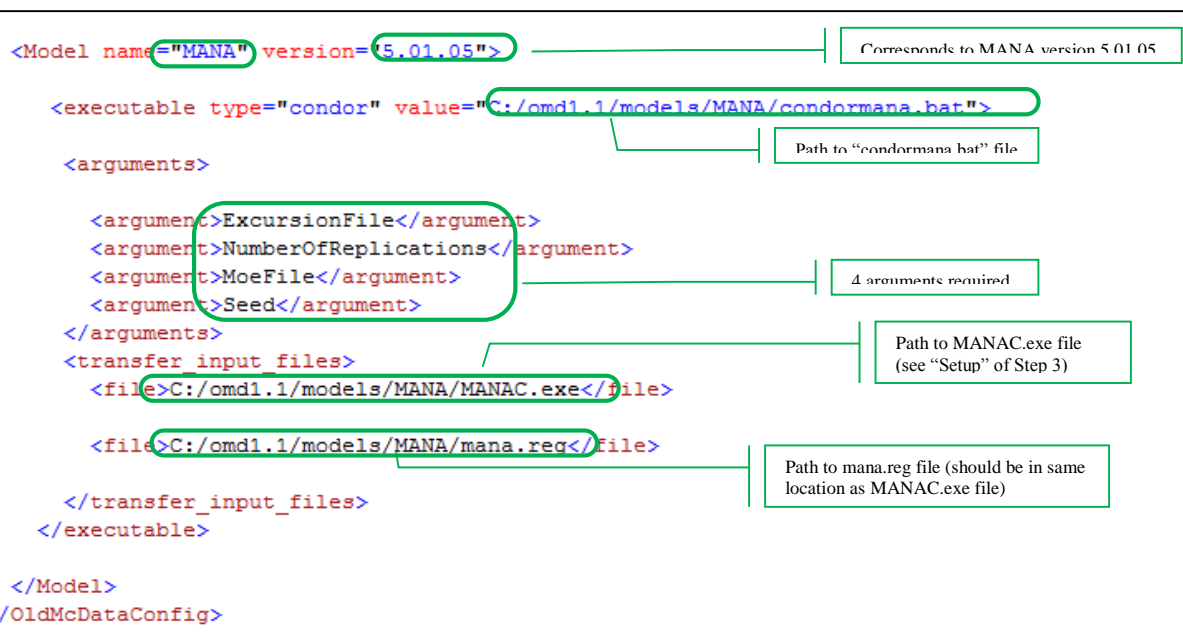


FIGURE 41. "oldmcdata.config.xml" file on Microsoft Visual Studio

- ✓ Configure the Experiment Study file (.xml) to ensure mapping, references, and verbiage mirrors the "oldmcdata.config.xml" file

Step 1. Navigate to the "Study Name" folder (Path: C drive → "omd1.1" → "studies" → "Study Name") and locate the Experiment Study file (.xml). See the figure below:

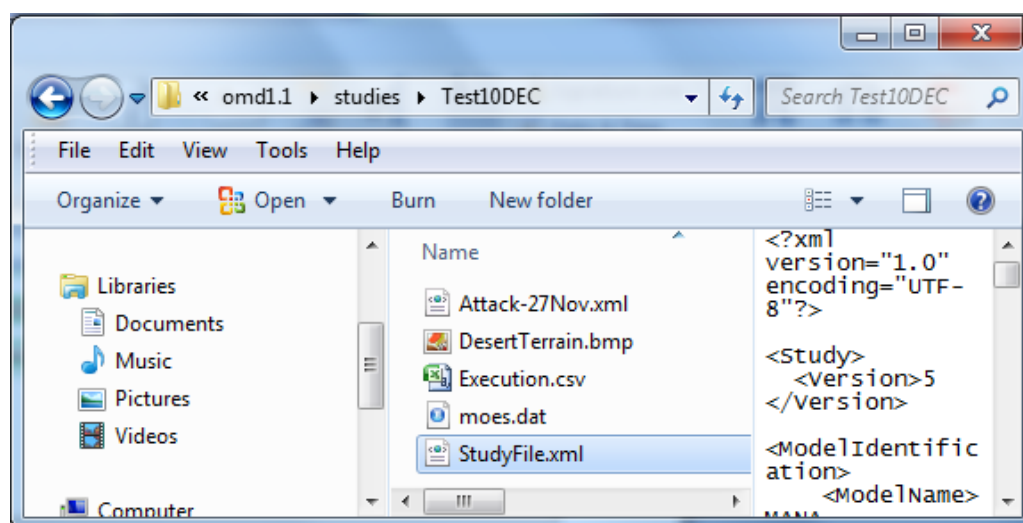


FIGURE 42. Locate the "Experiment Study" file within the "Study Name" folder

Step 2. Open the Experiment Study file (.xml) in an XML viewer. Within your XStudy file, ensure it is linked to the correct version of MANA. The Experiment Study file (.xml) is case sensitive. See the example executable to the right for MANA version 5.01.05. The “Major” pertains to the first portion of the version number. The “Minor” pertains to the last two portions of the version number.

```
<Study>
  <Version>5</Version>
  <ModelIdentification>
    <ModelName>MANA</ModelName>
    <ModelVersion>
      <Major>5</Major>
      <Minor>01.05</Minor>
    </ModelVersion>
  </ModelIdentification>
```

FIGURE 43. “Experiment Study file” XML Script

- ✓ Check the connectivity of the remote computers to the master computers.

Step 1. Open Command Prompt in the master computer from the “omd1.1” folder (see Section 4.2 for further instruction)

Step 2. Issue the following command: `condor_status` (see Figure 43).

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@USMAWKDFCASE	WINDOWS	X86_64	Unclaimed	Idle	0.000	767	8+01:14:42
slot2@USMAWKDFCASE	WINDOWS	X86_64	Unclaimed	Idle	0.000	767	8+01:14:40
slot3@USMAWKDFCASE	WINDOWS	X86_64	Unclaimed	Idle	0.110	767	8+01:14:44
slot4@USMAWKDFCASE	WINDOWS	X86_64	Unclaimed	Idle	0.000	767	8+01:14:46
slot5@USMAWKDFCASE	WINDOWS	X86_64	Unclaimed	Idle	0.000	767	8+01:14:44
slot6@USMAWKDFCASE	WINDOWS	X86_64	Unclaimed	Idle	0.000	767	8+01:14:43
slot7@USMAWKDFCASE	WINDOWS	X86_64	Unclaimed	Idle	0.000	767	8+01:14:46
slot8@USMAWKDFCASE	WINDOWS	X86_64	Unclaimed	Idle	0.000	767	8+01:14:41
slot1@USMAWKDFIMAC	WINDOWS	X86_64	Unclaimed	Idle	0.160	4089	4+05:55:11
slot2@USMAWKDFIMAC	WINDOWS	X86_64	Unclaimed	Idle	0.000	4089	4+05:54:43
slot3@USMAWKDFIMAC	WINDOWS	X86_64	Unclaimed	Idle	0.000	4089	4+05:55:13
slot4@USMAWKDFIMAC	WINDOWS	X86_64	Unclaimed	Idle	0.000	4089	4+05:55:14
slot5@USMAWKDFIMAC	WINDOWS	X86_64	Unclaimed	Idle	0.000	4089	4+05:55:15
slot6@USMAWKDFIMAC	WINDOWS	X86_64	Unclaimed	Idle	0.000	4089	4+05:55:16
slot7@USMAWKDFIMAC	WINDOWS	X86_64	Unclaimed	Idle	0.000	4089	4+05:55:17
slot8@USMAWKDFIMAC	WINDOWS	X86_64	Unclaimed	Idle	0.000	4089	4+05:55:10
Total Owner Claimed Unclaimed Matched Preempting Backfill							
X86_64/WINDOWS	16	0	0	16	0	0	0
Total	16	0	0	16	0	0	0

FIGURE 43. Invoking “condor_status” in Command Prompt

The “central host machine” is the master computer. The “client machines” corresponds to the remote computers. In Figure 43, we have 16 remote machines connected to the master computer. They are currently in an “idle” state.

- If an error message appears, turn to Appendix F

4.3 Launch OldMcData

Step 1. Open the OldMcData Command Prompt. Use one of the 2 methods listed below:

a) Method 1:

1. Locate the “omd1.1” folder within the C drive.
2. Press the [[Shift]] key while simultaneously right clicking the “omd1.1” folder.
3. Left click on “Open Command Prompt from Here” (see Figure 44)

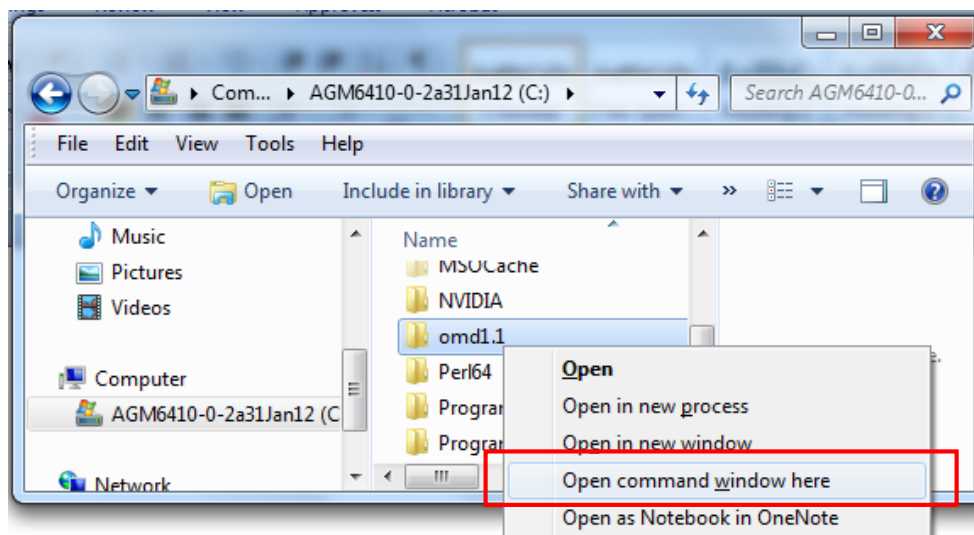


FIGURE 44. Opening Command Prompt from the “omd1.1” folder

b) Method 2:

1. Click on the Start menu and type in “Command Prompt” in the search bar
2. Open Command Prompt
3. Issue the following command: `cd \om*` (see Figure 45)

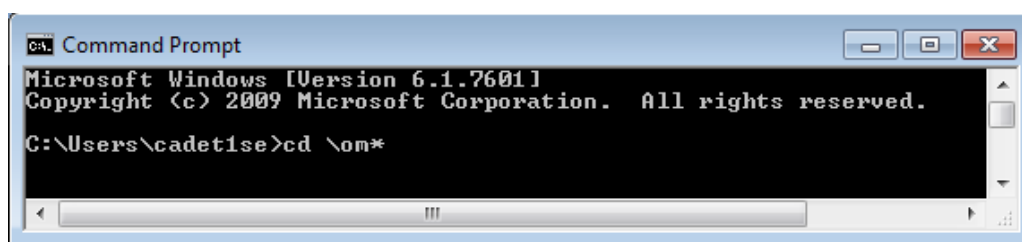


FIGURE 45. Command Prompt

Step 2. Initiate OldMcData

- Issue the following command into the Command Prompt:
`oldmcdata.start.bat [full path to “Study Name” folder] [name of Experiment Study file]`
 - The path to the “Study Name” folder should resemble: `C:\omd1.1\studdies\StudyName`
 - Ex. `oldmcdata.start.bat C:\omd1.1\studdies\Test10DEC StudyFile.xml`
- For troubleshooting options, see Appendix C

Step 3. OldMcData Outputs are saved in the “Study Name” folder (Path: C drive → “omd1.1” folder → “studies” folder → “Study Name” folder). The four folders that are created after executing the cluster run are listed below:

- “Excursions” folder
 - Contains the MANA .xml files for each individual **design point** of the experiment.
- “Output” folder
 - .err and .log files are from Condor
 - .out file records the scripts from Command Prompt
 - Caslocs .csv files: contain the casualty location results. It lists the coordinates of the victims at the time of the incident, squad information, the weapon used, shooter information, and more.
 - Agentstate data: contains casualty results, the number of hits each agent took, and the number of rounds that left each weapon used.
 - Multicontactdetection data: Contains data on which agents killed which agents at what time and with which weapon.

4.4 Monitor Progress using Condor

Enter the following commands in the command prompt to regulate the runs:

Command	Function
condor_status	Shows how many processors are ‘up’ and available
condor_q	Checks how many jobs are still running
condor_rm -all	Kills all jobs running
condor_rm -<IDnumber>	IDnumber is the job ID number you want to kill
condor_submit submit-X.dat	Submits a particular excursion , where X is the excursion number
condor_restart - all condor_master	Restarts Condor. Issue these two commands if none of the jobs are running.
condor_store_cred	Stores a password that is necessary for accessing Condor

Table 2. Condor Commands

Post-Process Output

Invoke the MANA post-processor, manaMiner, to extract and concatenate output data.

5.1 Create Summary File

The Summary file is an aggregation of the MANA output files. It is limited by MANA functionalities in that it only captures the items that are checked in the “Data Output” drop down menu (see Figure 2) when initially creating the Scenario file (.xml). This step is optional and should only be used when post-processing is not needed.

Step 1. Open Command Prompt from “omd1.1” folder

Step 2. Issue the following command:

`manaPP.bat [full path to “Study Name” folder] [name of Experiment Study file] [Summary File Name].csv 5`

- The path of the “Study Name” folder is: C drive → “omd1.1” folder → “studies” folder → “Study Name” folder)
- Summary File Name: For the third input of the command, create a title that the Summary file will be called.
- Ex. `manaPP.bat C:\omd1.1\studies\Test10DEC StudyFile.xml TestOutput.csv 5`



FIGURE 46. Summary File Script on Command Prompt

All slashes in Command Prompt must be backwards facing

Step 3. Check the “Study Name” folder to see if the Summary file has been generated. The screenshot below is an example of a Summary file:

Identifiers	SDRSpeed	SDRKillHt	SDRStealt	SDRVisHe	SDRDR1	SDRDR2	SDRDR3	SDRDR4	SDRDR5	Run	Seed	Alleg1Cas	Alleg2Cas	Alleg0Cas	Blue Reac	Red Reac	Neutral Ri	Steps	Sqd1Cas	Sqd2C
1	IOEO	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	1	3.14E+08	6	12	0	No	No	No	78	6
2	IOEO	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	2	3.39E+08	12	11	0	No	No	No	80	12
3	IOEO	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	3	-1E+09	10	12	0	No	No	No	78	10
4	IOEO	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	4	-3.3E+08	4	12	0	No	No	No	81	4
5	IOEO	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	5	2.11E+09	12	10	0	No	No	No	78	12
6	IOEO	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	6	1.14E+09	11	12	0	No	No	No	79	11
7	IOEO	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	7	3.69E+08	11	12	0	No	No	No	78	11
8	IOEO	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	8	6.82E+08	11	12	0	No	No	No	79	11
9	IOEO	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	9	7.47E+08	8	12	0	No	No	No	80	8
10	IOEO	10.61842	3	82.63158	5.353947	13.75605	412.6815	687.8025	1375.605	2063.408	10	-1.1E+09	6	12	0	No	No	No	78	6
11	IOE1	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	1	3.14E+08	6	8	0	Yes	No	No	54	6
12	IOE1	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	2	-5.9E+08	7	7	0	Yes	No	No	54	7
13	IOE1	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	3	3.99E+08	8	11	0	Yes	No	No	54	8
14	IOE1	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	4	-1.7E+09	7	10	0	Yes	No	No	54	7
15	IOE1	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	5	-2.3E+08	9	10	0	Yes	No	No	54	9
16	IOE1	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	6	2.33E+08	6	12	0	No	No	No	43	6
17	IOE1	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	7	-1.7E+09	7	11	0	Yes	No	No	54	7
18	IOE1	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	8	2.44E+08	10	9	0	Yes	No	No	55	10
19	IOE1	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	9	-8.9E+08	8	11	0	Yes	No	No	54	8
20	IOE1	10.14474	5	73.1579	2	15.16289	454.8867	758.1445	1516.289	2274.434	10	-4.1E+08	8	9	0	Yes	No	No	54	8
21	IOE2	12.84237	4	63.68421	6.671053	17.88105	536.4315	894.0525	1788.105	2682.158	1	3.14E+08	11	12	0	No	No	No	71	11
22	IOE2	12.84237	4	63.68421	6.671053	17.88105	536.4315	894.0525	1788.105	2682.158	2	9.15E+08	12	12	0	No	No	No	74	12
23	IOE2	12.84237	4	63.68421	6.671053	17.88105	536.4315	894.0525	1788.105	2682.158	3	3.78E+08	12	11	0	No	No	No	70	12
24	IOE2	12.84237	4	63.68421	6.671053	17.88105	536.4315	894.0525	1788.105	2682.158	4	-9.9E+08	12	10	0	No	No	No	70	12

FIGURE 47. Summary File on Microsoft Excel

5.2 Post-Processing Setup

PRE-CONDITIONS The following is a checklist that should be completed before post-processing output:

- ✓ Ensure the “Output” folder (Path: C drive → “omd1.1” folder → “studies” folder → “Study Name” folder → “Output” folder) includes agtendstates, caslocs, and mdet csv files. (See Figure 48) These three files pertain to the options checked under the “Data Output” tab: “Record casualty location data,” “Record agent state data,” and “Record multi-contact detection.”

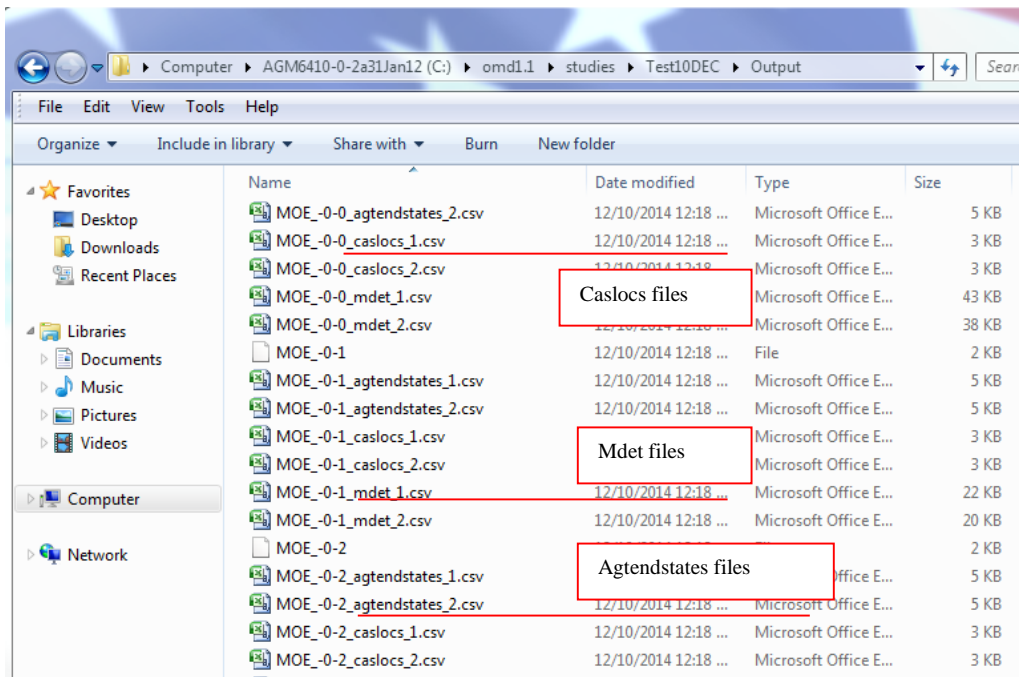


FIGURE 48. Output Folder in the omd1.1 folder

- ✓ Check to ensure your “Study Name” folder (Path: C drive → “omd1.1” folder → “studies” folder → “Study Name” folder) has the following files:
 - MANA Scenario file [.xml]
 - Experiment Study file [.xml]
 - Analysis file [.csv]: This is the original DOE matrix created before the Translation and Execution files.
 - moes.dat file: copy and paste from the “manaminer” folder (Path: C drive → “omd1.1” folder → “tools” folder → “mana” folder → “manaminer” folder) to the “Study Name” folder.

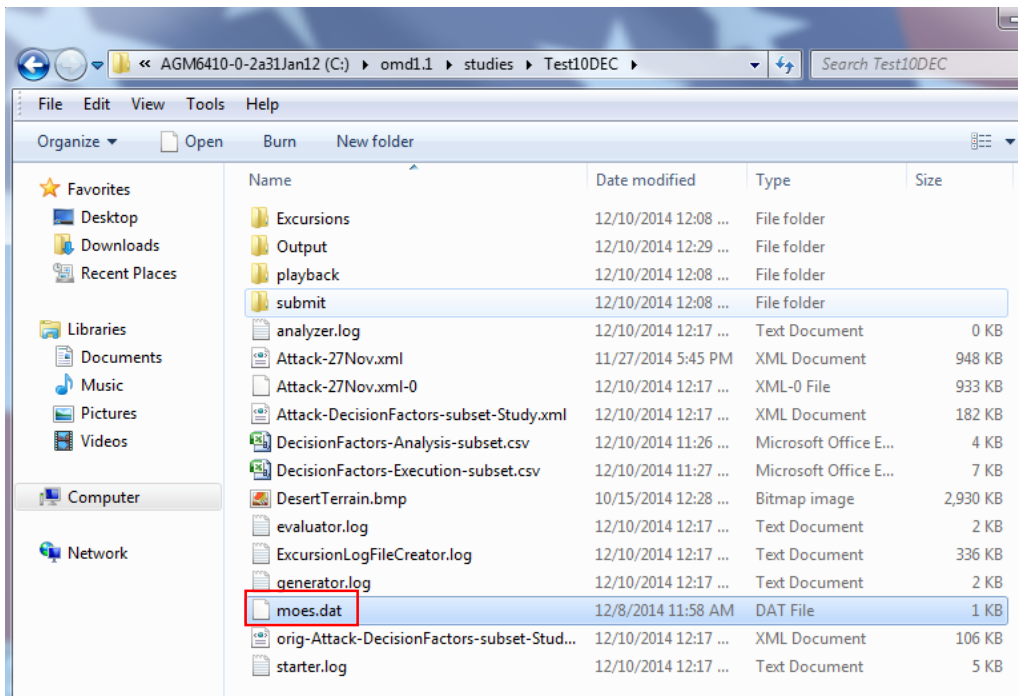


FIGURE 49. Check contents of the “Study Name” folder

- ✓ Personalize the moes.dat file (Path: C drive → “omd1.1” folder → “studies” folder → “Study Name” folder). The moes.dat file contains all the measures of effectiveness that are of interest. The following is a list of MOEs available:

MOE	Description
TWA_SA	Time weighted average of the Blue forces' situational awareness
TWA_REDKIA	Time weighted average of the Red forces killed
ClassDist	
RedKIA	Tracks the number of Red forces killed.
BlueKIA	Tracks the number of Blue forces killed.
CivKIA	Tracks the number of civilians killed.
IED_Det	Tracks the number of IEDs detonated
ATK_Complete	

RedKillTime,0.5	
NoHits	Tracks the number of hits taken by the Blue force
IDFCivKill	
NoRounds	Tracks the number of rounds expelled
BLOSDet	Tracks the number of beyond line of sight detection for the Blue forces
BlueForceFratricide	Tracks fratricide incidents amongst the Blue forces
AllMultiRunMOEs	

Table 3. Measures of Effectiveness

- ✓ Open the run.manaminer.bat file in the “manaminer” folder (Path: C drive → “omd1.1” folder → “tools” folder → “mana” folder → “manaminer” folder) to ensure it is mapped to the manaminer.singleDR.R file path, which is also located in the “manaminer” folder. In the example below, the path to the manaminer.singleDR.R file is underlined.

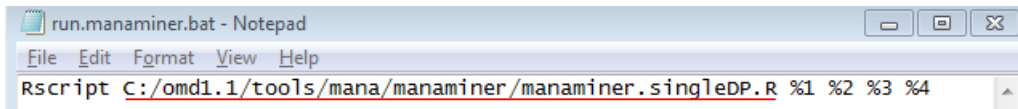


FIGURE 50. “run.manaminer.bat” file opened in Notepad

MANAMINER REQUIREMENTS For the customized ManaMiner post-processing script to work, the following requirements must be met:

1. R packages must be installed in the R system library.
2. ManaMiner and R environment variables must be created.

INSTALL “R” PACKAGES Find the path to the R System Library

Step 1. Open “RStudio”

Step 2. Input the following command in the R console: `.libPaths()`, as shown in Figure 51.

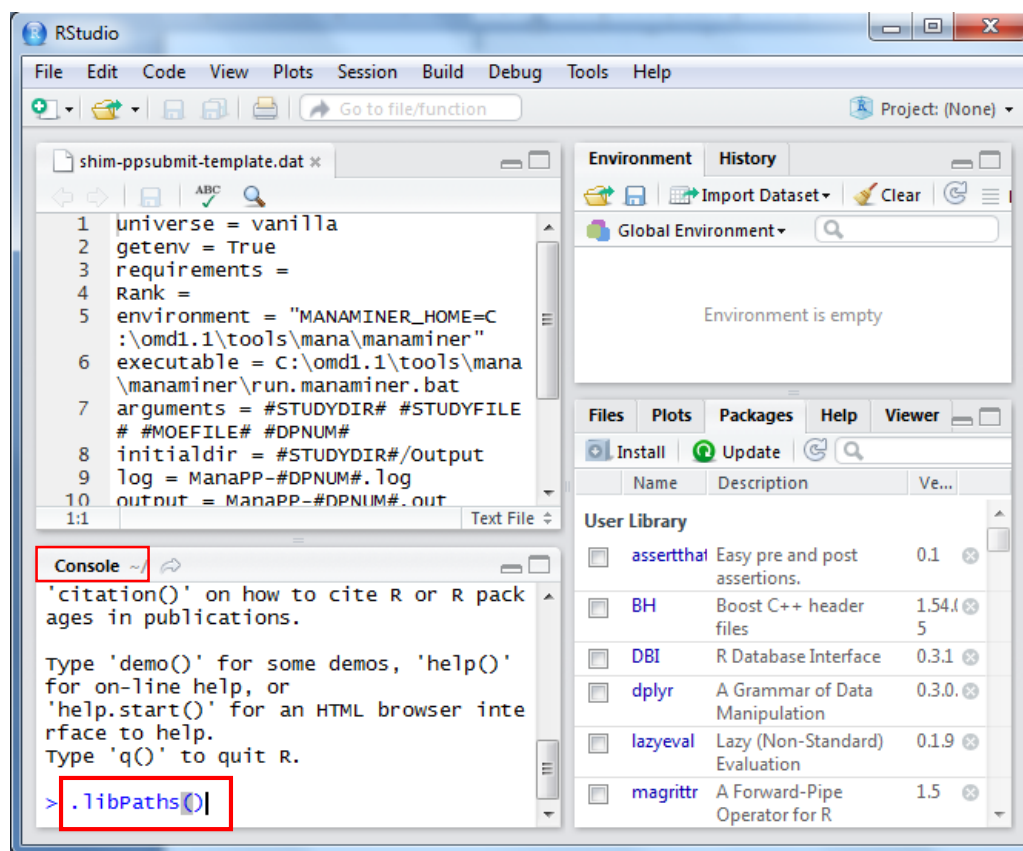


FIGURE 51. R Studio Console

Step 3. Two paths will appear (see Figure 52). The first path is usually the personal library. The second path is usually the system library path.

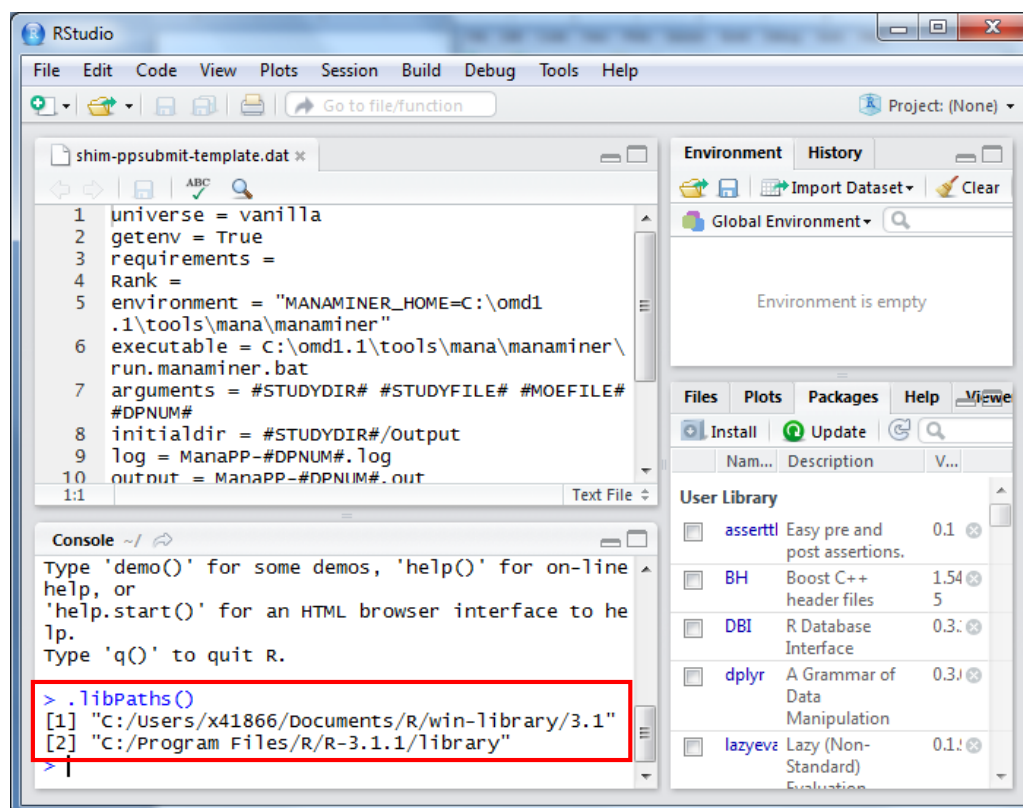


FIGURE 52. Example output script from invoking the “`.libPaths()`” command in R

Install R Packages into the R System Library

**Note: West Point users require administrative privileges to install packages.*

The following R packages must be installed in the R system library:

- dplyr
- reshape2
- plyr
- stringr
- XML

Step 1. Input the following command in the R console:

```
install.packages(c(list of packages),lib="<path to R system library>")
```

- For the “list of packages” segment of the command, place quotation marks around the package names and separate them by a comma.

- The “path to R system library” segment should resemble: C:/Program Files/R/R-3.1.1/library

ex. `install.packages(c("dplyr","reshape2","plyr","stringr","XML"),lib="C:/Program Files/R/R-3.1.1/library")`

ENVIRONMENT VARIABLES

Create manaminer and R environment variables

**Note: West Point users require administrative privileges to create environment variables.*

Step 1. Right click on the Computer icon on your Desktop and click on “Properties.”

Step 2. In the “Properties” window select the “Advanced System Settings” button on the left column.

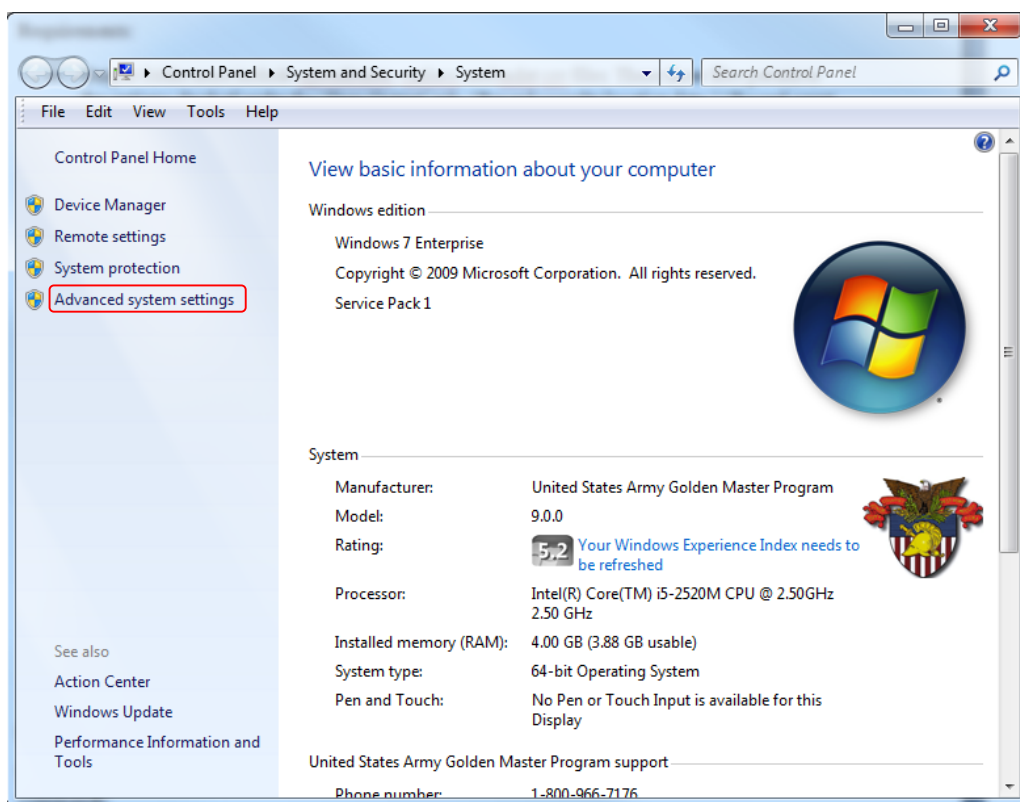


FIGURE 53. “Properties” window

Step 3.

In the “System Properties” window select “Advanced” tab and click on “Environment Variables...” button given at the bottom of the window

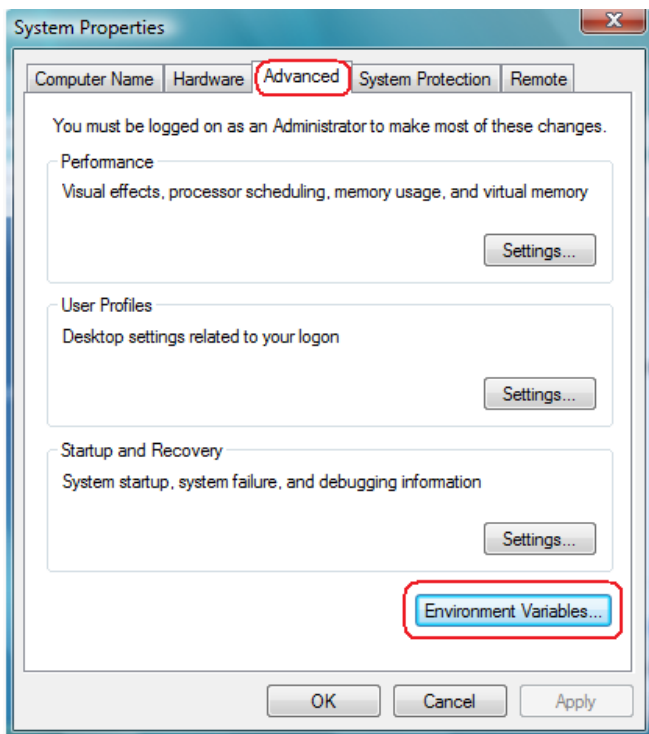


FIGURE 54. “System Properties” window

p 4. Add a new variable by clicking on “New...” button. In the New User Variable dialog box type in the following Variable names and Variable values and click “OK”

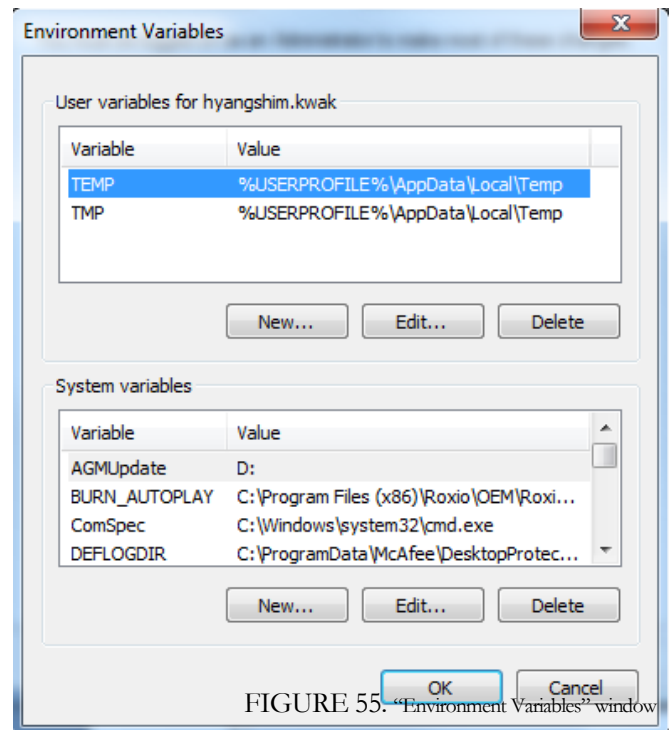


FIGURE 55. “Environment Variables” window

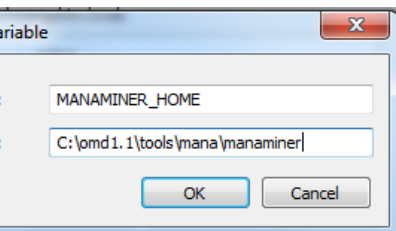


FIGURE 56. “New System Variable” Window with ManaMiner variable

a. MANAminer Environment Variable (Figure 56)

- i. Variable name: MANAMINER_HOME
- ii. Variable value: <path to manaminer folder>

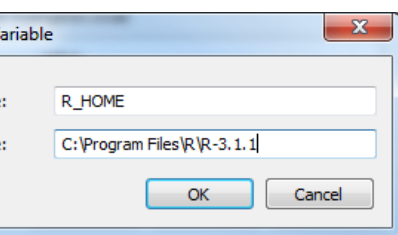


FIGURE 57. “New System Variable” Window with R Environment variable

b. R Environment Variable (Figure 57)

- i. Variable name: R_HOME
- ii. Variable value: <path to location of where R is stored>

Step 5. Edit Path Environment Variable: Locate the Path variable and type in %R_HOME%\bin after the existing string. Separate the inputted string from the previous with a “;” (See Figure 58).

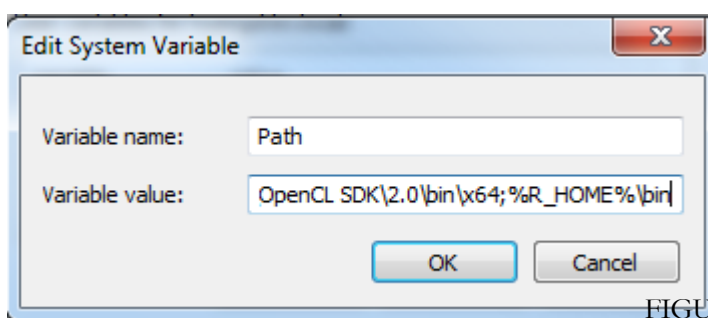


FIGURE 58. “Edit System Variable” window

Step 6. Check the “shim-ppsubmit-template.dat” file (Path: C drive → “omd1.1” folder → “tools” folder → “mana” folder → “manaminer” folder) to ensure that the “MANAMINER_HOME” environment variable is mapped to the correct path and the “executable” line is mapped to the location of the “run.manaminer.bat” file (Path: C drive → “omd1.1” folder → “tools” folder → “mana” folder → “manaminer” folder).

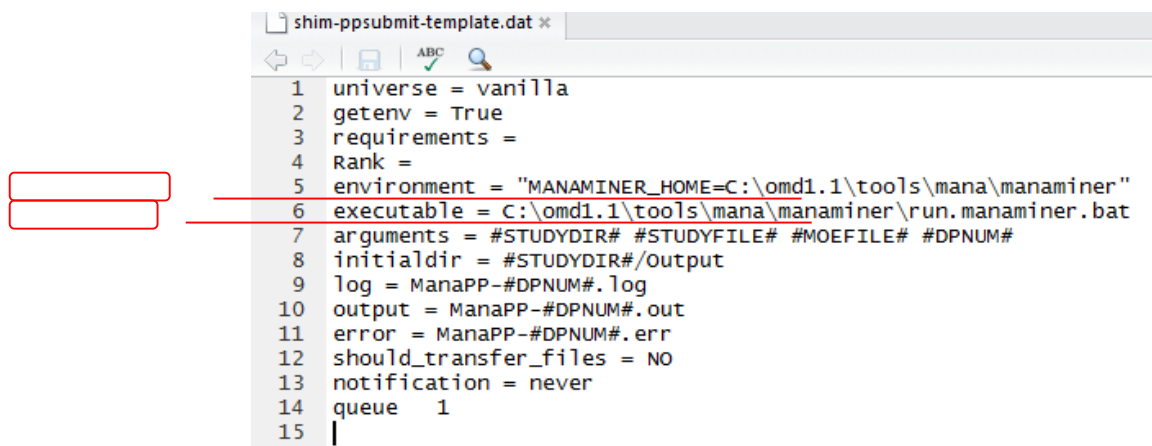
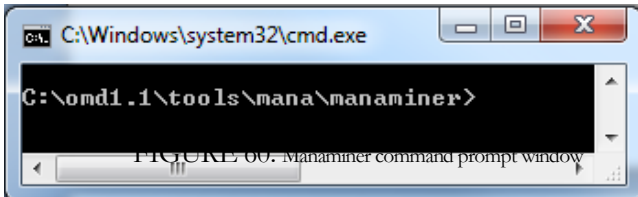


FIGURE 59. “shim-ppsubmit-template.dat” file on Microsoft Visual Studio

5.3 Concurrent Processing of Design Points

Concurrent Processing relies on two tasks, each run sequentially. The first task, or script (run.manaminer.jobs.R), takes the following inputs: Scenario file (.xml), the Analysis file, the “moes.dat” file, and the “shim-ppsubmit-template.dat” file. The second task or script (concat.manaminer.jobs.R), is run after you have completed the first task and each post-processing job is complete.



Step 1. Open the MANAminer directory in Command Window (Figure 60)

Step 2. Invoke the following command (see Figure 61):

Rscript run.manaminer.jobs.R <full path to “Study Name” folder> <name of your Scenario file> <name of your moes.dat file> <name of your Analysis file> shim-ppsubmit-template.dat

- The path to the “Study Name” folder should resemble: C:/omd1.1/studies/StudyName
- The “moes.dat” file is located in your “Study Name” folder
- Ex. Rscript run.manaminer.jobs.R C:\omd1.1\studies\Test10DEC Attack.xml moes.dat Analysis.csv shim-ppsubmit-template.dat

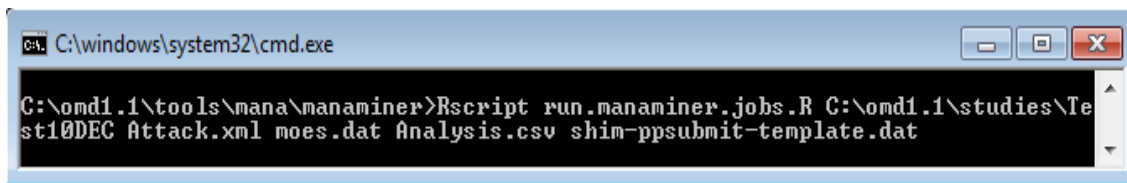


FIGURE 61. Example invocation of run.manaminer.jobs.R script

After that set of post-processing jobs is complete, you will need to run the second script that concatenates all the data and merges the DOE with that output. It writes the completed output to a file called "<your “Study Name” folder>-allOutput.csv", where it takes the name of your study folder.

Step 3. Invoke the following command in the MANAminer directory:

Rscript concat.manaminer.jobs.R <full path to “Study Name” folder> <name of Executable file>

- Ex. Rscript concat.manaminer.jobs.R C:\omd1.1\studies\Test10DEC Execution.csv

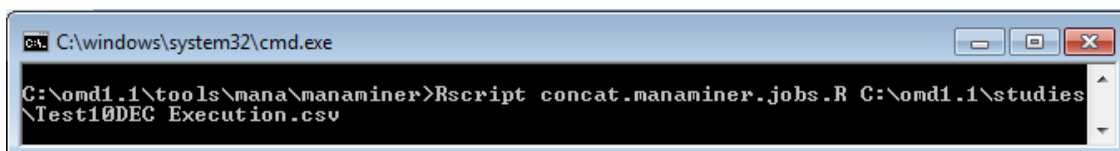


FIGURE 62. Example invocation of “concat.manaminer.jobs.R” script

5.4 Sequential Processing of Design Points

This is a backup method should the concurrent processing of design points (section 5.3) fail. This script post-processes each design point in order, and hence, can be really slow for lots of output. It calls the `manaminer.singleDP.R` script for each design point, concatenates the results when complete, and puts the concatenated output in your “Study Name” folder, with a file name called “allDP.output.csv”.

Within the manaminer command prompt window, invoke the following command:

`Rscript manaminer.allDP.R <full path to “Study Name” folder> <name of your Scenario file> <name of your Executable file> moes.dat`

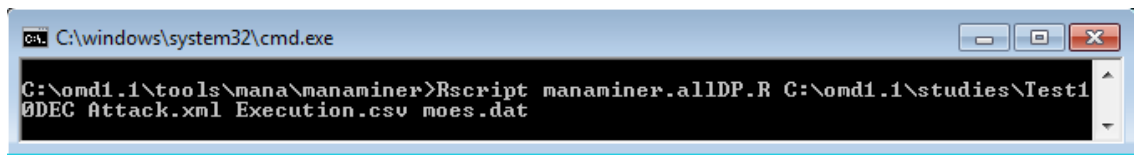


FIGURE 63. Example invocation of “manaminer.allDP.R” script

5.5 Contents of Output Folder

The “Output” folder is located within the “Study Name” folder. The path is as follows: C drive → “omd1.1” folder → “studies” folder → “Study Name” folder → “Output” folder

Within the “Output” folder are several types of files:

ManaPP-#.out file shows the MOEs and the directory of the output files.

```
ManaPP-0.out - Notepad
File Edit Format View Help
using measures_to_compute_file: C:/omd1.1/studies/Test10DEC/moes.dat
manaminer_home: C:/omd1.1/tools/mana/manaminer
output_dir: C:/omd1.1/studies/Test10DEC/output
study_file: C:/omd1.1/studies/Test10DEC/Attack-DecisionFactors-subset-Study.xml
measures to compute:
  compute_TWA_SA
  compute_TWA_RED KIA
  compute_ClassDist
  compute_Red KIA
  compute_Blue KIA
  compute_Civ KIA
  compute_IED_Det
  compute_ATK_Complete
  compute_RedKillTime
  compute_BlueNOHits
  compute_IDFCivKill
  compute_BlueNOHits
  compute_BLOSdet
  compute_BlueForceFratricide
  compute_AllMultiRunMOEs
post-processing DP number: 0
mana files listing:
  C:/omd1.1/studies/Test10DEC/output/MOE_-0-0
  C:/omd1.1/studies/Test10DEC/output/MOE_-0-0_agtendstates_1.csv
  C:/omd1.1/studies/Test10DEC/output/MOE_-0-0_agtendstates_2.csv
  C:/omd1.1/studies/Test10DEC/output/MOE_-0-0_caslocs_1.csv
  C:/omd1.1/studies/Test10DEC/output/MOE_-0-0_caslocs_2.csv
  C:/omd1.1/studies/Test10DEC/output/MOE_-0-0_mdmet_1.csv
  C:/omd1.1/studies/Test10DEC/output/MOE_-0-0_mdmet_2.csv
computing TWA_SA for rep_num: 1
computing TWA_RED KIA for rep_num: 1
computing BlueClassDist for rep_num: 1
computing Red KIA for rep_num: 1
computing Blue KIA for rep_num: 1
computing Civ KIA for rep_num: 1
computing IED_Det for rep_num: 1
computing ATK_Complete for rep_num: 1
computing RedKillTime for rep_num: 1
computing BlueNOHits for rep_num: 1
computing IDFCivKill for rep_num: 1
computing BlueNOHits for rep_num: 1
computing BLOSdet for rep_num: 1
computing BlueForceFratricide for rep_num: 1
computing AllMultiRunMOEs for rep_num: 1
computing TWA_SA for rep_num: 2
computing TWA_RED KIA for rep_num: 2
computing BlueClassDist for rep_num: 2
computing Red KIA for rep_num: 2
computing Blue KIA for rep_num: 2
computing Civ KIA for rep_num: 2
computing IED_Det for rep_num: 2
computing ATK_Complete for rep_num: 2
computing RedKillTime for rep_num: 2
computing BlueNOHits for rep_num: 2
computing IDFCivKill for rep_num: 2
computing BlueNOHits for rep_num: 2
computing BLOSdet for rep_num: 2
computing BlueForceFratricide for rep_num: 2
computing AllMultiRunMOEs for rep_num: 2
writing melted results for dp_num: 0 to : C:/omd1.1/studies/Test10DEC/output/Attack-27Nov.-0-0.melted.output.csv
```

FIGURE 36. Example ManaPP-#.out file

ManaPP-#.err file shows any errors or warnings that were encountered

Appendix A: References

Upton, Stephen. “OldMcData- The Data Farmer User’s Manual” Version 1.1. June 18, 2010.

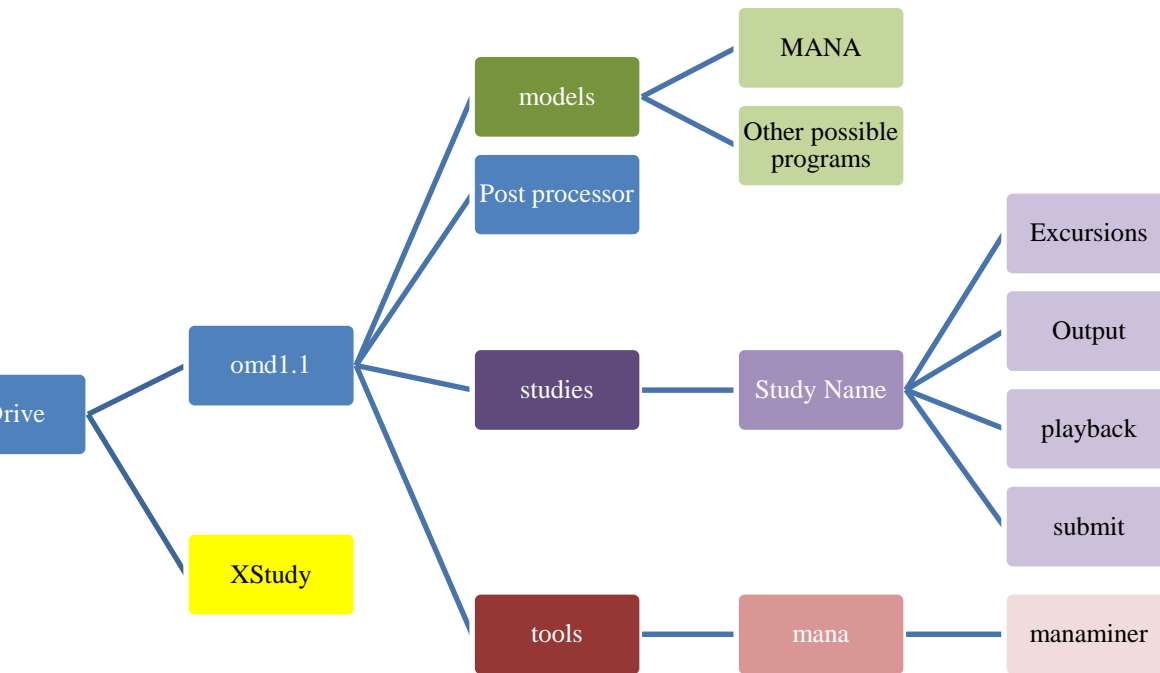
MacCalman, Alexander D., and Hyangshim Kwak. “Engineering Resilient System Architectural Considerations to Incorporate Experimental Design Methods.” United States Military Academy, Department of Systems Engineering. September, 2014. Print.

Appendix B: Glossary

- ✓ **Analysis File** (.csv): The experimental design the analysts will use to perform linear regression after appending the output data to it.
- ✓ **Design of Experiment** (DOE): The field of DOE allows us to efficiently explore a high-dimensional tradespace problem in a feasible amount of time.
- ✓ **Design Point** (DP): An individual row on the executable file (.csv). Each row represents a different set of settings to run. We sometimes refer to a DP as an excursion.
- ✓ **Excursion**: See “**Design Point**”
- ✓ **Executable File** (.csv): A Microsoft Excel file where each column is a variable in the design of experiment and each row is a design point. The file is uploaded on XStudy to map the variables in the design and the elements of a MANA scenario file.
- ✓ **Experiment Study File** (.xml): This file is created using the XStudy program. It creates the mapping between the variables in the design and the elements of a MANA scenario file. It also specifies version of MANA that is being used and the number of random replications to run for each design point.
- ✓ **Scenario File** (.xml): A MANA file that contains the baseline parameters for all variables that will be manipulated in the DOE. It contains all the details for the scenario.
- ✓ **Summary File** (manaPP.bat): A summary file is created after all simulations in the DOE have been run. The file is a compilation of the outputs from every simulation. It concatenates the casualty data, number of steps run, and whether or not Blue/Red/Neutral reached their final goal.
- ✓ **Translation File** (.csv): An intermediary file created in Microsoft Excel that translates the analysis file into the executable file.

Appendix C: OldMcData Folder Organization Layout

OldMcData automatically creates folders. Knowing the location and contents of each folder is critical for successful cluster runs and efficient data abstraction.



Folder Name	Description and Contents
Excursions	Contains the MANA .xml files for each individual design point of the experiment.
MANA	Contains “condormana.bat” file
models	Contains individual folders of each program that Condor and OldMcData will run simulations on
omd1.1	Contains all folders required and data processed through OldMcData
Output	Contains .err and .log files from Condor, .out files from Command Prompt, and output data from MANA.
playback	Did not use folder
Post processor	Did not use folder
studies	Contains individual folders for each study
Study name	Contains the MANA Scenario (.xml) and Executable (.csv) files.
submit	Did not use folder
tools	Contains post processing folder
mana	Contains “manaminer” folder
manaminer	Contains files required for post-processing
XStudy	Contains “xstudy.bat” file

Appendix D: Installing XStudy

- ✓ Requires JAVA versions greater than 6.0
- ✓ Installer places a xstudy.bat file into the directory
- ✓ Install XStudy into the C drive
- ✓ Double-clicking the xstudy.bat file should start the XStudy application

Appendix E: Downloading Condor

Download Condor only if completely necessary. The procedure for downloading Condor is complicated because Condor requires supplemental programs. If possible, opt to use a master computer that already has Condor installed.

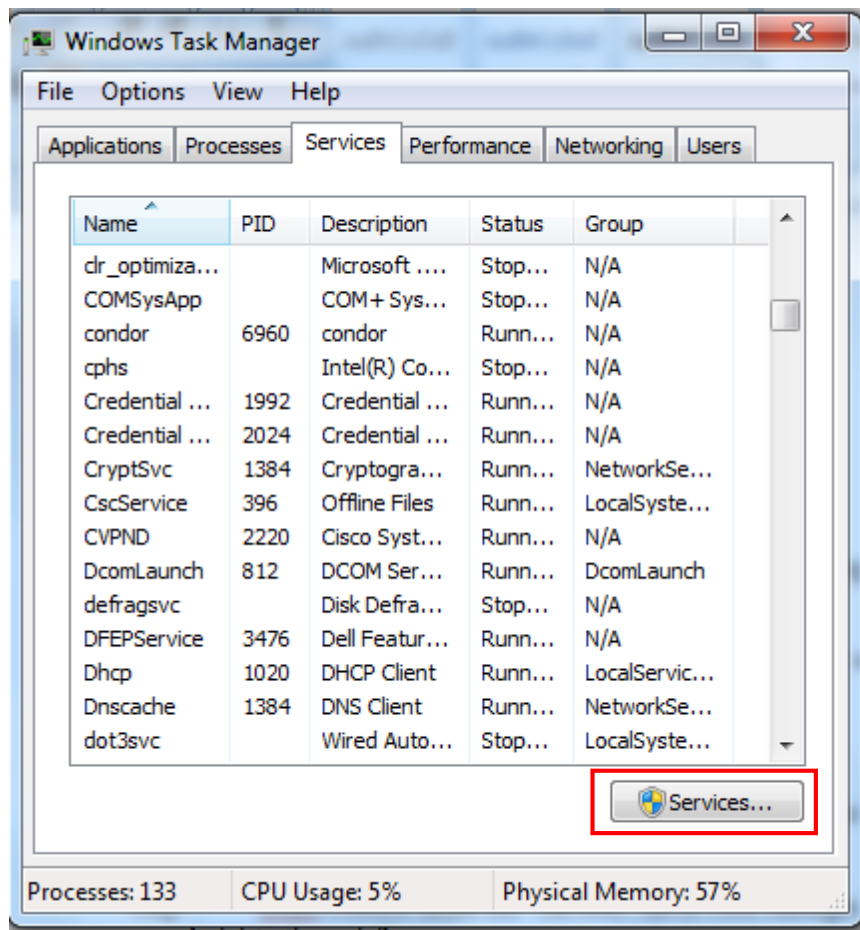
If you do download Condor, ensure you download Cygwin and GoPearl as well. Although you install Condor, you will not be able to farm any simulations out to remote computers because they do not recognize your computer.

Appendix F: OldMcData Troubleshooting Options

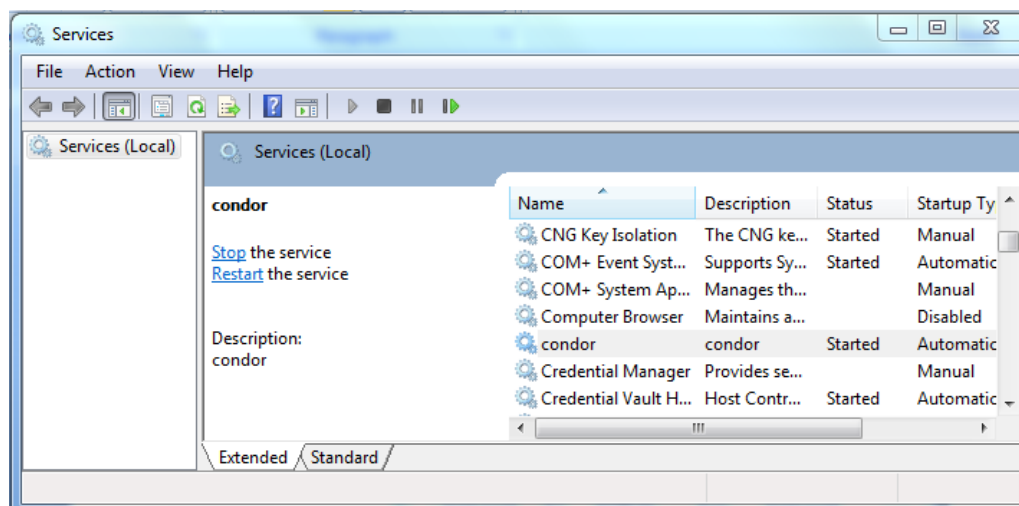
If you are unable to farm out to other client machines

- Step 1. Open Command Prompt on the master computer from the “omd1.1” folder
- Step 2. Issue `condor_restart -all`
- Step 3. In the remote computers, issue the following command: `condor_master`
- Step 4. In the master computer, press `[[control]]`, `[[alt]]`, and `[[delete]]` simultaneously.
- Step 5. Select “Start Task Manager”
- Step 6. Click on the “Services” tab and select the “Services” button.

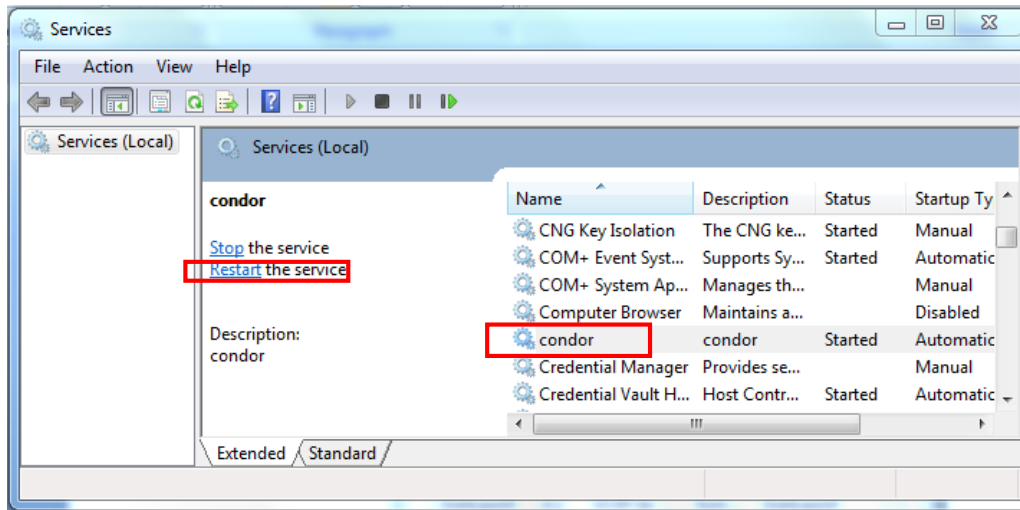
The following requires administrative privileges



Step 7. The “Services” window will appear, as shown below:



Step 8. Locate “condor” under the “Name” column and select “Restart.”



Step 9. Navigate back to the “Windows Task Manager.”

Step 10. Check task manager to ensure Condor demons are running in “All Users” folder

Step 11. Look up start and master log on remote machines

Appendix G: Modifying the “condor_config” file

The “condor_config” path location is: C drive → “condor” folder

- ✓ The following steps must be taken every time you modify the condor_config file:
 - Open up the “Computer Management” window
 - [[Start menu]] → Right Click on “Computer” → Left Click on “Manage”
 - Stop Condor
 - Double click on “Services and Applications” → “Services” → “Condor” → “Stop”
 - Check to see if Condor is alive by inputting condor_status in Command Prompt.

Appendix H: Points of Contact for Additional Help

Mary McDonald: MANA expert

Research Associate at SEED (Simulation Experiments & Efficient Designs) Center for Data Farming
Operations Research Department Naval Postgraduate School

Email: mlmcdona@nps.edu

Stephen Upton: Data Farming Expert

Research Associate at SEED (Simulation Experiments & Efficient Designs) Center for Data Farming
Operations Research Department Naval Postgraduate School

Email: scupton@nps.edu

John Melendez: SE Department Technical Expert (for USMA DSE only)

Simulation Warfighter Manager at the Department of Systems Engineering at West Point

Email: john.melendez@usma.edu

Business Phone: 845)938-5872

Appendix B: Design Creator Front-End User Manual

Note: This user manual is a modified excerpt from the appendix in MacCalman (2013) and refers to chapters in this dissertation:

MacCalman, A. D. 2013. Flexible Space-Filling Designs for Complex System Simulations. Doctoral dissertation. Monterey, CA, Naval Postgraduate School. This dissertation can be found at: http://calhoun.nps.edu/bitstream/handle/10945/34701/13Jun_MacCalman_Alexander.pdf?sequence=1. Any references to Chapters refer to the dissertation.

This appendix is a user manual of the Front-End Tool in the DesignCreator_32bit_Version.xlsm and the DesignCreator_64bit_Version.xlsm files used to run a genetic algorithm that creates a design. The purpose of the tool is to allow the user to create a custom design, with a specified number of design points and number of factors, by type, number of levels with the desired balance, and the model terms included in the regression matrix. In addition, the user can start the algorithm with an existing design and add columns to it; this allows us to leverage the cataloged 2nd Order NOLH designs that are included in the workbook by adding columns to them. Once the algorithm creates the design, there are some utilities available that will create a spreadsheet to translate a design, create higher-order terms, calculate the maximum absolute pairwise correlation, and create dummy variables for categorical factors.

The algorithm was written in Java™ 2 and requires the user to ensure that the Java Platform (JDK) is downloaded on their computer; visit the Oracle website at <http://www.oracle.com/technetwork/java/javase/downloads/index.html> to download. You can download the tool from the SEED Center website at <http://harvest.nps.edu/software.html>. Once downloaded, there will be two files: DesignCreator.xlsm (containing the Front-End Tool with utilities) and DOE.jar (the executable .jar file written in Java). Ensure that these files are saved to the same folder. If you are on a shared network computer we do not recommend that you save the files to the desktop. When opening the DesignCreator.xlsm file, the user must enable the macros in order to utilize the buttons throughout the workbook. The Front-End Tool will create an input.csv file and a runit.bat file (for Windows computers) or runit.txt file (for Macintosh computers) and save them to the same folder; these are the files the DOE.jar file needs to execute the algorithm from the Windows computer Command line or the Macintosh computer Terminal window.

Once the algorithm is complete, the output design will be saved as a .csv file in the same folder the DOE.jar file is in. The output file title name will have the number of rows, columns, the ρ_{map} , ML_2 , and the initial seed used for the random number generator (see Chapter II for the definition of ρ_{map} and ML_2). In the .csv file, the first four rows will contain the following, respectively: the factor type, the number of levels, the model terms included in the regression matrix, and the factor name, x_i , where i is the column number. If there are discrete or categorical factors in the design, the last row, separated by the word “balance,” will have the factor’s balance metric indicating the spread of the levels across the design points; see Chapter VI for the definition of balance. As a general rule, the user should never delete or change any of the worksheet names in the DesignCreator.xlsm file. Each section in this appendix describes the worksheets in the DesignCreator.xlsm file and provides instructions where appropriate.

readme

The *readme* worksheet provides the purpose of the tool, explains how to create designs and use the utilities. In addition, it references literature that pertains to the designs created by the genetic algorithm.


gpl

The worksheet describes the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 2.1 of the License or (at your option) any later version. This license ensures that the algorithm is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY, without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Front End

Input Parameter Settings

The *Front End* worksheet allows the user to enter the genetic algorithm input parameters. The blue-colored cells are the factor entry area used to specify the number of factors, by type, number of levels, and the model terms included in the regression matrix for the ρ_{map} calculation. The four types of factors are: continuous, discrete, categorical, and binary. For continuous factors, the number of levels must be equal to whatever is set as the “Number of Design Points” parameter in the green-colored entry area. For categorical and binary factors, only the main (linear) terms can be added to the regression matrix (model terms must be set to “M.”) Binary factors can only have the number of levels set to 2. Generally, the user should set the highest-order model terms in the first set of rows. The model term designations are the following: M for main effects; MQ for main and quadratic effects; MI for main and two-way interactions; and MQI for main, quadratics, and two-way interactions. The model terms order, from highest to lowest, are MQI, MI, MQ, and M. The model term designations significantly impact the algorithm run time. The Column labeled “Minimum Feasible Imbalance” shows the minimum analytically achievable imbalance for all discrete and categorical factors. To see these imbalance values, press the “Calculate Balance Feasibility” button. See the Balance Check worksheet for guidance on the number of design points needed to achieve a desired imbalance amount for a given number of levels. Figure B1 shows a snapshot of the factor entry area in the *Front End* worksheet.



Simulation
Experiments &
Efficient
Designs
Center for Data Farming
Naval Postgraduate School

Enter the desired factors in the blue area. Indicate their type, number of levels, and model terms. The algorithm creates one column and then adds the remaining columns one at a time until the design has the total number of desired factors. When adding an additional column, the algorithm will attempt to minimize the correlations between columns in a regression matrix. The Model Terms entry area to the right specifies the terms that will be included in the regression matrix for each column.

Note: Creating a full second-order regression matrix (MQI) for 12 continuous factors may take over 3 days to complete. We recommend that you use the cataloged 2nd Order NOLH designs in this workbook and augment additional factors with the Model Terms set to either M or MQ.

Number of Factors	Factor Type	Number of Levels	Model Terms	Minimum Feasible ImBalance
2	discrete	4	M	0.142857143
1	continuous	21	M	
1	categorical	3	M	0

Model Terms Key:
M: Main effects only
MI: Main and two-way interaction effects
MQ: Main and quadratic effects
MQI: Main, quadratic, and two-way interaction effects

Run Algorithm (Windows Computer Only)

For MAC computers, see instructions below.

Note: In order to run the algorithm you must have **Java Platform (JDK)** downloaded on your computer. If it is not already, click on the following Oracle website link to download it:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Figure B1. Factor entry area in the *Front End* worksheet.

The red-colored cells are the algorithm’s internal input parameters that will not be of interest to the general user of the design creator. Chapter IV discusses the experimental designs we performed to determine the appropriate input parameter settings for design searches. The user can change these input settings, if desired (see Chapter III for the algorithm steps and definitions of input parameters), and can restore the default settings by pressing the macro button underneath the red-colored cell area. Changing these internal input parameter settings will impact the algorithm’s performance and run-time length; see Chapter IV for guidance on the performance and run-time length for different number of design points and columns with the default internal parameter settings. For designs that are not difficult to minimize the ρ_{map} , we recommend setting the number of trials (*numTrials*) equal to 1 in order to speed up the algorithm’s run time.

The green-colored cells are the input parameters the general users will need to set each time they run the algorithm. Because the algorithm is run as a batch file from the Command or Terminal window, the user may decide to increase the number of algorithm instantiations that will be executed. Setting the “Number of Batch Replications” parameter to greater than 1 will allow the user to send a batch file to a computer cluster to perform multiple replications of the algorithm. Because of the stochastic nature of the algorithm, we recommend performing multiple replications when searching for efficient designs and then selecting the design with the smallest ρ_{map} . If the user does not intend to send a batch file to a computer cluster, he/she can run the algorithm multiple times in separate Command/Terminal windows. The “Number of Design Points” parameter is the number of experiments or rows in the desired output design matrix. The “Start With Design” boolean parameter lets the algorithm know whether to add the desired factors entered in the blue-colored cell area to an existing design located in the *Start Design* worksheet. When the “Start With Design” parameter is set to TRUE, ensure that the “Number of Design Points” parameter is set to the same number of rows in the design that is pasted into the *Start Design* worksheet. The “Jiggle Operations” boolean parameter lets the algorithm know whether to perform the jiggle operations on the continuous factors (see Chapter III for a description of the jiggle operations). If the algorithm starts with an existing design, the jiggle operation will only be performed on the newly added continuous columns. The “Show Comments” boolean parameter lets the algorithm know whether

to show the comments in the Command/Terminal window during the algorithm's execution. When sending a batch file to a computer cluster, the "Show Comments" parameter should be set to FALSE. Figure B2 shows a snapshot of the input parameter entry area in the *Front End* worksheet.

Input Parameter	Setting	Description
Number of Batch Replications	1	The number of command line batch replications written to the batch file.
Number of Design Points	20	The number of rows in the design matrix. Each row designates the factor settings for each experiment.
Start With Design	FALSE	TRUE means that the algorithm will add columns to the design that is pasted into the Start Design worksheet. FALSE means that the algorithm will create a new design.
Perform Jiggle Operations	TRUE	TRUE means that the algorithm will perform the jiggle operation, FALSE means that it will not. The jiggle operation will not be performed on columns in the Start Design worksheet.
Show Comments	TRUE	TRUE means that the algorithm comments will be displayed in the command/terminal window. Set to FALSE when sending batch files to a high performance computer cluster.
<i>numExploreGen</i>	100	Number of exploration generations.
<i>numExploitGen</i>	200	Number of exploitation generations.
<i>popSize</i>	100	Size of the population of candidate columns.
<i>copyPortion</i>	0.1	Portion of candidate columns copy into the next generation.
<i>halfWidth</i>	0.5	The bounded distance that prevents the jiggle operator for perturbing outside a range.
<i>numJigGen</i>	100	Number of jiggle generations.
<i>numTrials</i>	3	Number of exploration trials each consisting of a set of exploration generations with its own initial population of candidate columns.
<i>swapPortion</i>	0.2	Portion of design points swapped during a swap operation.
<i>poolSize</i>	100	Size of the pool that contains a set of candidate columns.
<i>genExitCriteria</i>	20	Number of generations performed without improvement of the fitness function.
<i>jigglePortion</i>	0.2	Portion of design point jigged during a jiggle operation.
<i>colAttempts</i>	3	Number of attempts to find a column with a new initial population of solutions if an attempt did not meet the maximum correlation threshold.
<i>jigglePasses</i>	3	Number of times the jiggle operator is performed on the columns.
<i>corrThreshold</i>	0.05	The maximum correlation a column threshold must be before added to the design. The algorithm will continue to find a column to add to the design for a set number of attempts (<i>colAttempts</i>).

Figure B2. Input parameter entry area in the *Front End* worksheet.

Algorithm Execution

Once the input parameters are set, the steps to execute the algorithm will depend on the type of operating system on your computer (Windows or Macintosh). For Windows computers, simply press the "Run Algorithm" macro button; each time you press this button, a new Command line window will open and run a different instantiation of the algorithm. Macintosh computers must run the algorithm from the Terminal window, with the current directory set to the file location where the DesignCreator.xlsm and DOE.jar files are saved. The first step is to press the "Create Flat Files" macros button. Then, open the Terminal window and change the directory to where the algorithm is saved. At the Terminal Command prompt, type the following:

```
./runit.txt
```

To run additional algorithm instantiations simultaneously, open a new Terminal window and repeat the above steps. To open the Terminal window from the Finder, the user can go to System Preferences and click on "Keyboard," select the "Keyboard Shortcuts" tab and click "Services" from the left menu; scroll down on the right and check the box next to "New Terminal at Folder." Setting this preference will allow the user to right click on a folder in the Finder and click "New Terminal at Folder" to open the Terminal at the desired folder. This preference setting will save the user from having to change the directory manually to where the algorithm is located each time you open the Terminal window.

When the "Show Comments" parameter is set to TRUE, the comments shown in the Command or Terminal window reveal the progress of the algorithm. Figure A3 shows a Command line window that searched for a three continuous factor 2nd order design with 20 design points. The algorithm performed three exploration trials (*numExploreGen* = 3) and three jiggle generation passes (*jigglePasses* = 3). The final time shown at the bottom of Figure B3 is in hours.

```

Y:\trunk\Disertation\FrontEnd>java -jar D0E.jar 20 False True True 100 200 100
0.1 0.5 100 3 0.2 100 20 0.2 3 3 0.05
design Points: 20 seed: 2149891

1 column, continuous factor type, 20 discreteLevels, mode: MQI, 1 columnAttempt,
designSize: 1
1 exploration trial correlation: 0.0030075187969924814
2 exploration trial correlation: 0.0035120253120068706
3 exploration trial correlation: 0.0038094149848396284
best from exploration generations: 0.0030075187969924814 time elapsed: 0.0661632
0333333334 minutes
best from exploitation generations: 0.0030075187969924814

2 column, continuous factor type, 20 discreteLevels, mode: MQI, 1 columnAttempt,
designSize: 2
1 exploration trial correlation: 0.04602415128730918
2 exploration trial correlation: 0.02857142857142857
3 exploration trial correlation: 0.0298522151520584
best from exploration generations: 0.02857142857142857 time elapsed: 0.209332816
66666667 minutes
best from exploitation generations: 0.02857142857142857

1 jiggle pass:
2 jiggled column: 0.013533527345797927 original column: 0.02857142857142857
1 jiggled column: 0.00662377093987147 original column: 0.013533527345797927
0 jiggled column: 0.005520766537884698 original column: 0.013516121050120719

2 jiggle pass:
2 jiggled column: 0.005520766537884698 original column: 0.005520766537884698
1 jiggled column: 0.003323758202155466 original column: 0.006572470707733691
0 jiggled column: 0.003943008496000514 original column: 0.005520766537884698

3 jiggle pass:
2 jiggled column: 0.00394300849600049 original column: 0.00394300849600049
1 jiggled column: 0.002658994167946492 original column: 0.00394300849600049
0 jiggled column: 0.0034261573867559723 original column: 0.005472171446045812

design size: 3, maximum correlation: 0.003426157386755961, time: 0.0191719396666
6667, ML2: 0.006221993446700047, seed: 2149891,
Y:\trunk\Disertation\FrontEnd>

```

Figure B3. Command line window during the algorithm execution.

Balance Check

This worksheet calculates the minimum analytically achievable imbalance for a given number of discrete or categorical factor levels. Use this worksheet to help decide how many design points you need to ensure the design's imbalance is minimized. The algorithms will attempt to find discrete or categorical factors with an imbalance < 0.1 . After 50 attempts, if the algorithm did not find a column, then it will return the column with the lowest imbalance. There are some design point and level combinations that cannot achieve a 0.1 balance. This worksheet will help guide which design points are feasible for a given number of factor levels.

Cataloged Designs

This worksheet has hyperlinks that will navigate the user to other worksheets that contain the cataloged 2nd Order NOLH design. Once there, the user can press the macro button to automatically copy the design into the *Start Design* worksheet. We recommend using these cataloged designs for up to 12 continuous factors when you can afford to perform the number of experiments needed for each design. When the user desires to add discrete factors to a set of continuous factors (up to 12), with the model terms set to “MQI” (for a full second-order model), we recommend copying a cataloged design to the *Start Design* worksheet and then deleting two continuous columns for every one discrete factor (this is only a rule of thumb). Adding additional columns (of any type) to the cataloged designs, with the model terms set to “M” or “MQ” do not require that you delete continuous columns.

Start Design

If the user desires to add additional columns to an existing design, paste the design into this worksheet and set the “Start Design” parameter to TRUE in the *Front End* worksheet. The first row designates the factor type. Ensure one of the following text entries is in each column in the first row: continuous, discrete, binary, or categorical. Specify the number of levels for the factor in the second row. For continuous factors, the algorithm does not care what is entered because the number of levels for a continuous factor is always the number of design points. The third row contains the model terms (M, MI, MQ, and MQI). These entries have no impact to the algorithm. The fourth row is reserved for the factor name. Ensure that the design (with the first four rows) is pasted into cell B1.

Coded Design

Paste a design with the first four row entries as indicated in the *Start Design* worksheet instructions into cell B1. If there are discrete or categorical factors in the original .csv output file, be sure not to paste the word “balance” and the balance metric into this worksheet. Also, avoid pasting empty cells that may get highlighted after selecting the current region in the .csv output file. Press the “Create Translation Worksheet” macro button to create a formula worksheet that will allow the user to translate the coded design point levels to the factors range desired for the experiments. To calculate the ML_2 and ρ_{map} metrics, press the “Insert Design into Design Tools Worksheet” macro button. If the design has categorical factors and the user wants to examine the first-order correlations of the design with the categorical dummy variables, press the “Insert Design into Categorical Design Worksheet.”

Translated Design

After pressing the “Create Translation Design” macro button in the *Coded Design* worksheet, the macro will insert the formulas into the cells that will allow the user to translate the design to the desired factor ranges. The blue-colored cells are copies of the first three rows from the *Coded Design* worksheet (factor type, number of levels, and model terms). For continuous factors, enter the low and high setting for each factor. Users have the option to round the continuous factor to a discrete factor; however, we do not recommend doing this. Rounding a continuous factor is an old technique to create discrete factors but can severely impact the ρ_{map} of the original design (especially the 2nd Order ρ_{map}). We should not have to round a continuous factor anymore because our algorithm is capable of creating designs with discrete factors for a specified number of levels. If the factor column is discrete, the sixth row allows the user to scale the column instead of rounding. Scaling a discrete factor to a number greater than 1 will spread the discrete levels over a wider range of values. If the factor type is either discrete or categorical, the high level will be protected and will add the number of levels to the low-level setting. The yellow-colored cells are protected to ensure the user does not change the translation formulas. After establishing the low and high levels and naming the factors, the user can copy and paste special values the translated design into another spreadsheet for their experiment.

Design Tools

After pressing the “Insert Design into Design Tools Worksheet” macro button in the *Coded Design* worksheet, the design will appear (with the factor names only in the first row) in cell B1. The available macro buttons allow the user to calculate the ML_2 space-filling metric; center the design by subtracting the mean; create the quadratic terms; the second-, third-, and fourth-order terms; calculate the ρ_{map} , and calculate the distribution of all absolute pairwise correlations. Before you create the higher-order terms, you must ensure that you center the design first; otherwise, the main factors will be highly correlated with its own quadratic. Be sure to only press the higher-order macros button once; otherwise, the macro will expand out the terms with whatever is currently in the worksheet. Delete the

high-order terms in the worksheet if you desire to recreate a different set of higher-order terms. When the user presses the “Collect and Sort Abs Corr Distribution” macro button, the distribution of all absolute pairwise correlations of whatever design is currently in the worksheet will get pasted and sorted into the *Abs Corr Distro* worksheet.

Abs Corr Distro

After pressing the “Collect and Sort Abs Corr Distribution” macro button, the absolute pairwise correlation distribution will get pasted and sorted into this worksheet.

Categorical Design

After pressing the “Insert Design into Categorical Design Worksheet” macro button in the *Coded Design* worksheet, the design will appear in cell B1. From here, the user can designate the dummy variable convention before creating the dummy variables (see Chapter VI for a description of the different dummy variable conventions). After pressing the “Create Dummies” macro button, a new design will get pasted into the *Dummy Variables* worksheet with all the categorical factors converted into the set of dummy variables determined by the number of levels.

Dummy Variables

This worksheet will contain the design with dummy variables after pressing the “Create Dummies” macro button in the *Categorical Design* worksheet. Pressing the “Find First-Order Correlation Distro with Dummy Variables” macro button will paste and sort the absolute pairwise correlation distribution into the *Abs Dummy Corr Distro* worksheet.

Appendix C: JMP Dashboard Building Instructions

The appendix outlines the details on how to use and build the dashboard in JMP. The first section provides a brief introduction to the dashboard and how it is used. The second section walks through each step needed to build the dashboard. There are two videos that accompany these instructions split into two parts. Below are the links to each part; if prompted for a password, use “dashboard.”

Part 1

<http://adsurgo.adobeconnect.com/p5a1vt8zd7f/>

Part 2

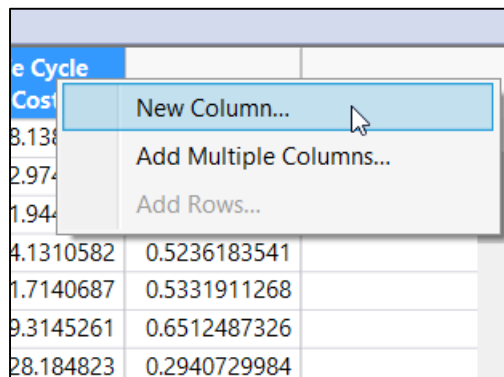
<http://adsurgo.adobeconnect.com/p857bf84cxe/>

Dashboard Introduction

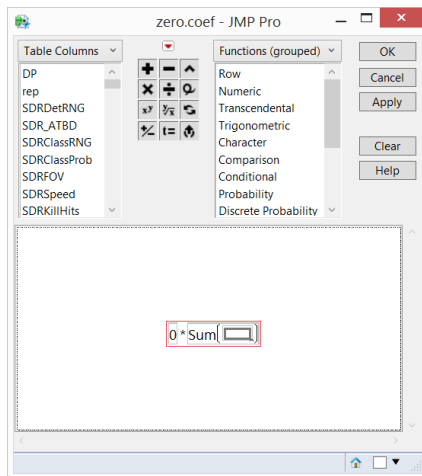
1. Run the application (use the correct data table with the added columns). This example uses the file MeanATK_Sample4_expanded_formulas.jmp
2. The dashboard appears in three windows: factor settings, contour dashboard, and prediction profiler dashboard.

Expand the column formulas to depend on all of the inputs

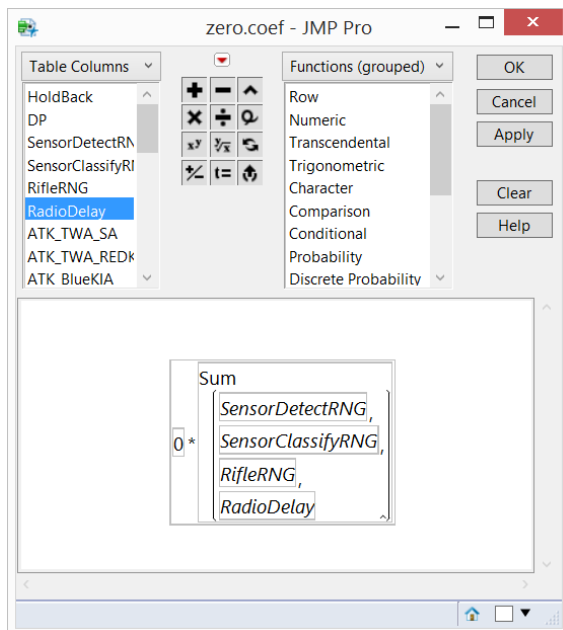
1. Open the source JMP data table. In this example, we use MeanATK_Sample4.jmp.
2. Right click on a new column header (at the far-right side of the data table) and click **New Column**.



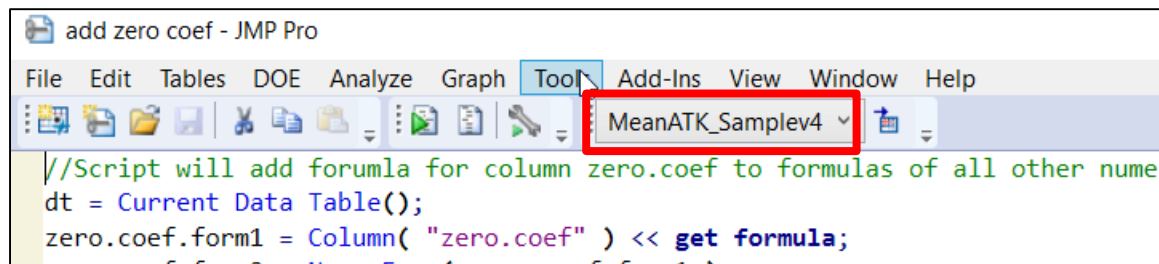
3. Enter “zero.coef” for the name and click OK.
4. Right click on column header “zero. coef” and click on **Formula** to open the formula editor for column zero.coef.
5. Click the “no formula” box and type 0*sum().
6. Press Enter




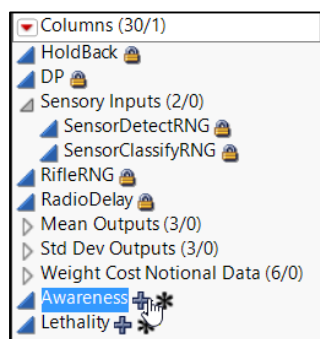
7. Click on the box inside the parenthesis in order to enter arguments to Sum.
8. Select all of the table columns that are used as predictors. In the “Table Columns” pane, click and release *SensorDetectRNG*. Scroll down, hold *Shift* and click *RadioDelay*. This will also include all of the columns that lie in-between.



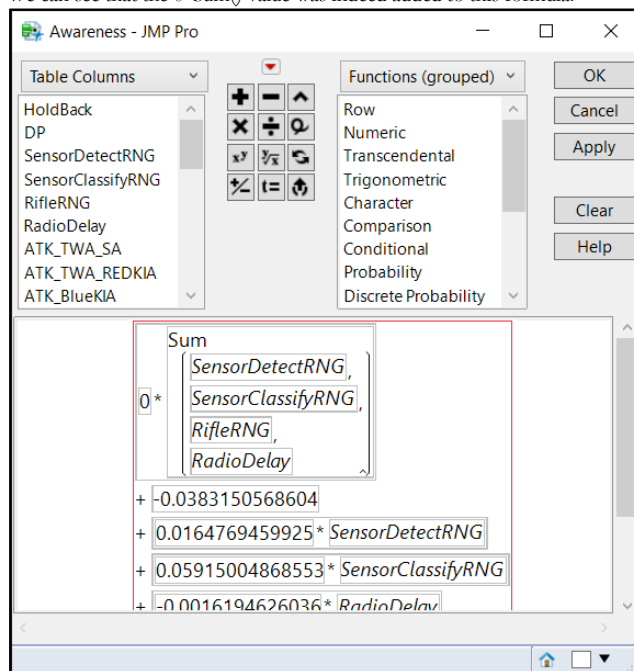
9. Click OK.
10. Open and run “add zero coef.jsl”. This script will add the newly created formula to the formulas of the continuous columns. This will allow the contour profilers to display correctly, since the contour profiler axes can only use factors upon which at least one of the responses depends. Note: Make sure that *MeanATK_Samplev4* is the current data table before running this script (it will be as long as it is the only table open, or if it the last table you were working with).



11. You can verify that the script worked by looking at one of the formulas of the response columns. For example, click the  sign next to *Awareness* in the **Columns** pane on the left side of the data table window to view the formula.



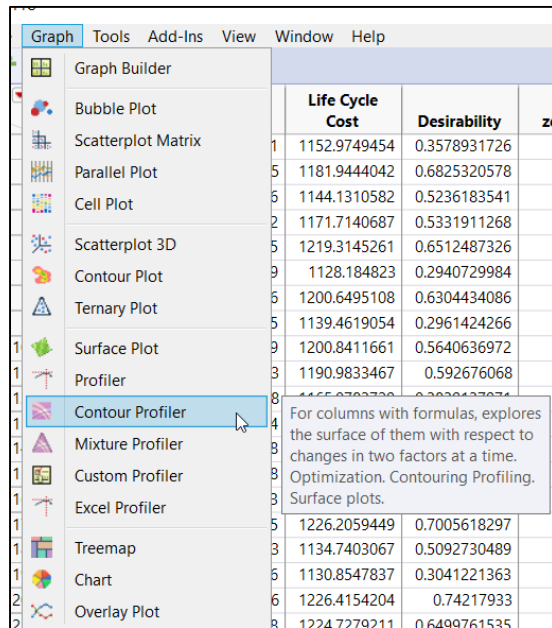
We can see that the $0 * \text{Sum}()$ value was indeed added to this formula.



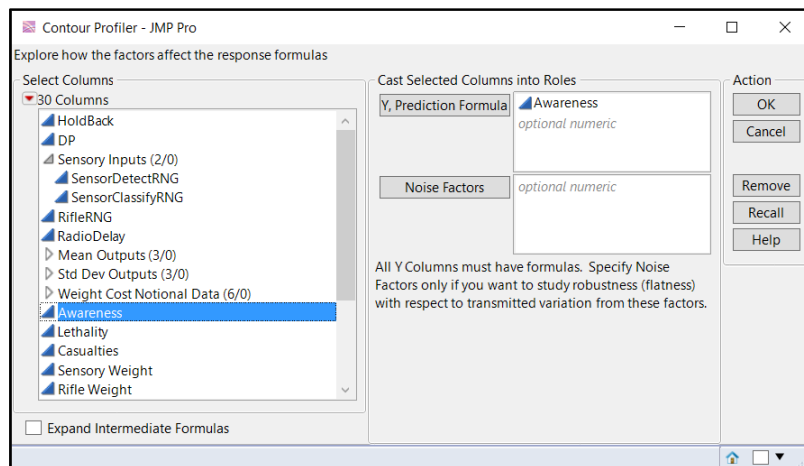
12. Close the open formula editor from the previous step.
 13. Save the JMP file as "MeanATK_SampleV4_expanded_formulas.jmp".

Build the Contour Profiler Dashboard

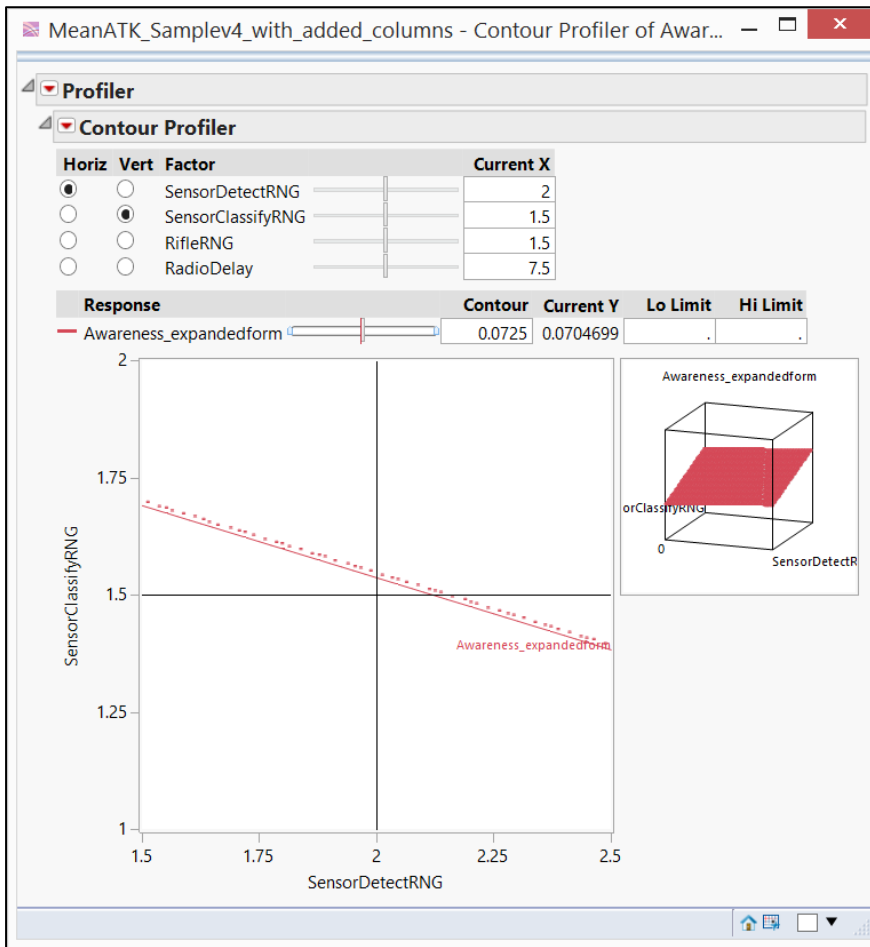
1. Create a Contour profiler using any one of the responses that will be used for its factor setting sliders.
Here, we will use the *Awareness* column. Select **Graph > Contour Profiler** (from the top JMP menu bar).



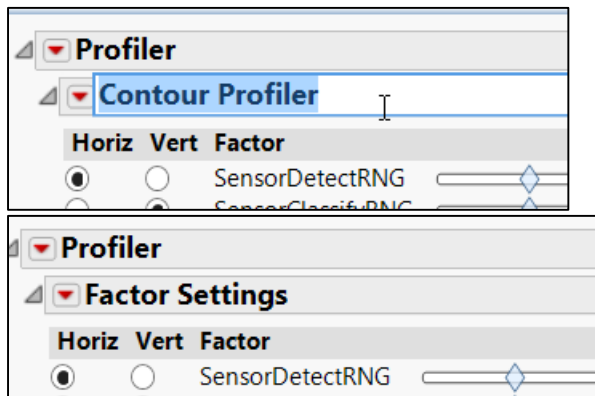
2. Select *Awareness*.
3. Click **Y, Prediction Formula**.



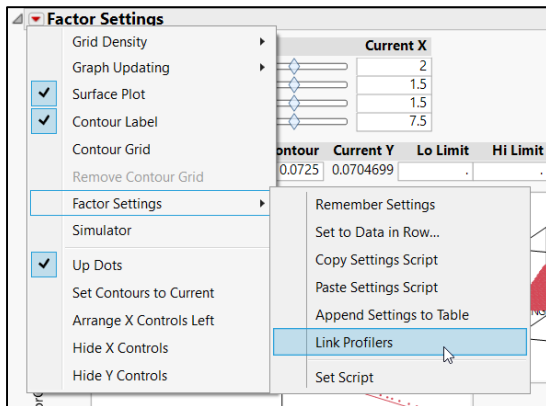
4. Click **OK**.



5. Double click the "Contour Profiler" option box title and change the name to "Factor Settings."

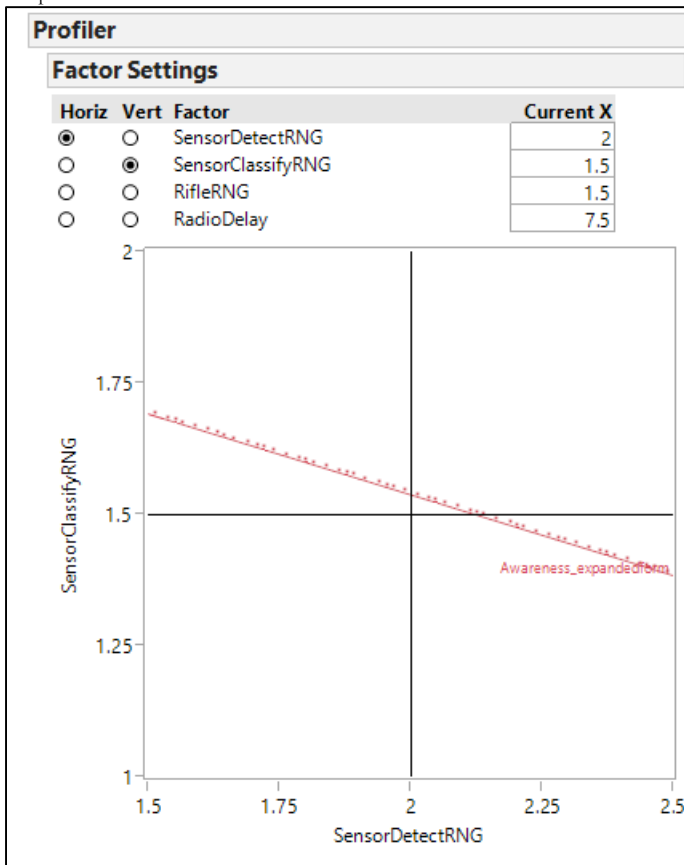


6. Click the red triangle that appears next to Factor Settings in order to select the following options. Leave the other options in the Factor Settings menu as they are if they are not mentioned here.
 - a. Select **Factor Settings > Link Profilers**



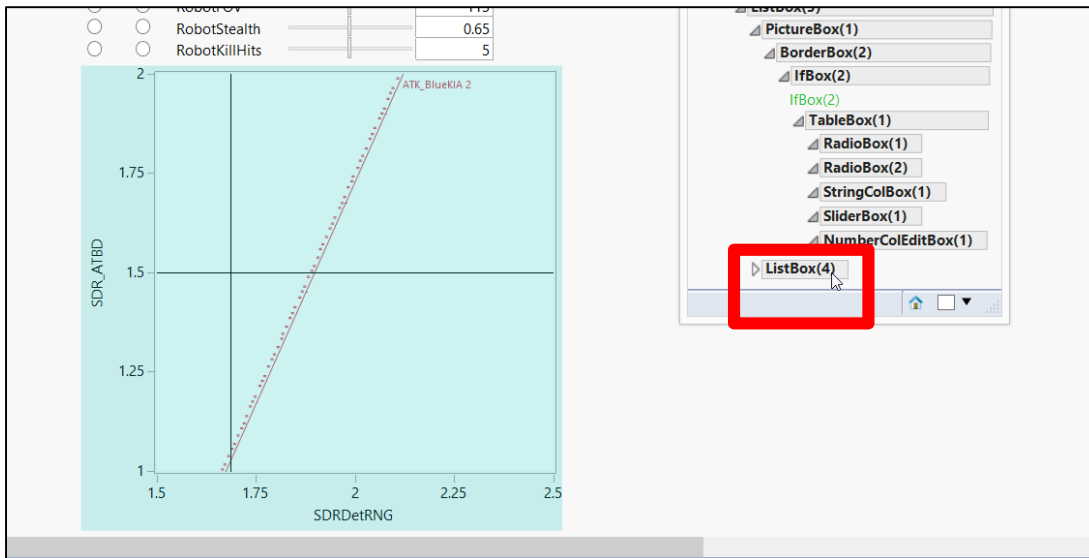
- b. De-select **Surface Plot**.
- c. De-select **Arrange X-controls left** (may already be de-selected).
- d. Select **Hide Y Controls**

The contour plot (the graph at the bottom of the window) is removed with a line of code inserted at a later step. Only the factor setting sliders from this particular window are needed.



In order to see which area in the display contains the unnecessary graph, right-click the grey triangle next to Factor Settings and select **Edit > Show Tree Structure**.





Hovering the mouse over ListBox(4) reveals that this is the name of the graph that needs to be deleted (leaving only the factor settings that appear above the graph). This is already hard-coded below, and nothing needs to be done here. The purpose of this step is to explain why ListBox(4) appears in the code below.

7. Close the “Show Tree Structure” window, but leave the contour profiler window open. The contour profiler must remain open in order to add it to the application builder later.

Next, create Contour Profiles for each of the groups of responses (survivability, lethality, casualties, and weight). The groups that the manual uses for the MeanATK data set are defined in the next step. Three of these groups consist of a single response, while three responses are grouped in the Weight group. When developing a dashboard for your own application, you may place the responses into as many or as few groups as you think makes sense. At one extreme, you could produce a contour profiler for each response, making each response its own group. At the other extreme, you could place all of the responses into a single contour profiler. However, placing all of the responses in a single profiler will make it more difficult to see how changes in the factor settings impact the responses. Going forward, the manual will make use of the four groups defined in the next step: if you change the number of groups, there are a few places in the JSL code embedded later in the manual where you will need to change the upper index of a “for-loop” from 4 to the number of contour profilers you have chosen: these instances are highlighted.

8. In this data table, there are 4 different groups we will use. Note that 3 of these groups contain only a single response. Also note that the different Cost columns and the Desirability column are not currently used anywhere in this manual.
 - a. Awareness
 - i. *Awareness*.
 - b. Lethality
 - i. *Lethality*
 - c. Casualties
 - i. *Casualties*
 - d. Weight
 - i. *Sensory Weight*
 - ii. *Rifle Weight*
 - iii. *Radio Weight*

9. Click **Graph > Contour Profiler**.

10. Select *Awareness* for **Y, Prediction Formula**. We already used *Awareness* for the previous contour profiler we built: that was in order to harvest the factor settings sliders. We will leave that existing

profiler open, and will format this new one in a different way. From the new window we create here, we will be using the contour plot in the dashboard. See the next two screenshots.

Profiler								
Factor Settings								
Horiz	Vert	Group	Factor		Min	Current X	Max	
<input checked="" type="radio"/>	<input type="radio"/>	Sensory	SensorDetectRNG		1.5	2	2.5	
<input type="radio"/>	<input checked="" type="radio"/>		SensorClassifyRNG		1	1.5	2	
<input type="radio"/>	<input type="radio"/>	Weapon	RifleRNG		1	1.5	2	
<input type="radio"/>	<input type="radio"/>	Como	RadioDelay		0	7.5	15	

Figure 1 Screenshot from final dashboard. These factor settings were produced from the first Awareness Contour Profiler.

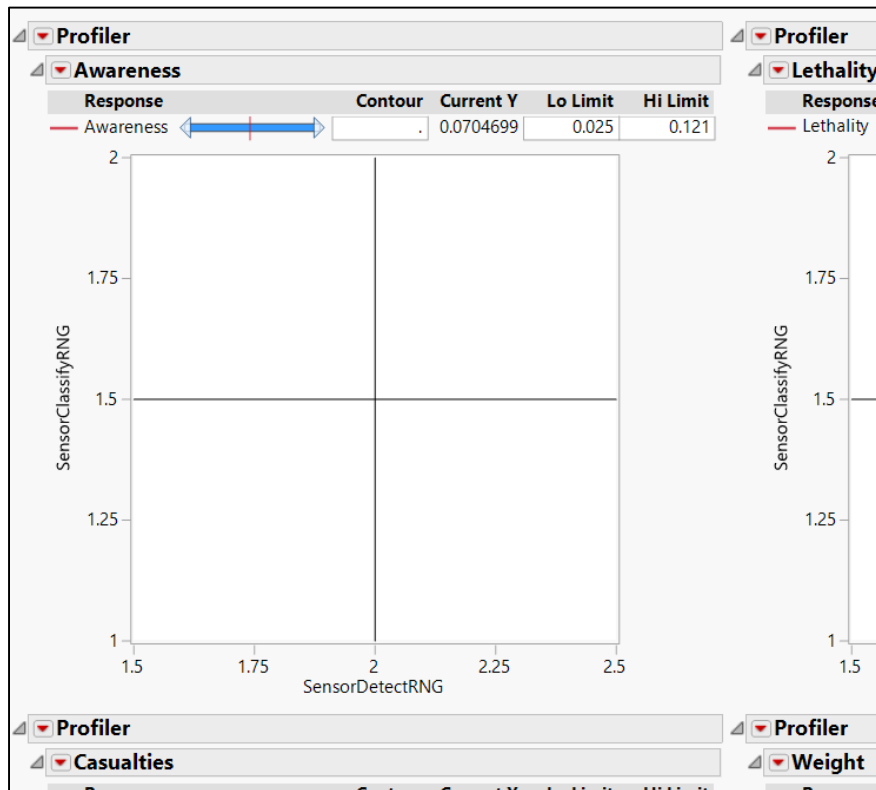
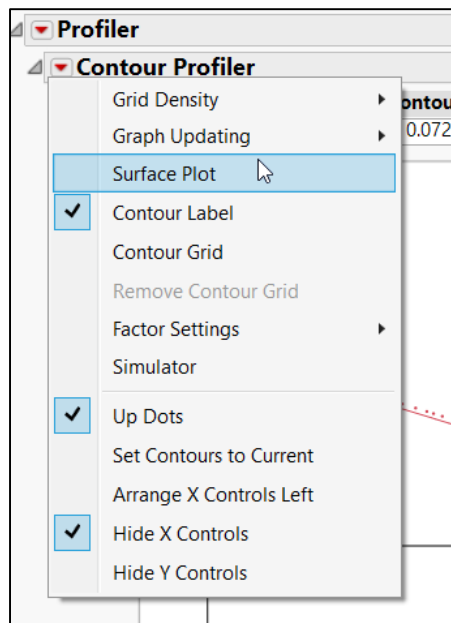
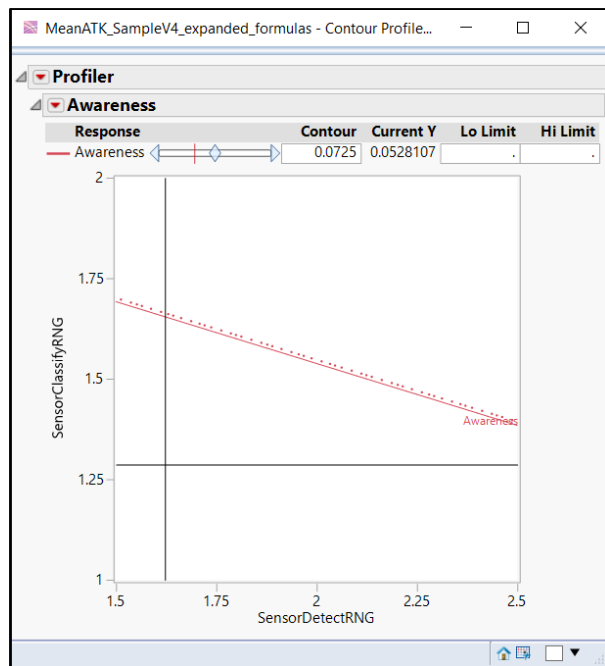


Figure 2 Screenshot from final dashboard. This plot was produced from the second Awareness Contour Profiler.

11. Click **OK**.
12. Click the red triangle next to **Contour Profiler** and
 - a. Select **Hide X Controls**
 - b. De-select **Surface Plot**



13. Double click the **Contour Profiler** option box title, and rename it to the group name, “Awareness”.



14. Leave this contour profiler open in the background.

15. Click **Graph > Contour Profiler**.

16. Select *Lethality* for **Y, Prediction Formula**.

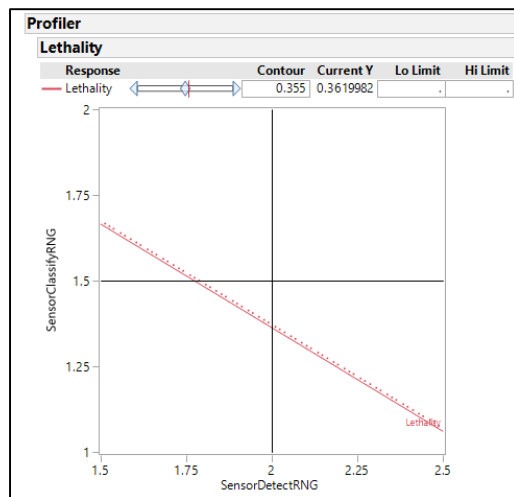
17. Click **OK**.

18. Click the red triangle next to **Contour Profiler** and

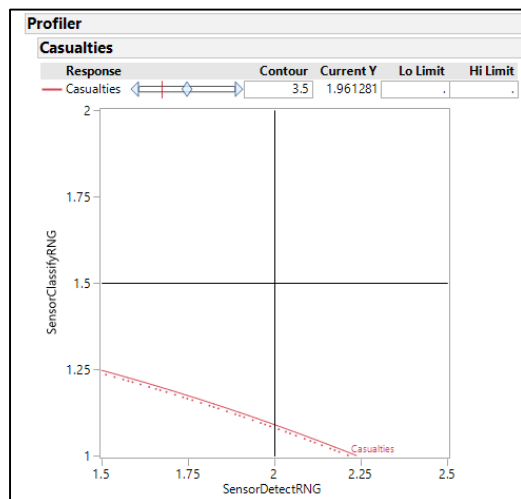
c. Select **Hide X Controls**

d. De-select **Surface Plot**

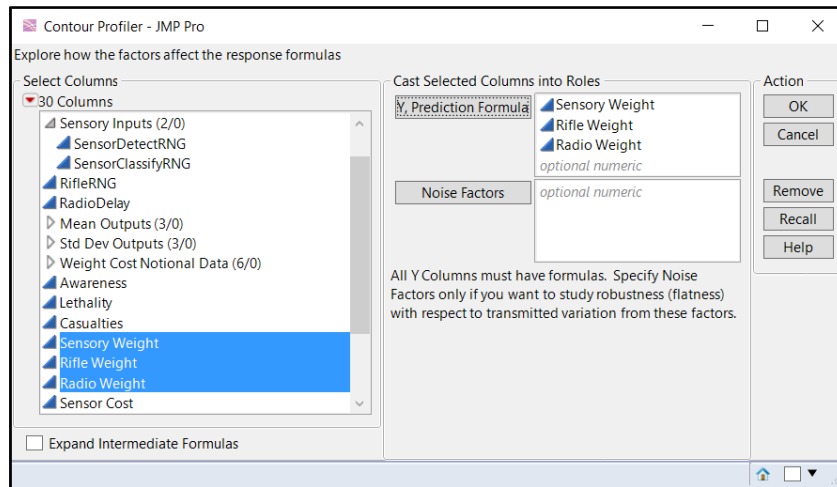
19. Double click the **Contour Profiler** option box title, and rename it to the group name, “Lethality”.



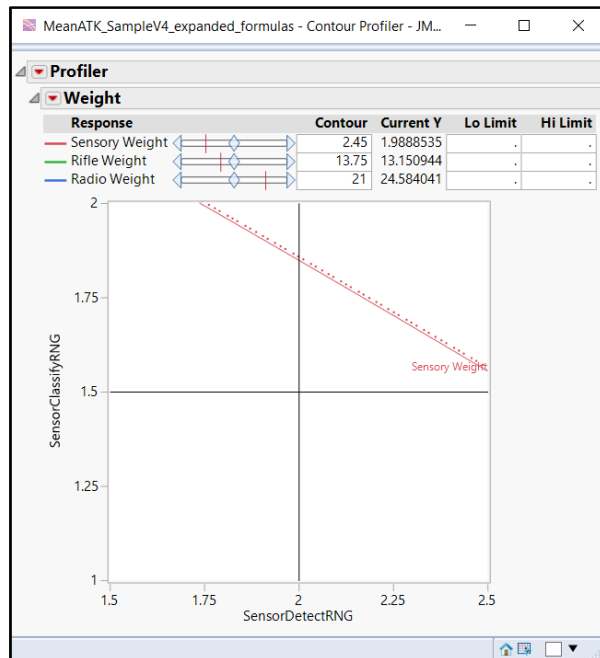
20. Leave this contour profiler open in the background.
21. Click **Graph > Contour Profiler**.
22. Select *Casualties* for **Y, Prediction Formula**.
23. Click **OK**.
24. Click the red triangle next to **Contour Profiler** and
 - e. Select **Hide X Controls**
 - f. De-select **Surface Plot**
25. Double click the **Contour Profiler** option box title, and rename it to the group name, "Casualties".



26. Leave this contour profiler open in the background.
27. Click **Graph > Contour Profiler**.
28. Select *Sensory Weight*, *Rifle Weight*, and *Radio Weight* for **Y, Prediction Formula**.

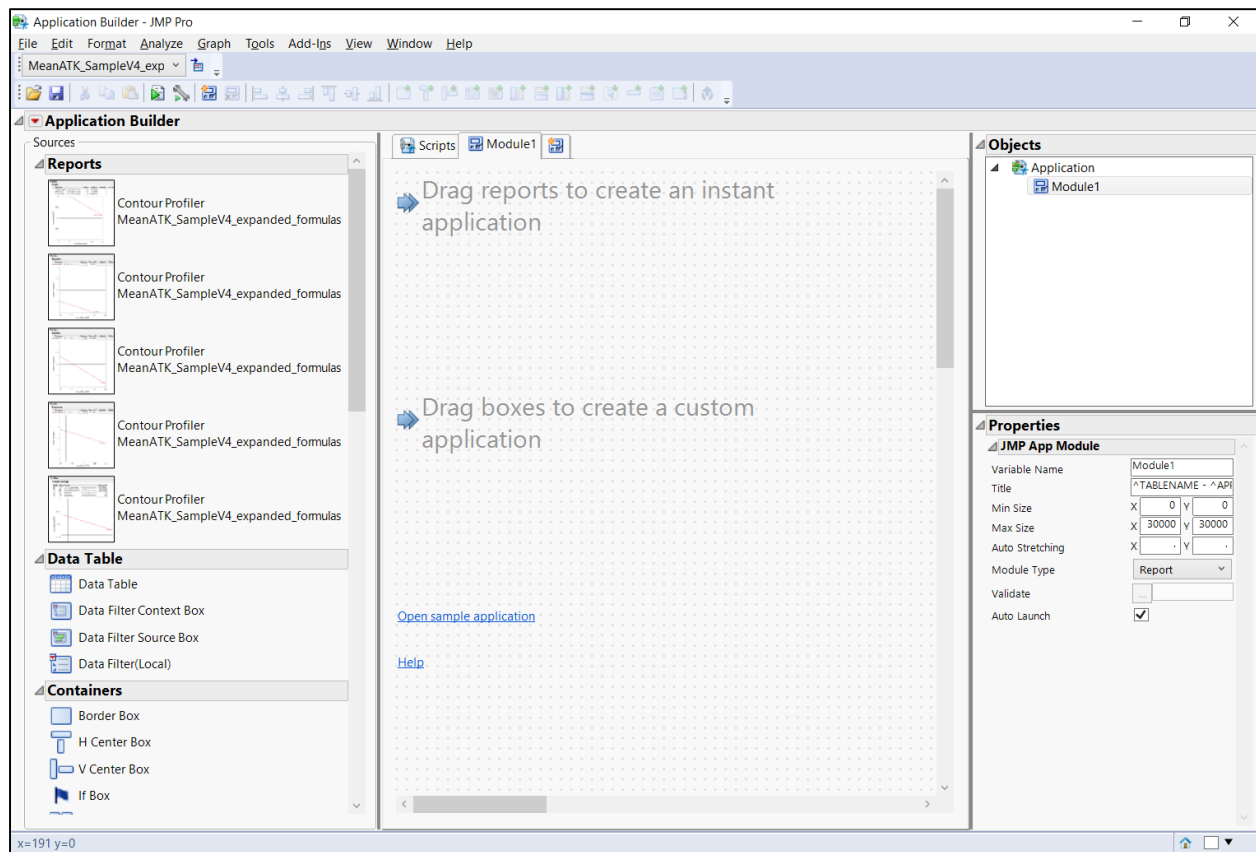


29. Click **OK**.
30. Click the red triangle next to **Contour Profiler** and
 - g. Select **Hide X Controls**
 - h. De-select **Surface Plot**
31. Double click the **Contour Profiler** option box title, and rename it to the group name, "Weight".
32. Leave this contour profiler open in the background.

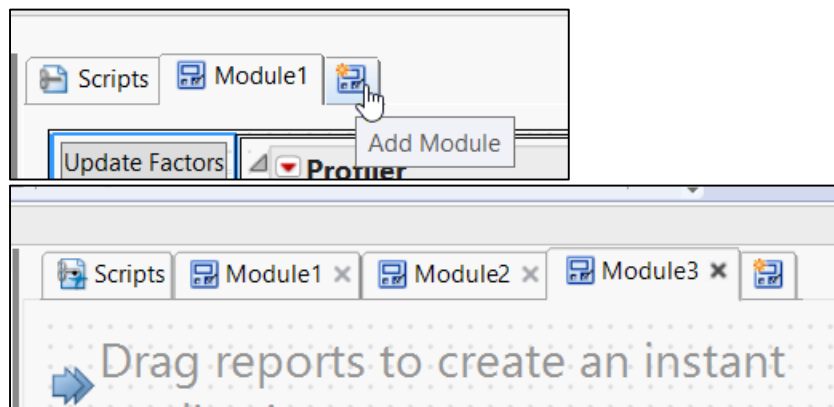


We will now begin to create the dashboard (in JMP, called an Application) and pull all of these contour profilers into the dashboard.

33. Select **File > New > Application** (from the top JMP menu bar, which may be hidden until the mouse scrolls over it) to open the application builder.

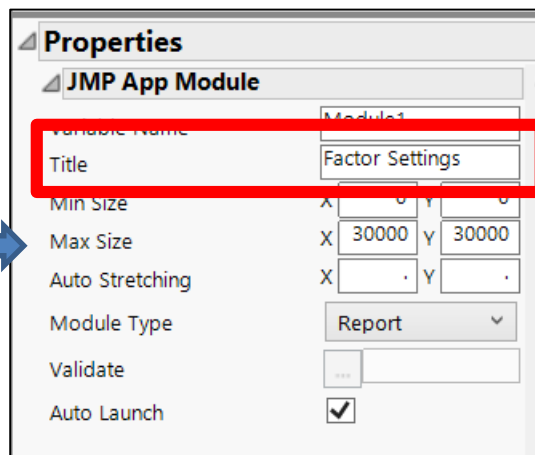
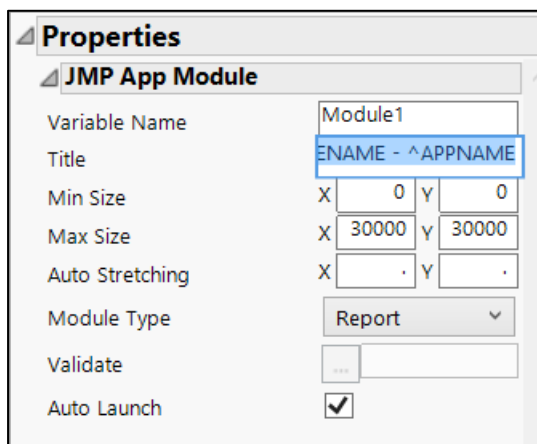
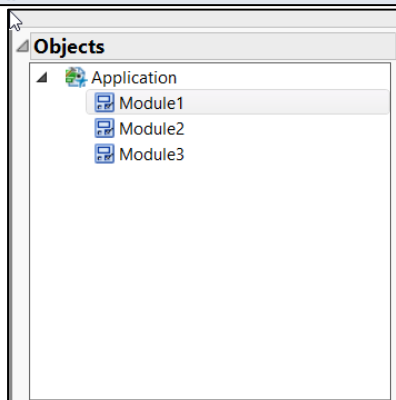
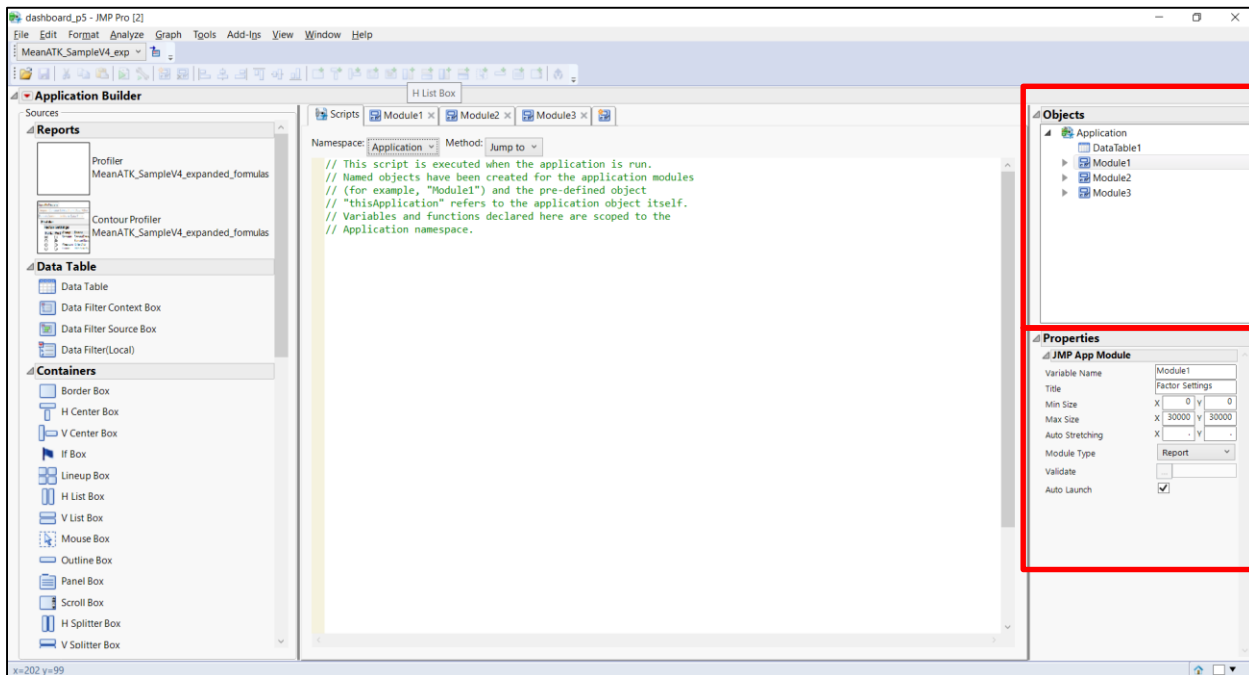


34. Click the “Add Module” tab (the icon next to “Module 1” in the top-middle of the screen) twice to create *Module2* and *Module3*.

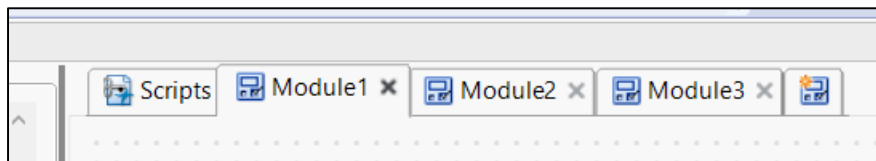


35. Select *Module1* from the Objects pane (which is located at the top right of the Application Builder window, see the screenshots below.) and change its Title to *Factor Settings* (in the properties pane on the bottom right of the Application Builder window, see the following screenshots). Note: the Objects Pane shows a hierarchical tree of all of the objects included in the Application Builder (dashboard). You can click on the objects here to select them or right-click on them to modify them (adding parent container objects, etc). In a couple cases later in this manual, it is easier to select some of the objects of interest in the Objects pane rather than in the center pane of the Application Builder (under the Module

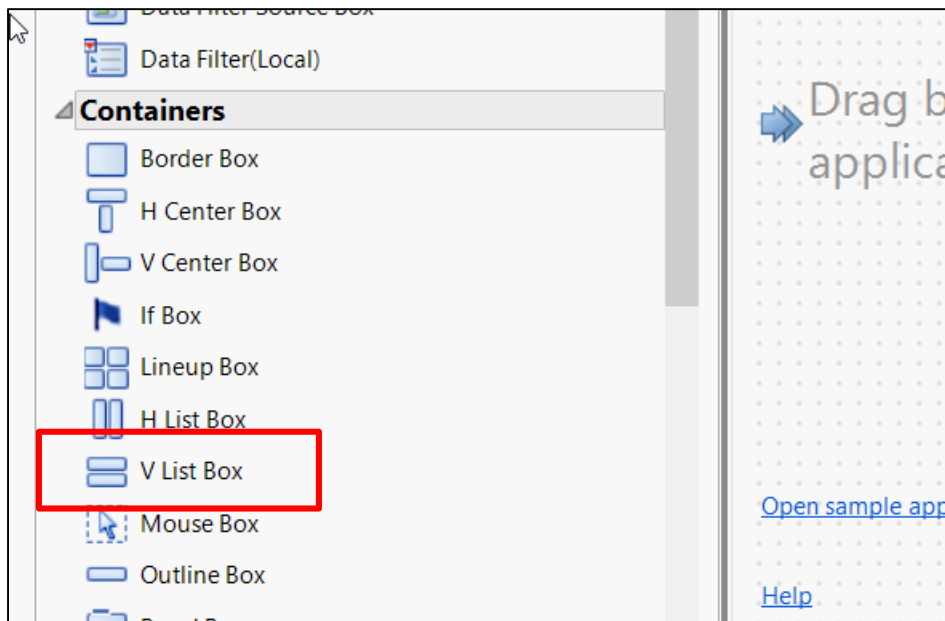
tabs) due to a GUI (graphical user interface) limitation of JMP in the center pane. These later cases include explicit instructions to use the Objects pane.

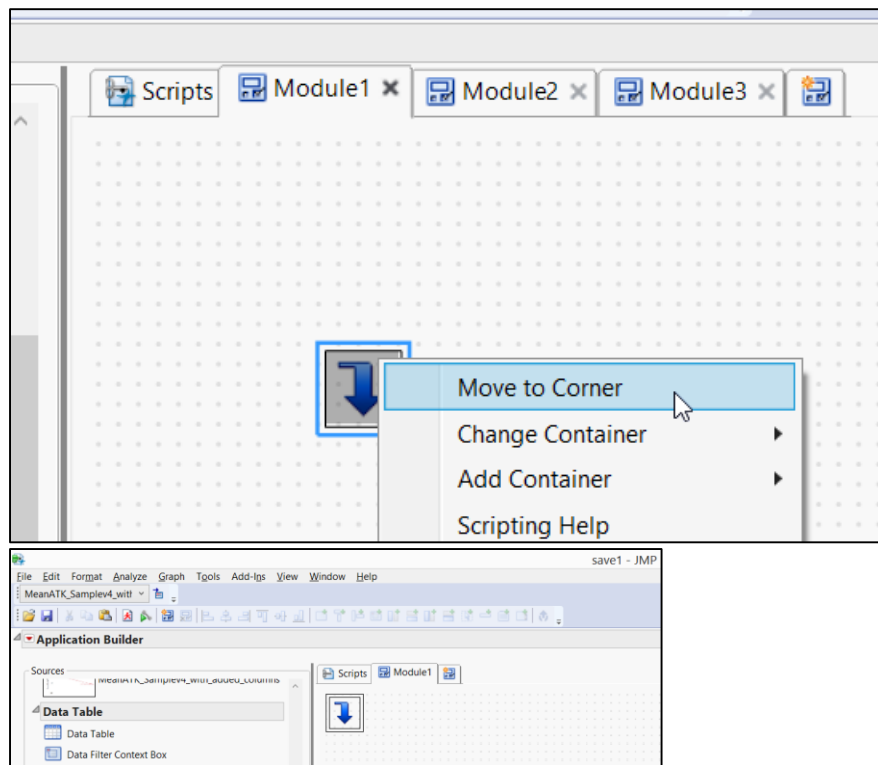


36. Likewise, select Module2 from the Objects pane and change its Title to *Contour Dashboard* in the Properties Pane.
37. Likewise, select Module3 from the Objects pane and change its Title to *Profiler Dashboard* in the Properties Pane.
38. Select the Factor Settings tab (which is still titled Module1. Note that throughout the instructions, these tabs are still labeled Module1, Module2, and Model3. These values are set in the Properties pane after selecting one of the Module tabs. Step 35 of this section changes the Titles of the modules, not the Variable Names. The titles are what appear in the window title bar at the top of the screen when the module opens, and the variable name is the keyword the JSL code uses to reference the module. While it makes sense in future applications to assign the modules more specific names, the manual keeps their default variable names throughout, so it's important to keep them as they are).

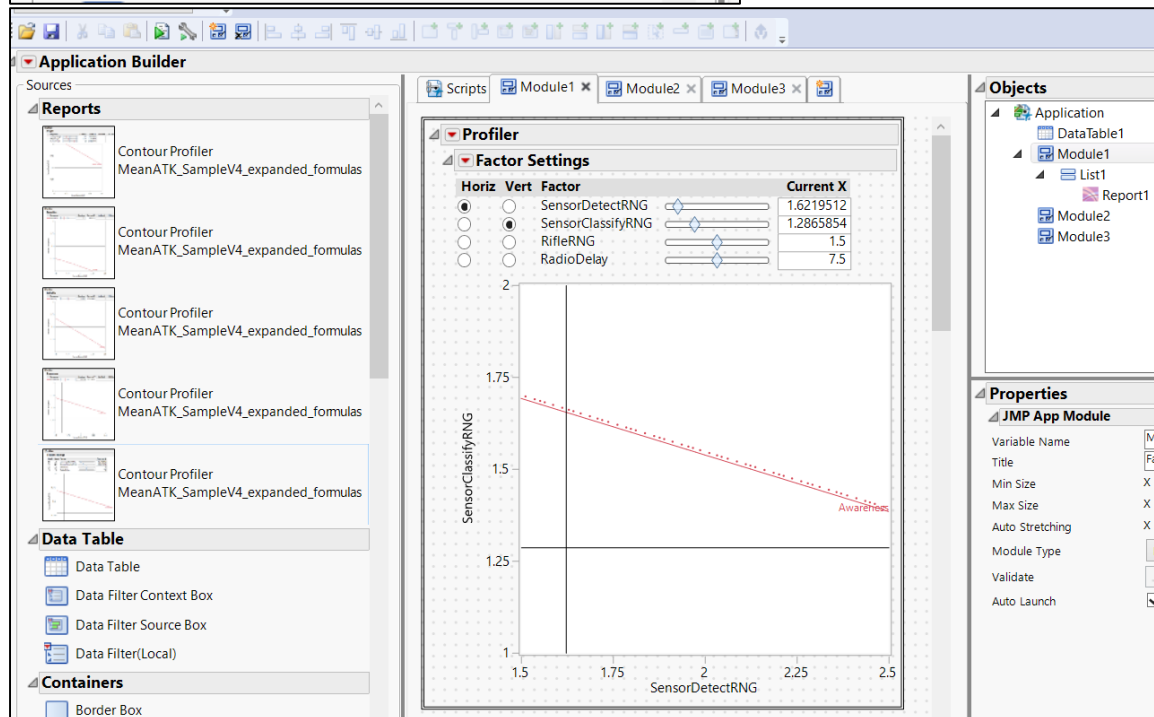
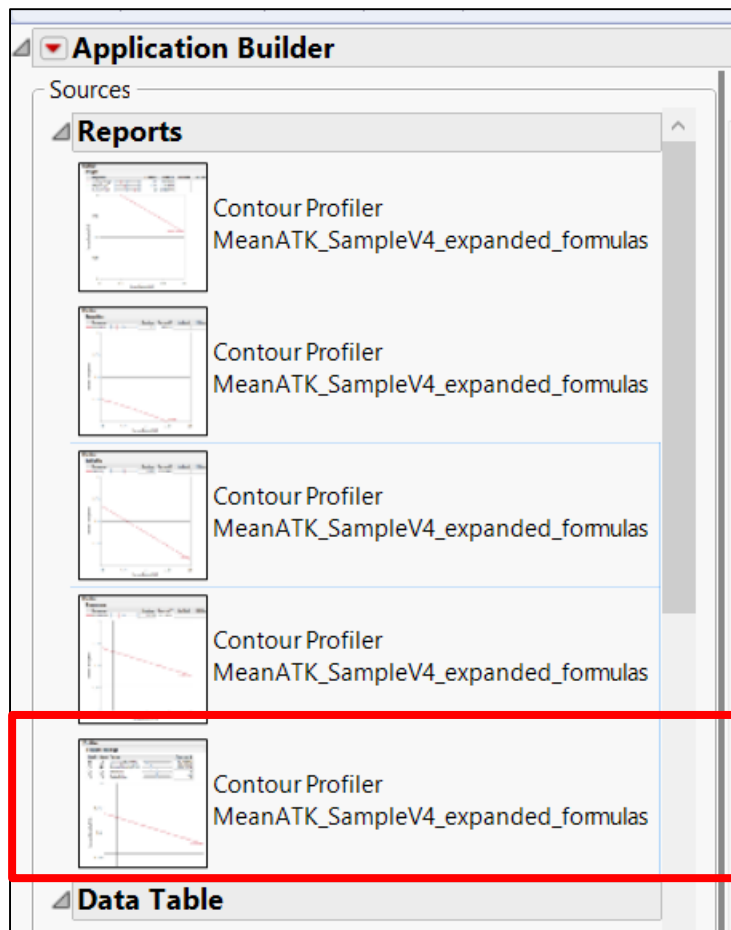


39. Drag a V list Box to the top left corner of Module1 (the Factor Settings module). (Just place it in Module 1, right-click and select **Move to Corner**).

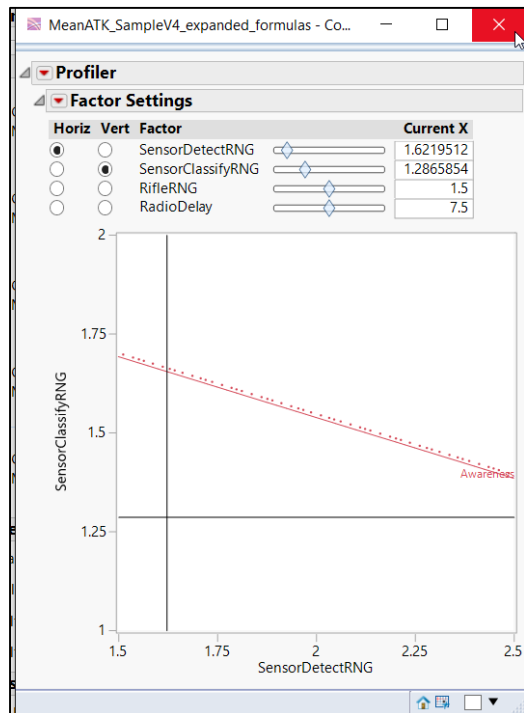




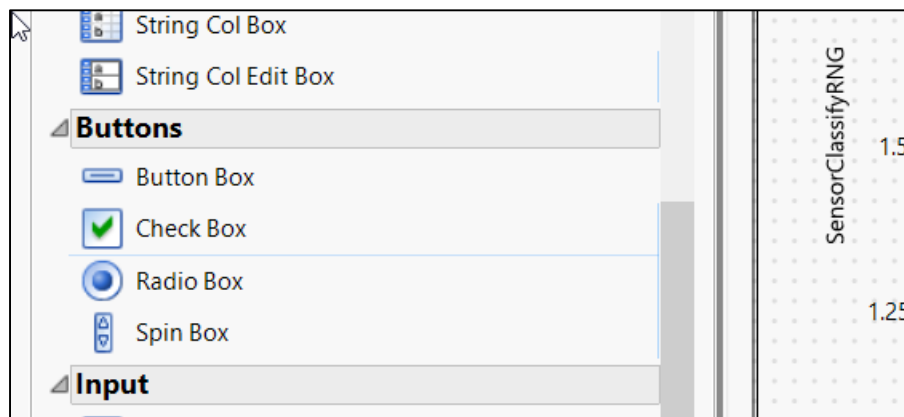
40. Drag the profiler with the factor settings into the V List Box. This will be the bottom Contour Profiler listed in the Reports pane of the Application Builder (see the screenshot below). The Contour Profilers are shown from newest created to oldest created.

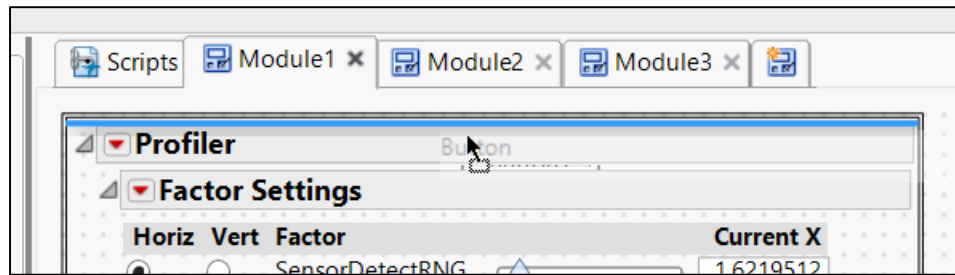


41. You may then close the original Contour Profiler window containing the factor settings. Doing so will remove it from the Reports Pane in the Application Builder as well (but it will leave the copy that you dragged into Module 1 there), meaning you can just work your way up from the bottom of the Reports pane in the following steps. Be careful to close the correct window.

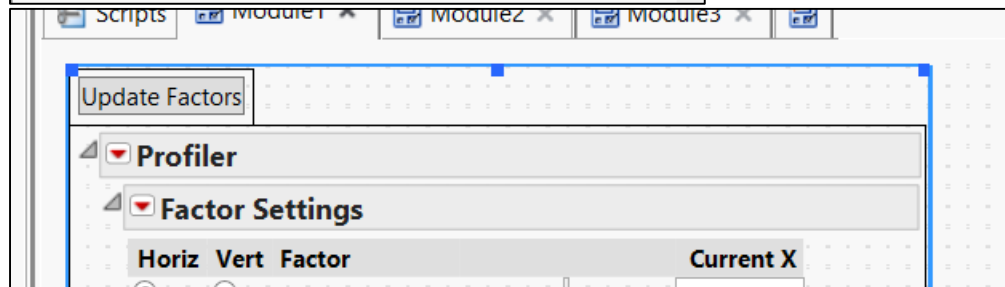
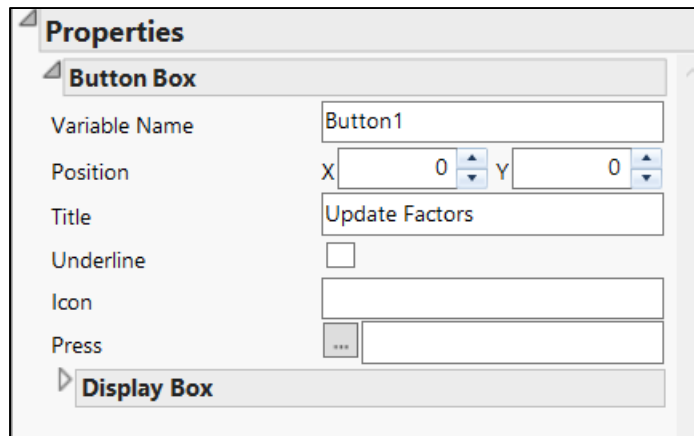


42. Click and drag a "Button Box" (from the left side of the Application Builder) to the top side of the V List Box containing created earlier. You will see a blue line when dragging the Button Box to show where it will be placed.

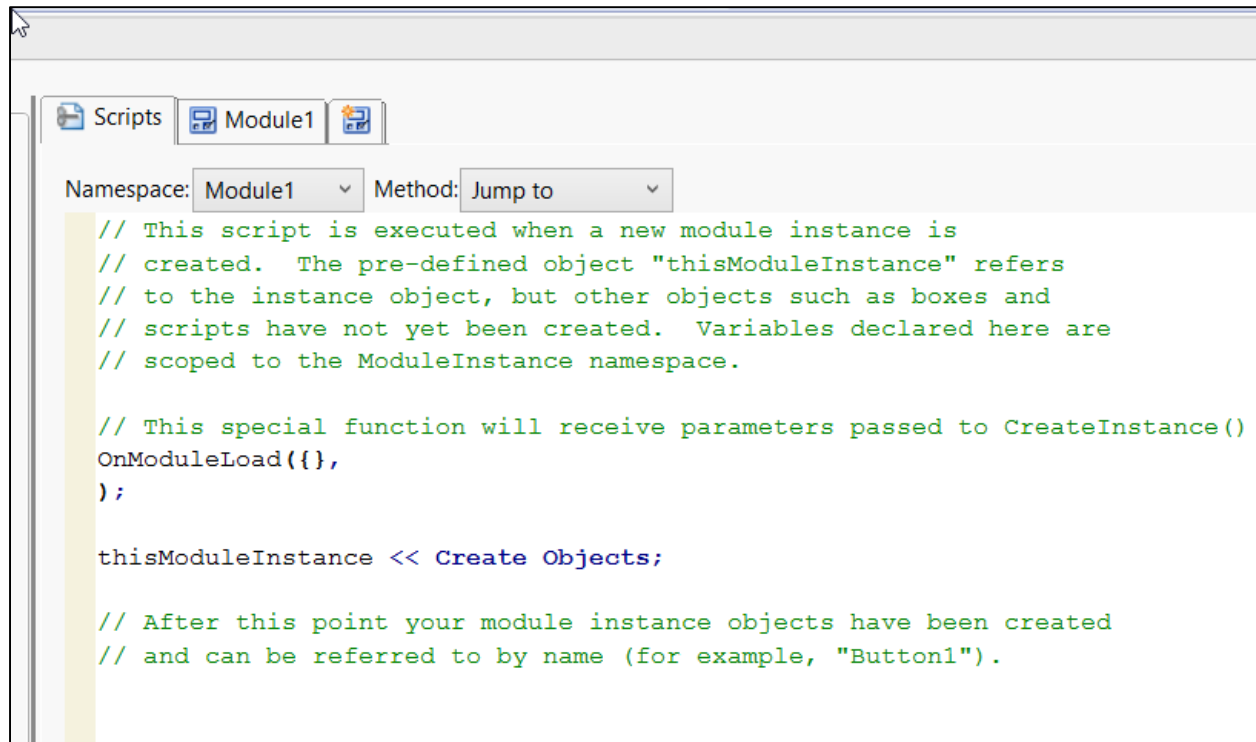




43. Click the Button1 box and change its Title in the Properties Pane to “Update Factors.”



44. Select the Scripts tab (next to the Module1 tab, which contains the factor settings).



45. Below the existing code in this area, enter the code. **When copying code from this manual to JMP, be careful to check if the code block continues onto the next page. The highlighted code blocks are pieces of code that need to be changed if you build the dashboard using a different data table. If rebuilding the dashboard for the MeanATKSamplev4 data set, nothing needs to be changed.**

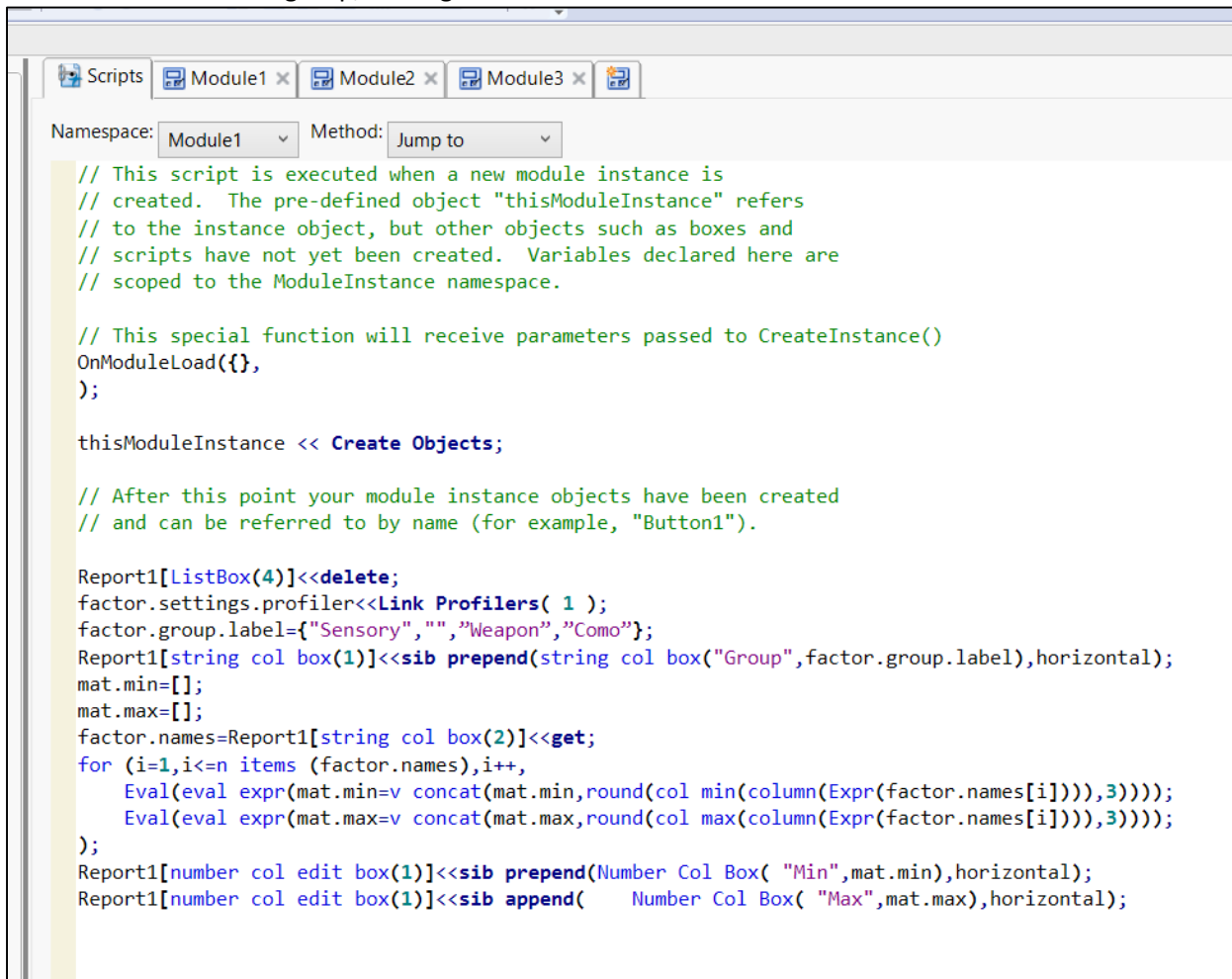
```

Report1[ListBox(4)]<<delete;
factor.settings.profiler<<Link Profilers( 1 );
factor.group.label={"Sensory","","Weapon","Como"};
Report1[string col box(1)]<<sib prepend(string col
box("Group",factor.group.label),horizontal);
mat.min=[];
mat.max=[];
factor.names=Report1[string col box(2)]<<get;
for (i=1,i<=n items (factor.names),i++,
    Eval(eval expr (mat.min=v concat (mat.min,round(col
min(column(Expr(factor.names[i])),3)))));
    Eval(eval expr (mat.max=v concat (mat.max,round(col
max(column(Expr(factor.names[i])),3)))));
);
Report1[number col edit box(1)]<<sib prepend(Number Col Box (
"Min",mat.min),horizontal);
Report1[number col edit box(1)]<<sib append(    Number Col Box (
"Max",mat.max),horizontal);

```

The first line deletes the graph whenever the script runs. The second line ensures that all of the profilers are linked. In the next step, we will manually assign references to the profilers (such as *factor.settings.profiler*, used above). Note how the *factor.group.label* list is manually constructed by first examining the factor list in the contour dashboard. This could be automated somewhat by extracting

the group name for each factor; however, this manual method allows us to only indicate the first column in each group, making the table a bit less cluttered.



```

Namespace: Module1 Method: Jump to

// This script is executed when a new module instance is
// created. The pre-defined object "thisModuleInstance" refers
// to the instance object, but other objects such as boxes and
// scripts have not yet been created. Variables declared here are
// scoped to the ModuleInstance namespace.

// This special function will receive parameters passed to CreateInstance()
OnModuleLoad({},
);

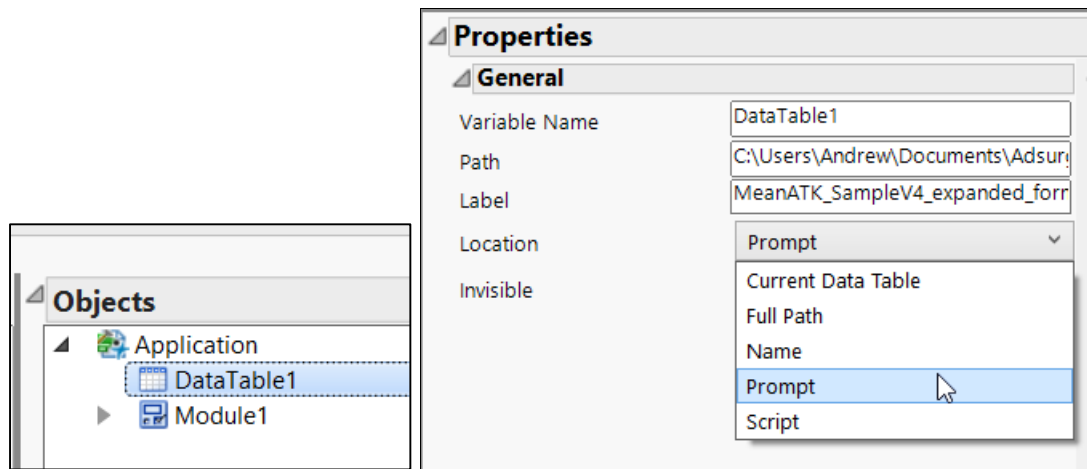
thisModuleInstance << Create Objects;

// After this point your module instance objects have been created
// and can be referred to by name (for example, "Button1").

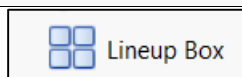
Report1[ListBox(4)]<<delete;
factor.settings.profiler<<Link Profilers( 1 );
factor.group.label={"Sensory", "", "Weapon", "Como"};
Report1[string col box(1)]<<sib prepend(string col box("Group",factor.group.label),horizontal);
mat.min=[];
mat.max=[];
factor.names=Report1[string col box(2)]<<get;
for (i=1,i<=n items (factor.names),i++,
  Eval(eval expr(mat.min=v concat(mat.min,round(col min(column(Expr(factor.names[i]))),3))));
  Eval(eval expr(mat.max=v concat(mat.max,round(col max(column(Expr(factor.names[i]))),3))));
);
Report1[number col edit box(1)]<<sib prepend(Number Col Box( "Min",mat.min),horizontal);
Report1[number col edit box(1)]<<sib append( Number Col Box( "Max",mat.max),horizontal);

```

46. Save your current progress. Recommendation: as you build the dashboard, save a new file each time. In case you make a mistake in a later step that you cannot retrace, you will not lose all of your work.
47. Select DataTable1 from the objects pane. In the Properties pane, change the selection for **Location** from **Current Data Table** to **Prompt**.



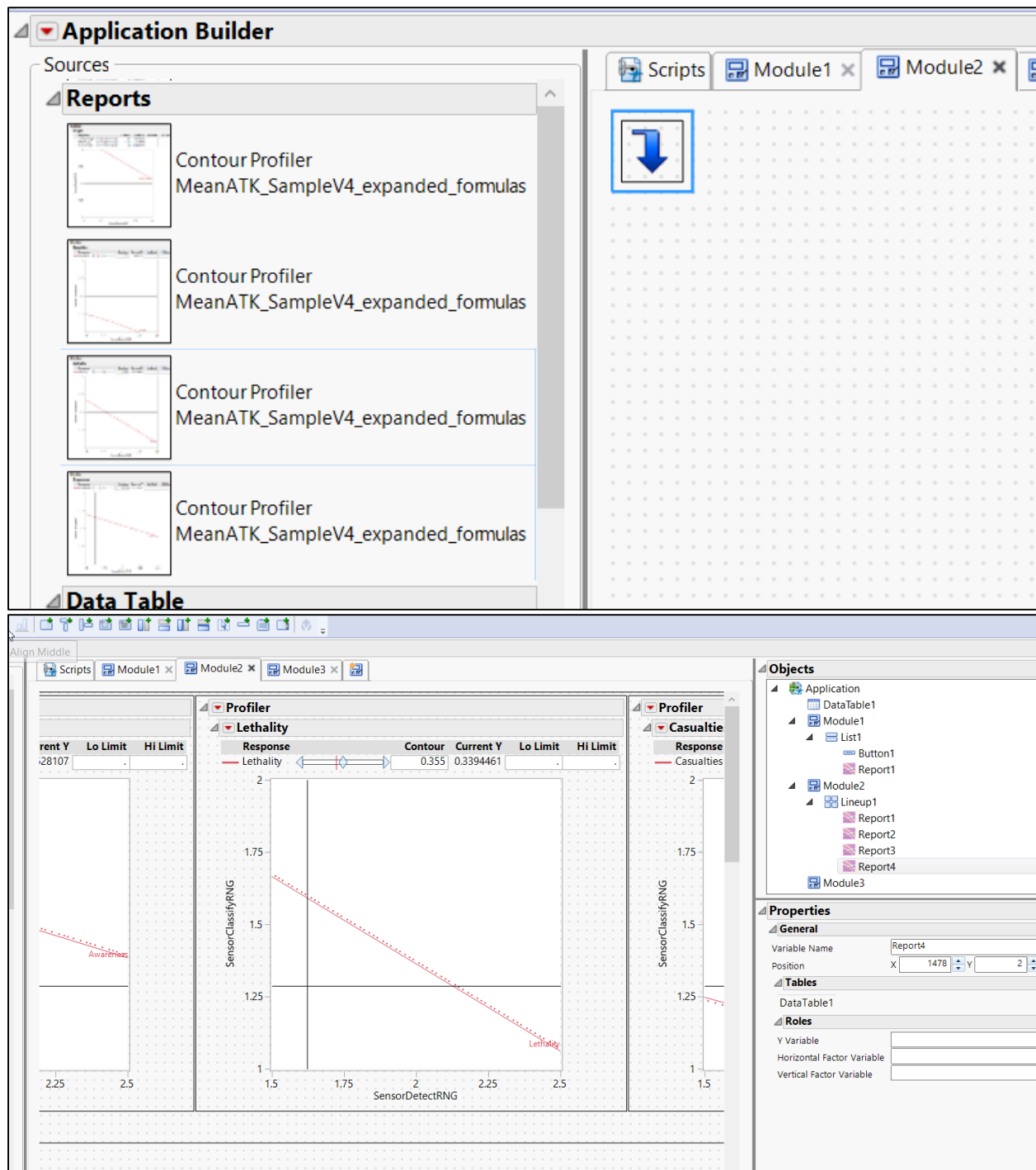
48. Drag a Lineup Box into **Module 2**.



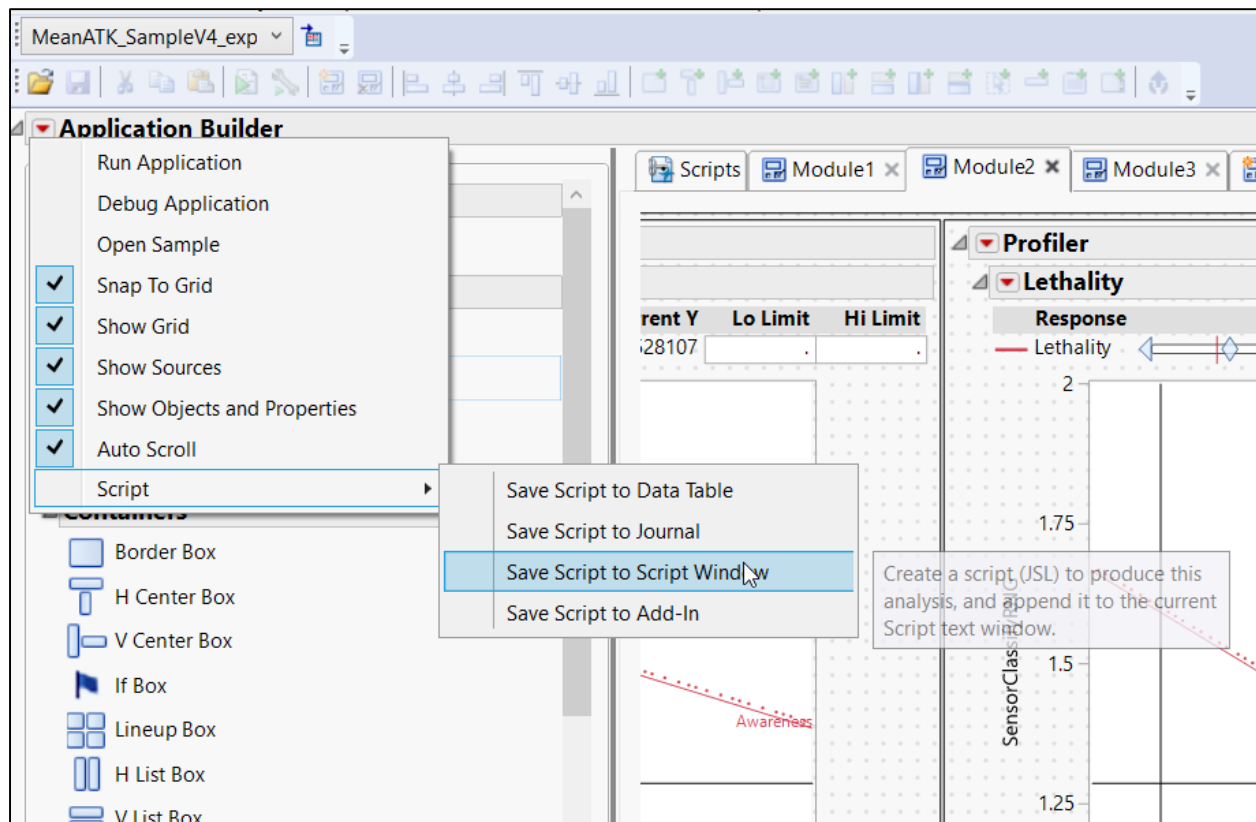
Right-click the new lineup box and select **Move to Corner**.



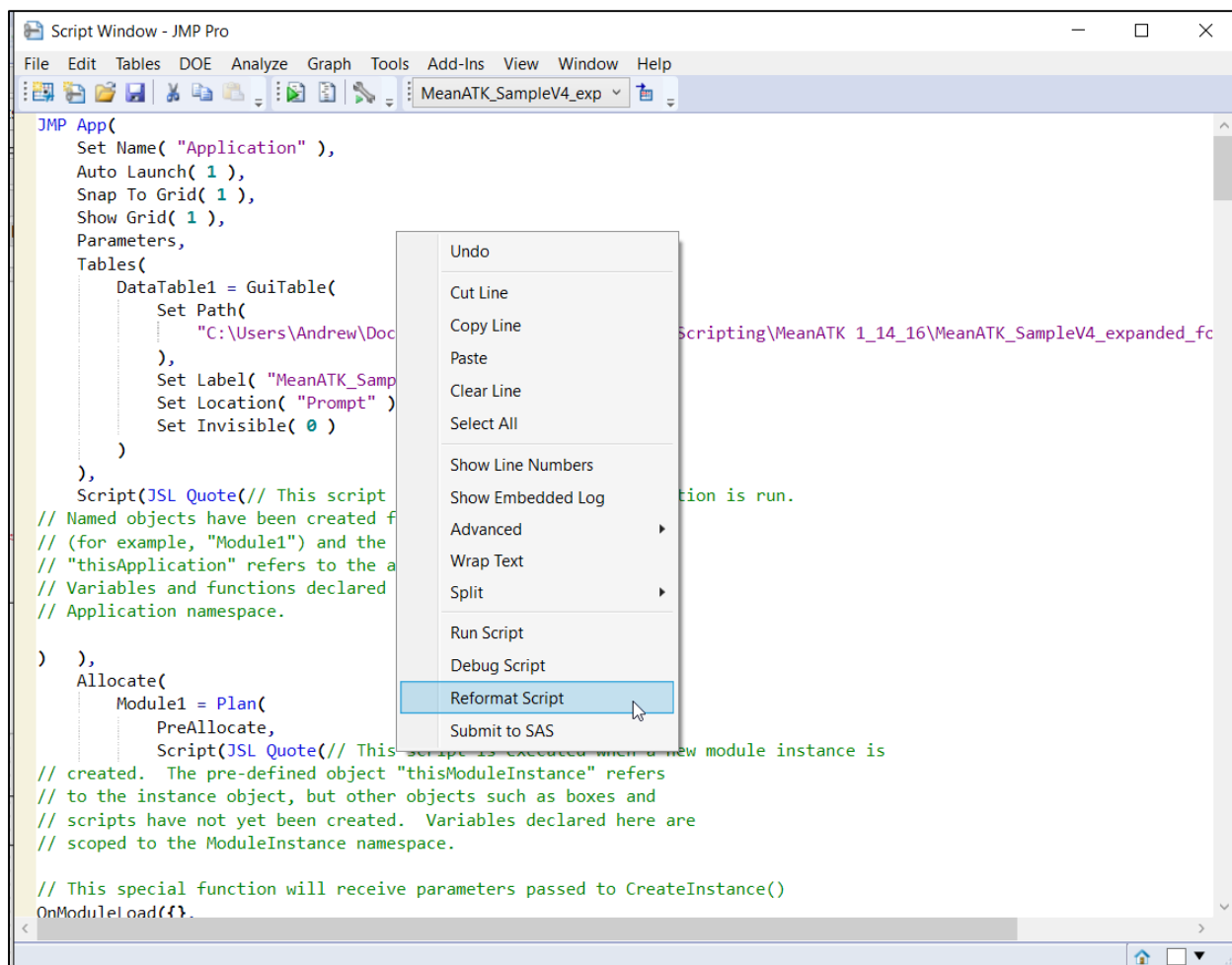
49. Drag the rest of the open contour profilers from the Reports Pane into the lineup box. Put them in a single row across the screen. We will change the underlying code to make them appear in a 2x2 grid. (You can also create a 2x2 grid of contour profilers by clicking and dragging a report to the bottom side of the lineup box. However, the sequence we follow makes things more manageable when building larger dashboards).



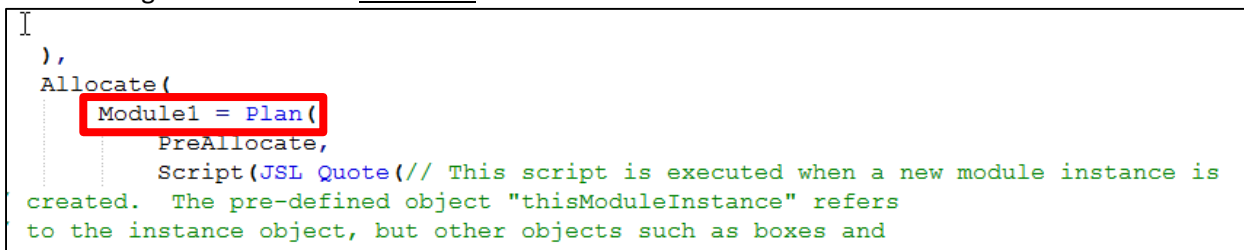
50. Close all of the open Contour Profiler windows (you can either close them one at a time after adding them to the Application Builder, or all at once afterward).
51. Click the red triangle next to Application Builder, hover over "Script", and select **Save Script to Script Window**. (Make sure you don't already have an open Script Window first). This script contains the commands to open an Application Builder with the reports and settings we have added. The following steps insert references to the contour profilers.



In order to make the code more readable, right click in the Script Window and select Reformat Script.



52. Scroll through the code of the Module1 area



The report of the factor settings profiler is named Report1 since it was the first (and only) report added to Module1. **WARNING:** each module will have its own Report1, Report2, etc. Make sure you are in the correct module.

53. Next to the **first instance** of "Contour Profiler" under Report1 (in *Module1*), type "factor.settings.profiler=".

```
Report1 = Platform(
  DataTable1,
  Contour Profiler(
    Y( :Awareness ),
    Contour Profiler(
      1,
      Surface Plot( 0 ),
      Hide Y Controls( 1 ),
      Term Value(
```

```
Report1 = Platform(
  DataTable1,
  factor.settings.profiler=Contour Profiler(
    Y( :Awareness ),
    Contour Profiler(
      1,
      Surface Plot( 0 ),
      Hide Y Controls( 1 ),
      Term Value(
```

Make sure that you have applied the factor.settings.profiler name to the first instance of “Contour Profiler(”, not the second instance that appears two lines later.

In the **Module2** area of the script tab, reports 1 through 4 contain the contour profilers that are visible in the Lineup Box. The following steps add a reference to these contour profilers so that we can extract the factor selections from the profiler in Report1 (in Module1) and then send them to the other profilers.

```
Module2 = Plan(
  PreAllocate,
  Script(JSL Quote(// This script is executed
/ created. The pre-defined object "thisModuleInstance
/ to the instance object, but other objects such as bo
/ scripts have not yet been created. Variables declar
/ scoped to the ModuleInstance namespace.
/ This special function will receive parameters pass
```

54. Each of the profilers in Module 2 need to be named “::profiler#”, with # replaced by the corresponding report number. For example, scroll to the next report in the code (in this case, Report4) and name the Contour Profiler “::profiler4”. The “::” indicates that the variable should be scoped globally, making it available to other modules.

```
Report1 = Platform(
  DataTable1,
  ::profiler1=Contour Profiler(
    Y( :Awareness ),
    Contour Profiler(
      1,
```

```
Report2 = Platform(
  DataTable1,
  ::profiler2=Contour Profiler(
    Y( :Lethality ),
    Contour Profiler(
      1,
```

```
Report3 = Platform(
  DataTable1,
  ::profiler3=Contour Profiler(
    Y( :Casualties ),
    Contour Profiler(
      1,
```

```
Report4 = Platform(
  DataTable1,
  ::profiler4=Contour Profiler(
    Y( :Sensory Weight, :Rifle Weight, :Radio Weight ),
    Contour Profiler(
      1,
```

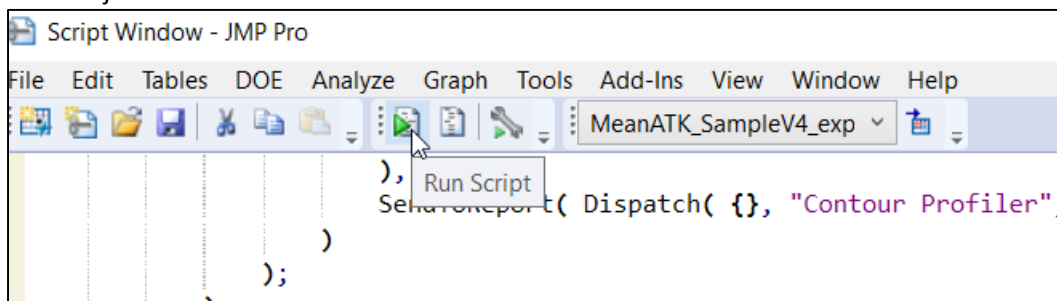
55. Scroll through the code until you find

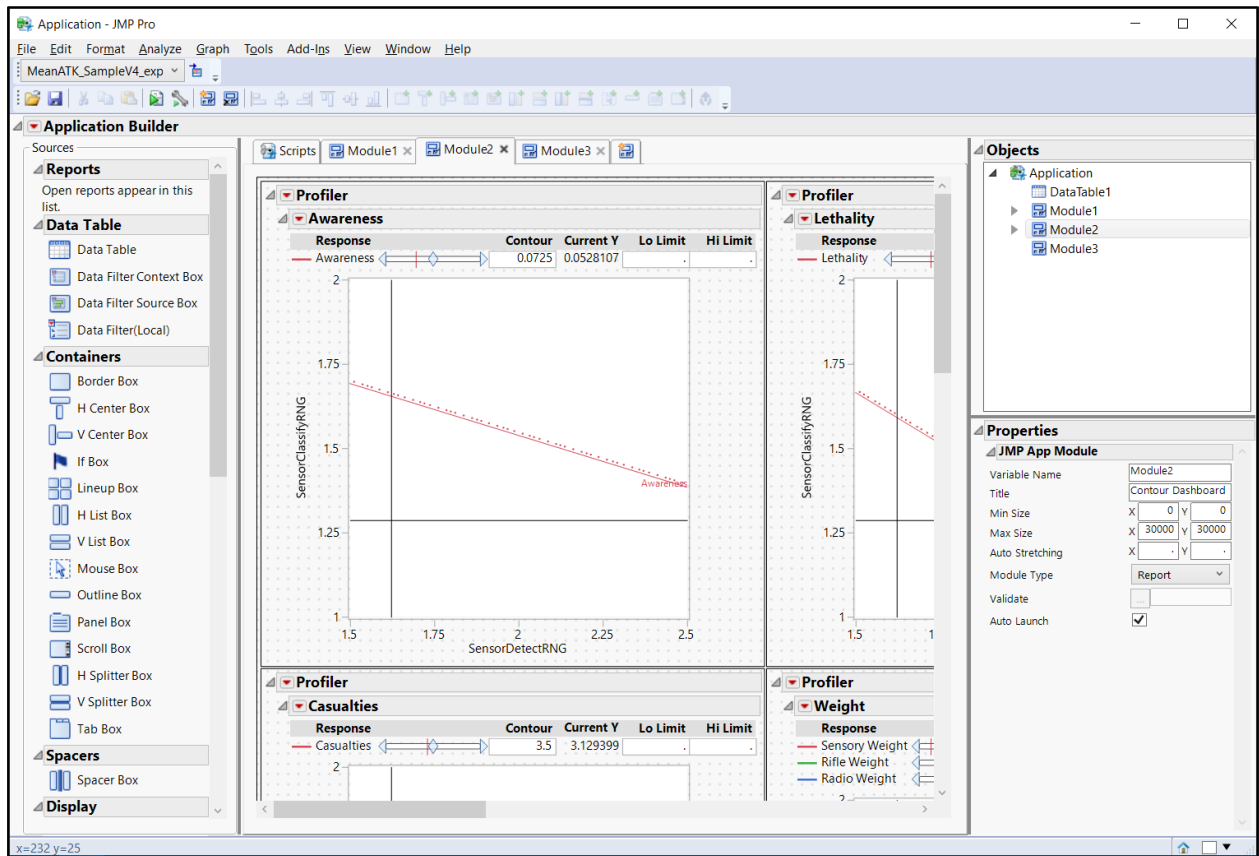
```
Initialize(
  Lineup1 << N Col( 4 );
```

56. The Lineup1 << N Col(4) commands control the number of columns in the lineup boxes. To make the Contour Plots appear in two columns, change the 4 to a 2.

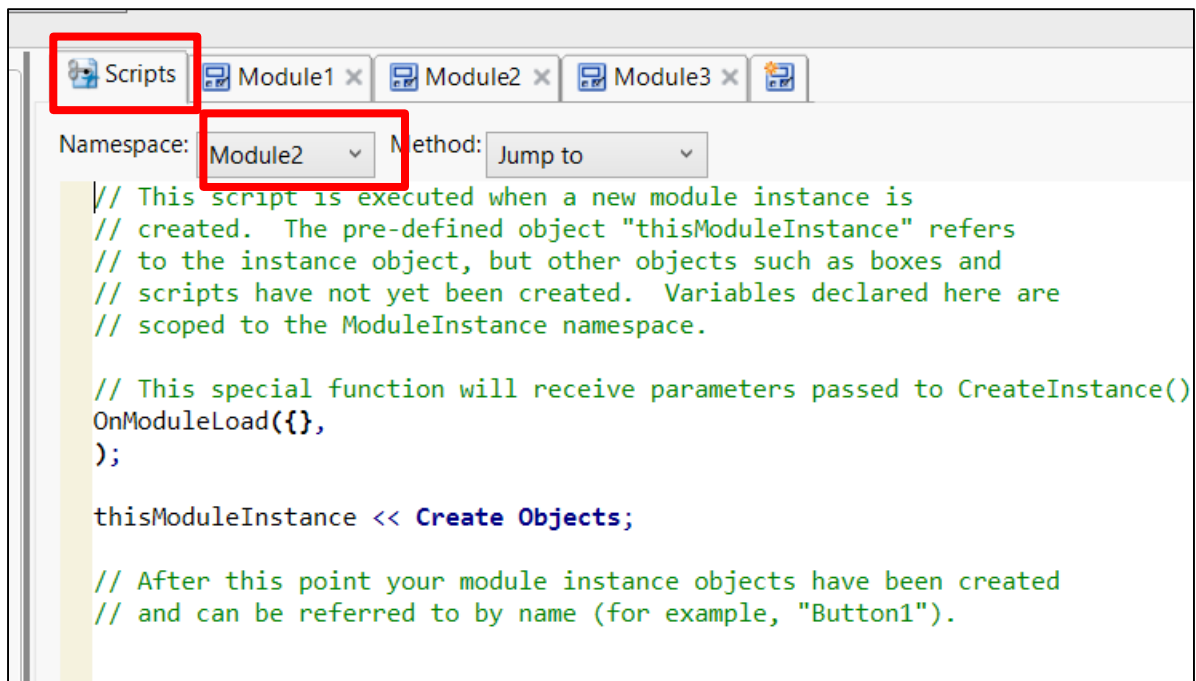
```
Initialize(
  Lineup1 << N Col( 2 );
```

57. After naming each of the profilers and updating the width of the Lineup Box, click the Run Script button to bring up a new application builder. It will look the same as before, but now contains the references that we just built in.





58. After clicking run and opening a new Application Builder, close the Script Window without saving.
59. Save the new application builder as a new file (e.g. Dashboard_pt2).
60. Close the original Application Builder window (not the new one that you just opened).
61. Navigate to the Scripts tab of the Application Builder for **Module 2** and paste the following code to the bottom of the script. The highlighted value of **4** would need to be changed when building the dashboard for different data sets.



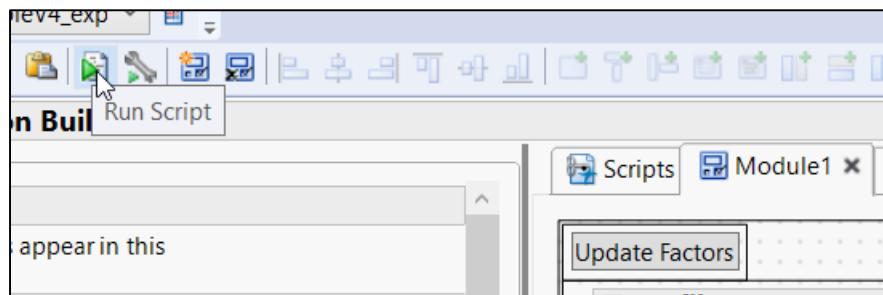

```

//loop over the contour profilers 1 to 4
//set the hi and lo limits to the response column max/min
For( j = 1, j <= 4, j++,
  Eval( Eval Expr( report.names = Expr( Parse( "Report" || Char( j ) ) ) [String
Col Box( 1 )] << get ) );
  Eval( Eval Expr( r2.ncb.lower = Expr( Parse( "Report" || Char( j ) ) ) [Number
Col Edit Box( 2 )] << get as matrix ) );
  Eval( Eval Expr( r2.ncb.upper = Expr( Parse( "Report" || Char( j ) ) ) [Number
Col Edit Box( 3 )] << get as matrix ) );
  For( i = 1, i <= N Items( report.names ), i++,
    r2.ncb.lower[i] = Eval(eval expr(round(col
min(column(Expr(report.names[i])),3)));
    r2.ncb.upper[i] = Eval(eval expr(round(col
max(column(Expr(report.names[i])),3)));
  );
  //update the hi and low limits in the contour profilers
  //make sure to use "set values" and not "set"
  Eval( Eval Expr( Expr( Parse( "Report" || Char( j ) ) ) [Number Col Edit Box(
2 )] << set values( r2.ncb.lower ) ) );
  Eval( Eval Expr( Expr( Parse( "Report" || Char( j ) ) ) [Number Col Edit Box(
3 )] << set values( r2.ncb.upper ) ) );
);
//delete contours
mat=[.];
for(j=2,j<=1000,j++,
  mat=v concat(mat,[.]);
);

//clear the contours from the profilers
//and delete the top "Profiler" (grey) outline boxes
for(i=1,i<=4,i++,
Eval( Eval Expr( Expr( Parse( "Report" || Char( i ) ) ) [Number Col Edit Box( 1 )] <<
set values( mat) ) ) );

```

62. Click the Module 1 tab. (The application cannot be launched from the Scripts tab).
63. Run the script to verify that the modules open correctly and that there have been no errors to this point.



64. After clicking Run Script, a prompt will appear asking which data table to use. Select MeanATK_SampleV4_expanded_formuas, click the button labeled MeanATK_SampleV4_expanded_formuas, and click OK.

Application

Please select or open a data table for each required table.

Select Table
MeanATK_SampleV4_expanded_formula
optional item
Browse

Cast selected table into role
MeanATK_SampleV4_expanded_formulas required item
Remove

OK Cancel

The following 3 windows should open.

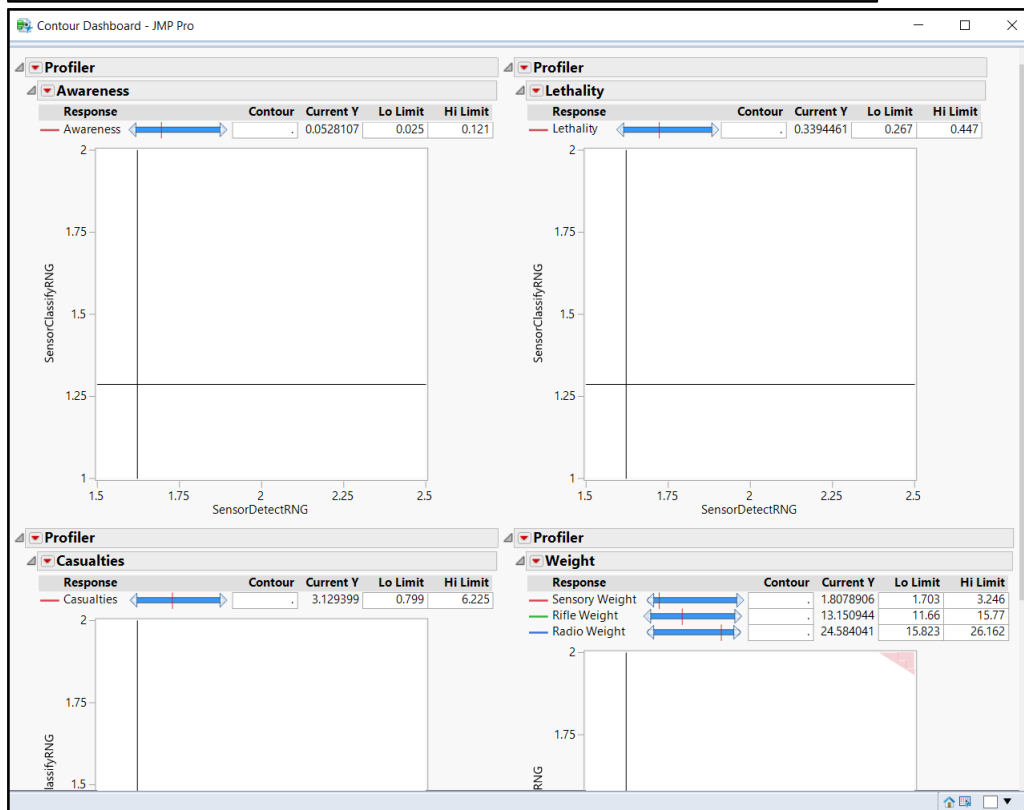
Factor Settings - JMP Pro

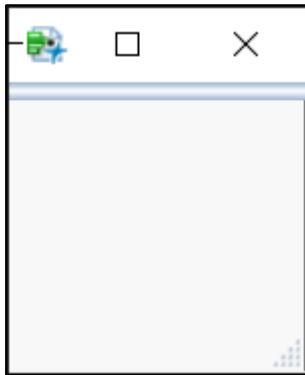
Update Factors

Profiler

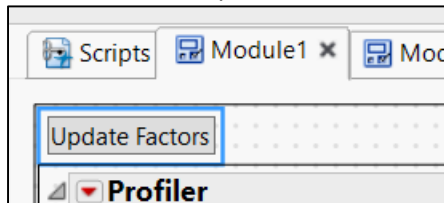
Factor Settings

Horiz	Vert	Group	Factor	Min	Current X	Max
<input checked="" type="radio"/>	<input type="radio"/>	Sensory	SensorDetectRNG	1.5	1.6219512	2.5
<input type="radio"/>	<input checked="" type="radio"/>		SensorClassifyRNG	1	1.2865854	2
<input type="radio"/>	<input type="radio"/>	Weapon	RifleRNG	1	1.5	2
<input type="radio"/>	<input type="radio"/>	Como	RadioDelay	0	7.5	15




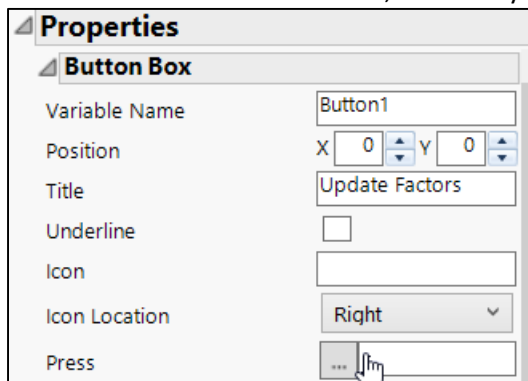


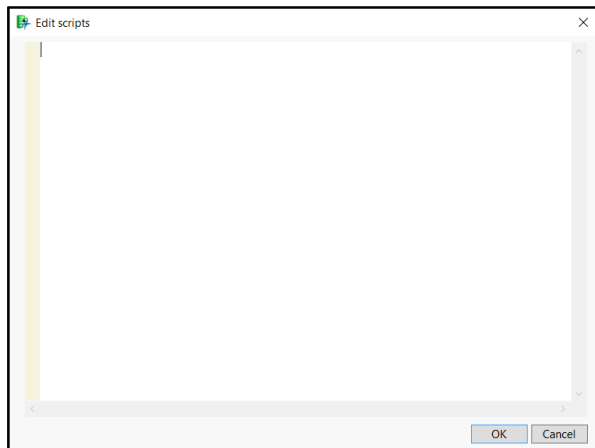
65. Close the 3 new windows that appeared, and return to the Application Builder.
66. Return to Module 1.
67. Click on the Update Factors button to select it.



68. Click the **Press** edit area (for the **Update Factors** button) to enter a script that will be run when the

button is pressed. Note: click the grey button  to paste in the script. If you paste it into the white text box next to that button, it will only save the last line of the script.



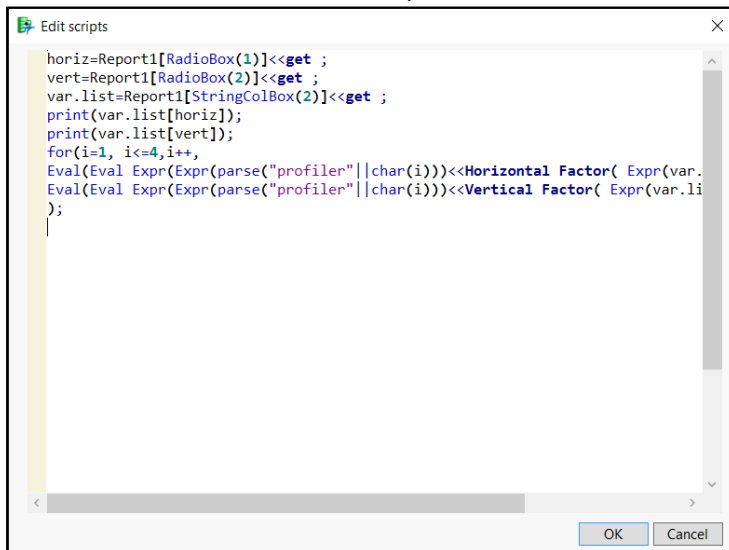


69. Enter the following script

```
horiz=Report1[RadioButton(1)]<<get ;
vert=Report1[RadioButton(2)]<<get ;
var.list=Report1[StringColBox(2)]<<get ;
print(var.list[horiz]);
print(var.list[vert]);
for(i=1, i<=4,i++,
Eval(Eval Expr(Expr(parse("profiler"||char(i)))<<Horizontal Factor(
Expr(var.list[horiz]))));
Eval(Eval Expr(Expr(parse("profiler"||char(i)))<<Vertical Factor(
Expr(var.list[vert]))));
);
```

The upper value of the loop (i<=4) should be set to the number of reports in the Contour Profiler.

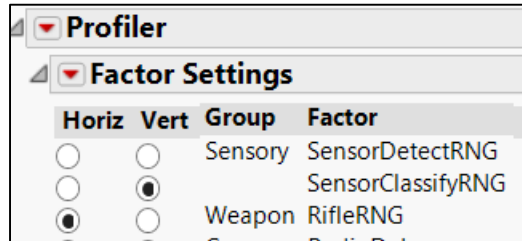
RadioButton(1) contains the selection for the horizontal factor in Report1 (this can be found using the tree structure as shown above).



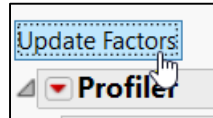
70. Click OK.

71. Click the Run Script button to run the dashboard again. The Update Factors button should now broadcast the factor selections to all of the profilers.

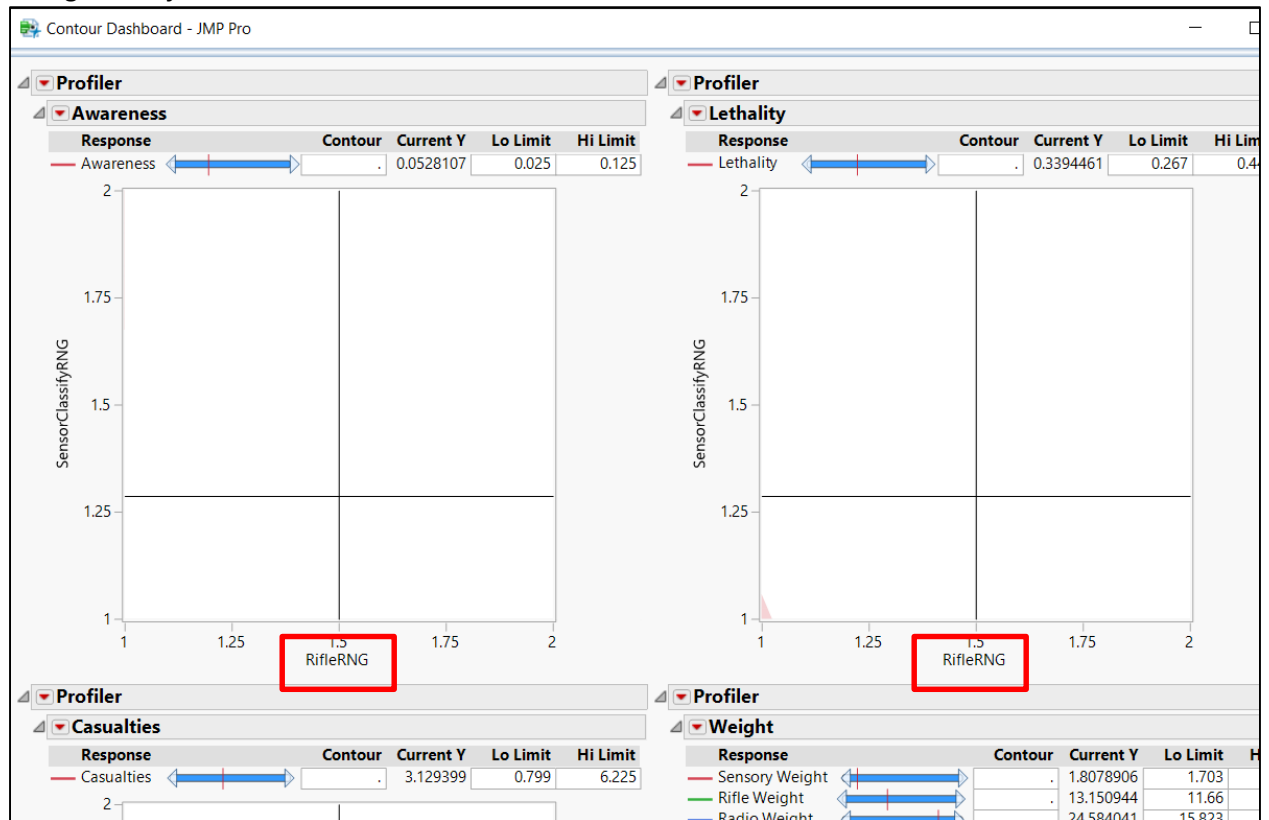
- a. For example, select *RifleRNG* for **Horiz** in the Factor settings window.



- b. Click the Update Factors button.



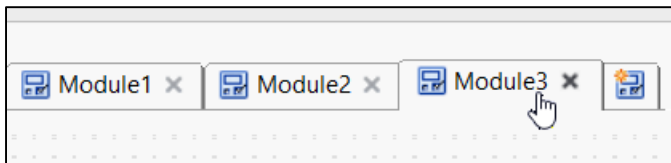
- c. Notice how the x-axis in the Contour Profilers in the Contour Dashboard window has been changed to *RifleRNG*



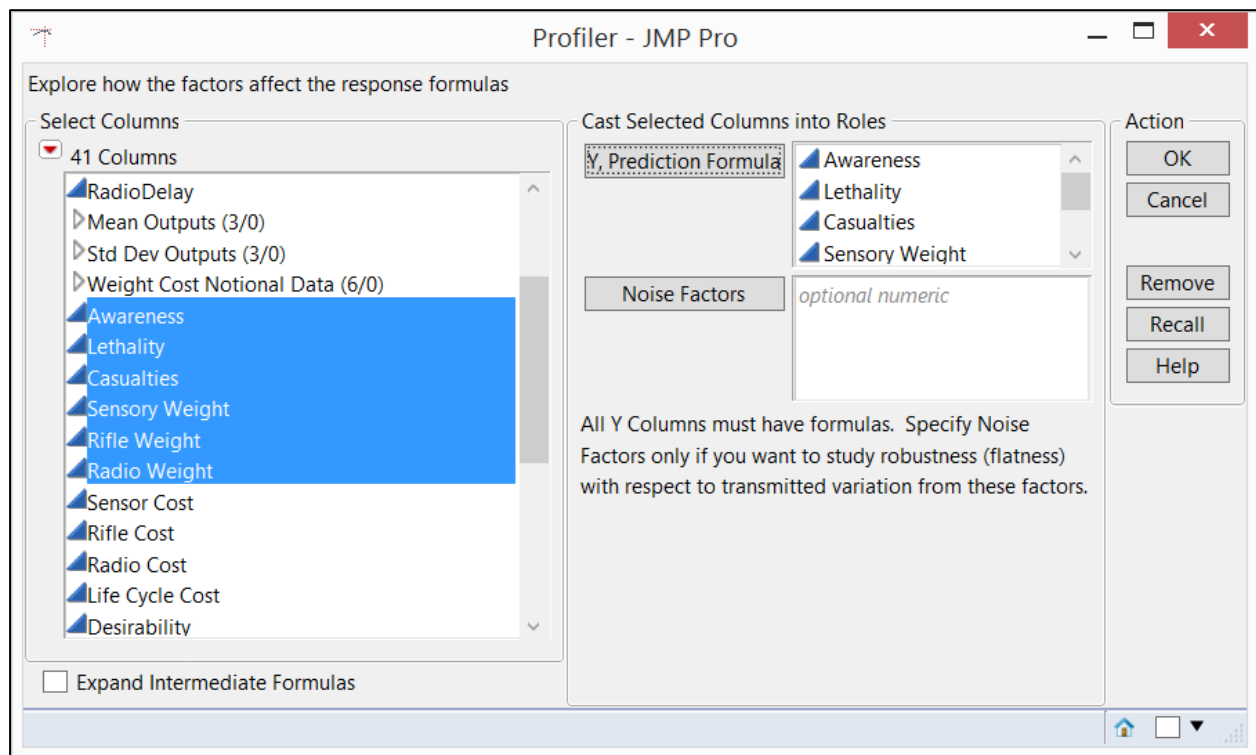
72. Save the application (preferably as a new file). Saving with a .jmpappsource extension will save the Application Builder in its current state, allowing it to be edited later. Saving with a .jmpapp extension will create a file that brings up the dashboard immediately when run.

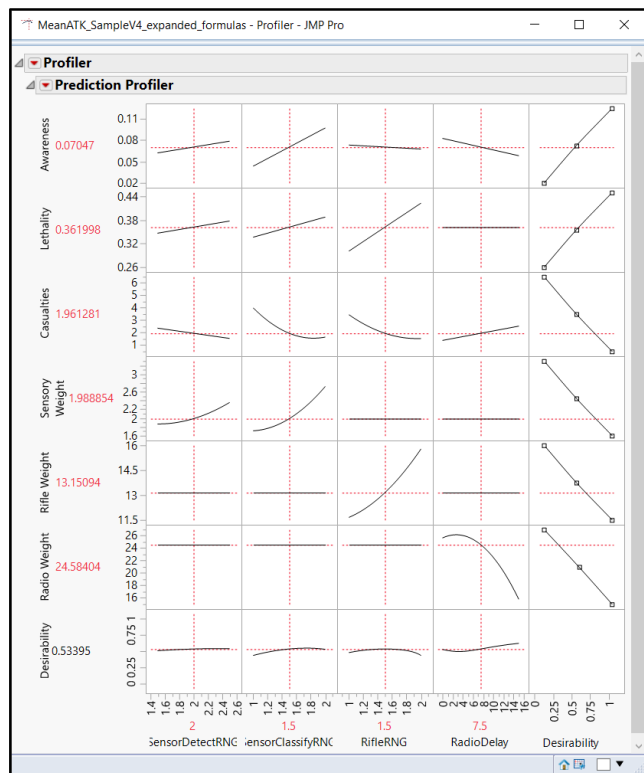
Profiler Dashboard

1. Navigate to Module3 in the Application Builder.

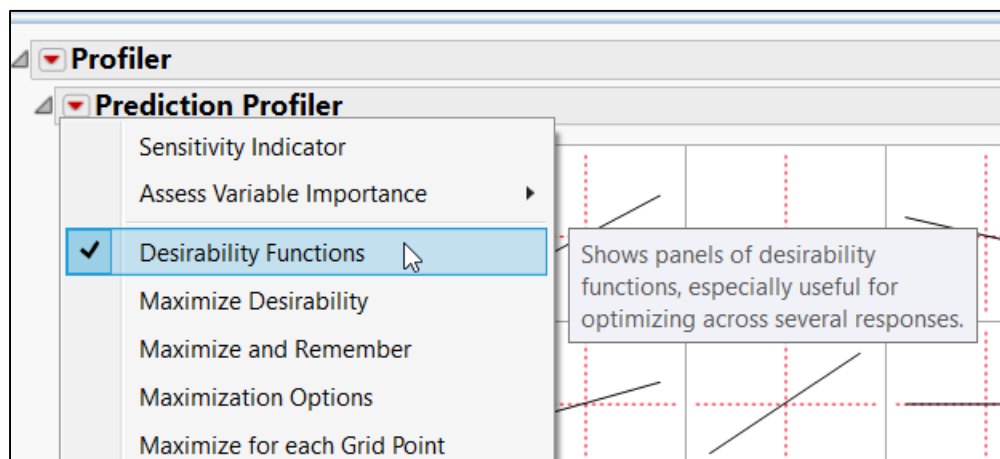


2. From the JMP menu, select **Graph > Profiler**.
3. Select all of the responses (columns *Awareness* through *Radio Weight*) for **Y, Prediction Formula**.
4. Click **OK**.



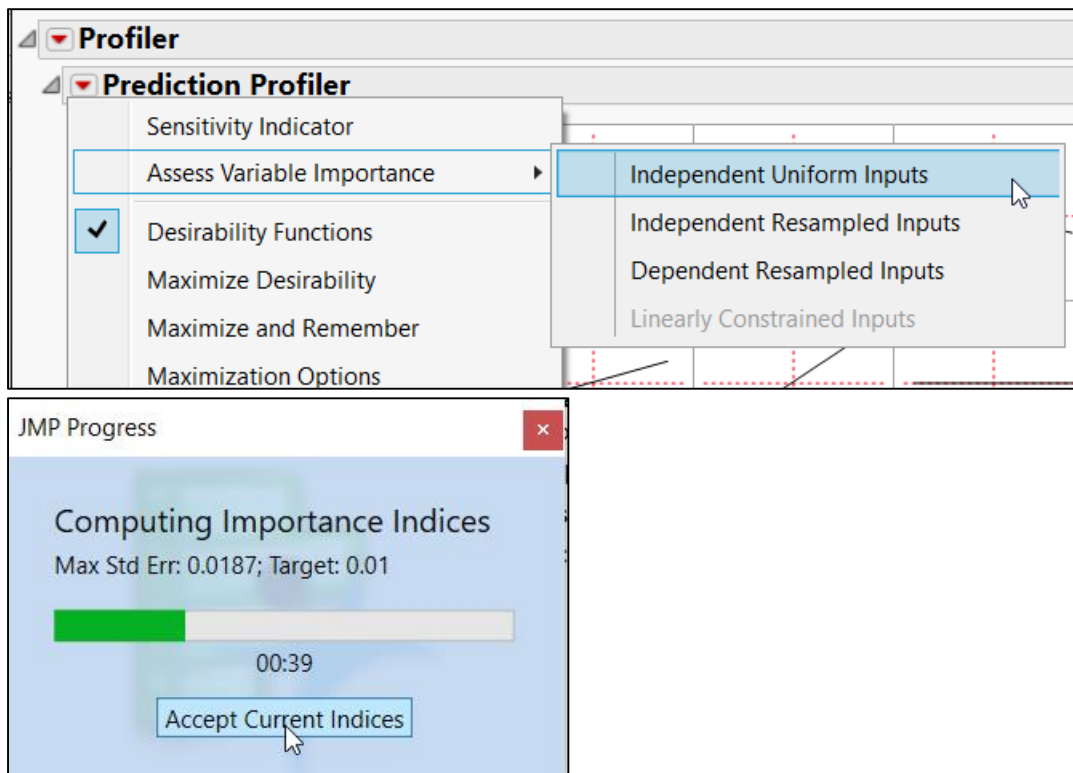


5. Click the red triangle next to Prediction Profiler and ensure that **Desirability Functions** is selected.

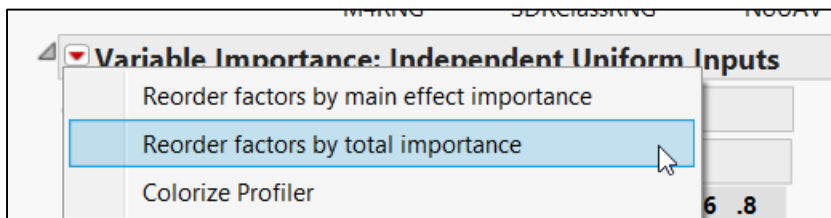


6. Click the red triangle next to Prediction Profiler and select **Assess Variable Importance > Independent Uniform Inputs**. Click “Accept Current Indices” when the button appears (do not need to let it run to completion).

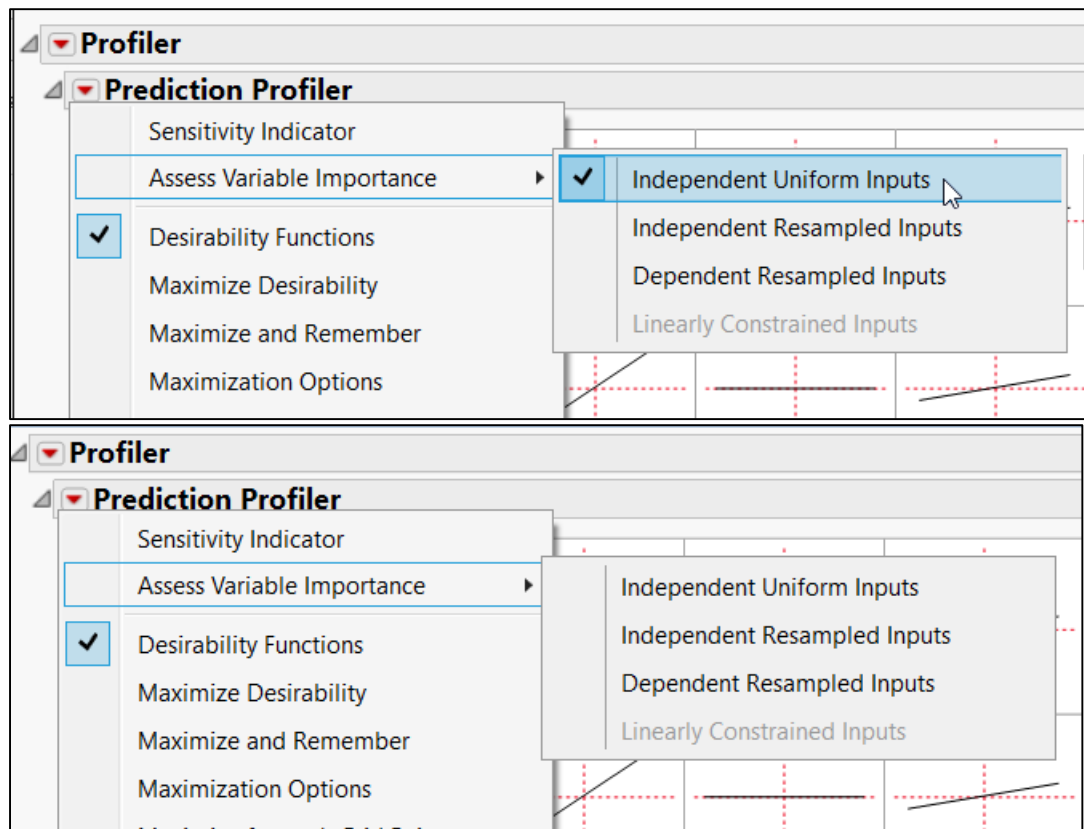
7.



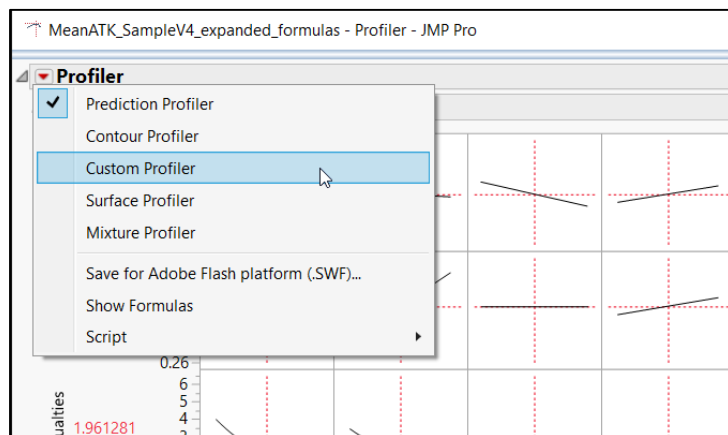
8. Click the red triangle next to Variable Importance: Independent Uniform Inputs and select **Reorder factors by total importance**.



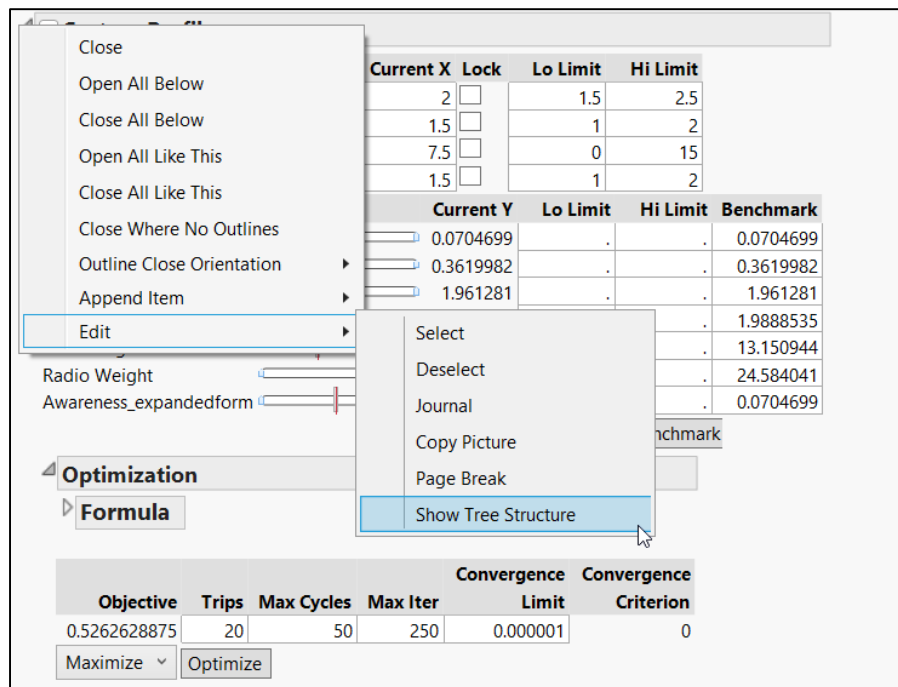
9. Click the red triangle next to Prediction Profiler and de-select **Assess Variable Importance > Independent Uniform Inputs**. This will keep new factor ordering without requiring the simulation to be run each time the dashboard is run.



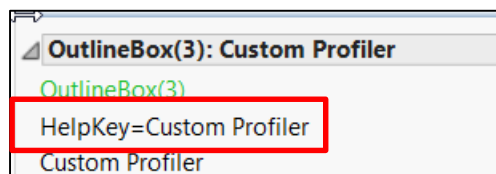
10. Click the red triangle next to Profiler and select **Custom Profiler**.



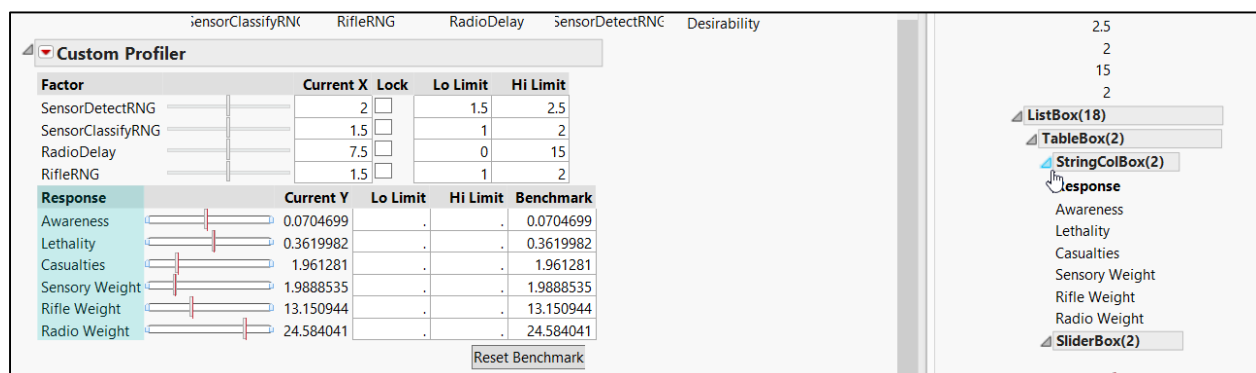
11. Right-click the grey triangle next to Custom Profiler and select **Edit > Show Tree Structure**.



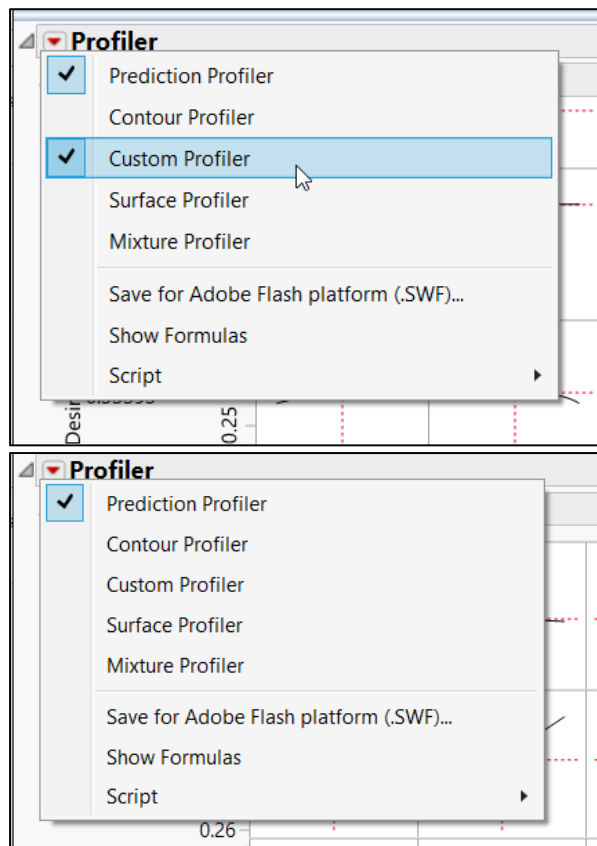
12. The Custom Profiler display area is named "Custom Profiler". We will use this later on to extract this part of the dashboard display.



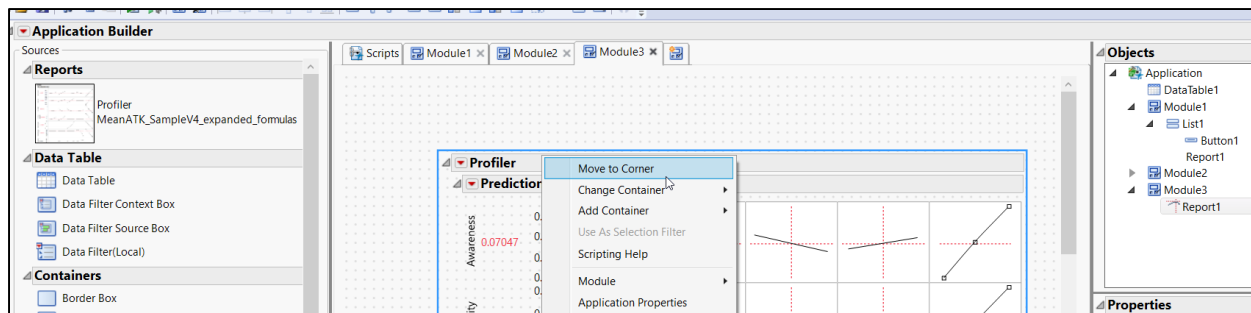
13. By hovering the mouse over parts of the tree structure, we can highlight corresponding display boxes in the Profiler to see that the column of response names resides in *StringColBox(2)*. We will later use the fact that this is the second StringColBox in the Custom Profiler display box to extract the response column names.



14. Click the red triangle next to Profiler and de-select Custom Profiler.

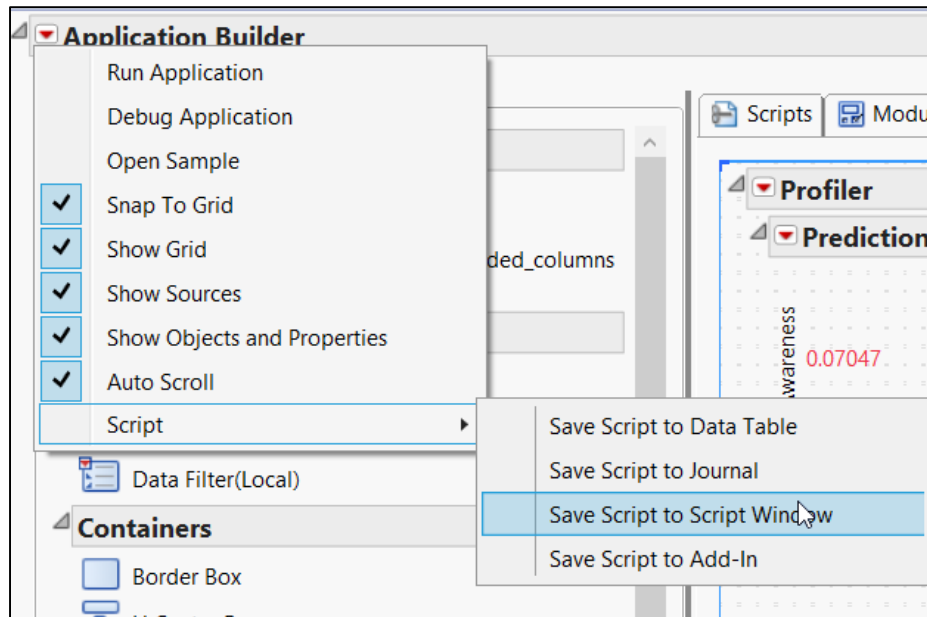


15. Return to the application builder and drag the newly created profiler into **Module 3**.
16. Right-click the profiler and select **Move to Corner**.
17. Close the open Profiler window after importing it into the Application Window. Also close the Display Tree window that was created.



Next, the Application Builder source code is modified to insert references to the profiler we added. (We cannot simply reference, .e.g, *Report1*, since this object refers to the display window that contains the profiler).

18. If you still have a JMP scripting file named *Script Window* open, close it.
19. Click the red triangle next to Application Builder and select **Script > Save Script to Script Window**.



20. Right-click in the Script Window and select **Reformat Script**.
21. Scroll down to the definition of *Report1* in *Module3* (**Warning: Make sure you don't modify Report1 of Module1 or Module2**).

```

Module3 = Plan(
    PreAllocate,
    Script(JSL Quote(// This script is executed when a
// created. The pre-defined object "thisModuleInstance" refer
// to the instance object, but other objects such as boxes and
// scripts have not yet been created. Variables declared here
// scoped to the ModuleInstance namespace.

// This special function will receive parameters passed to Cre
OnModuleLoad({},
);

thisModuleInstance << Create Objects;

// After this point your module instance objects have been cre
// and can be referred to by name (for example, "Button1").

),
    Allocate(
        Report1 = Platform(
            DataTable1,
            Profiler(
                Y(

```

22. Add "pred.profl=" to name the profiler object in *Report1* as shown below.

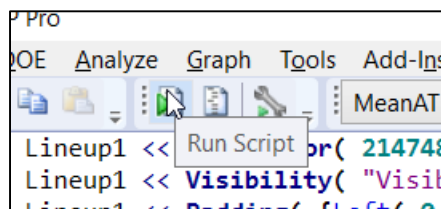
```

Report1 = Platform(
    DataTable1,
    pred.profl=Profiler(
        Y( :Awareness, :Lethality, :Casualties, :Sensory W
        Profiler(
            1,
            Desirability Functions( 1 ),
            :Awareness << Response Limits(
                {Lower( 0.02, 0.066 ), Middle( 0.0725, 0.5

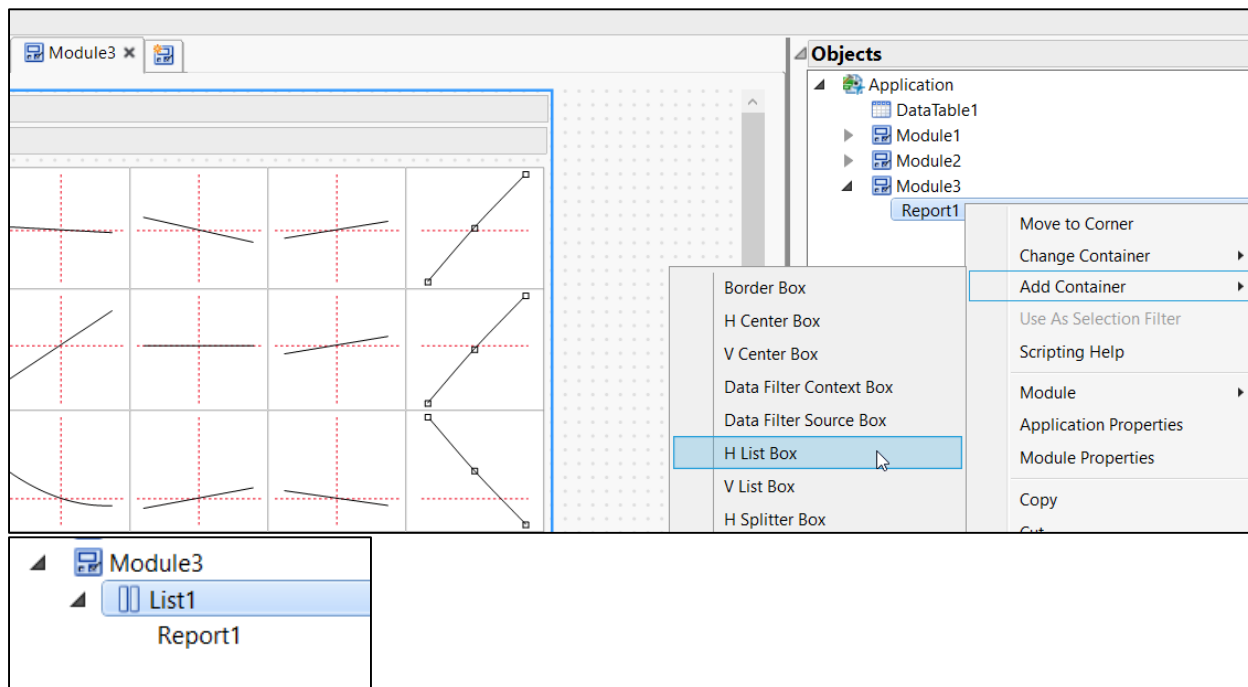
```

Make sure that this name applies to the first instance of “Profiler”, and not the one that appears two lines below it.

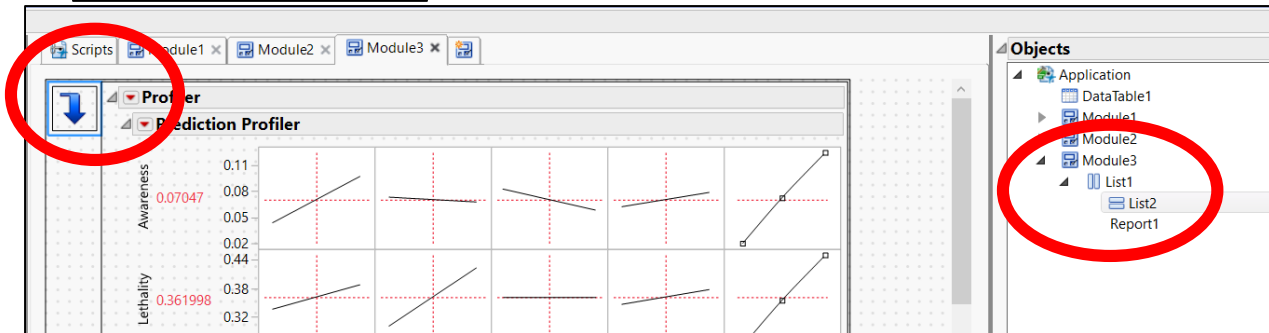
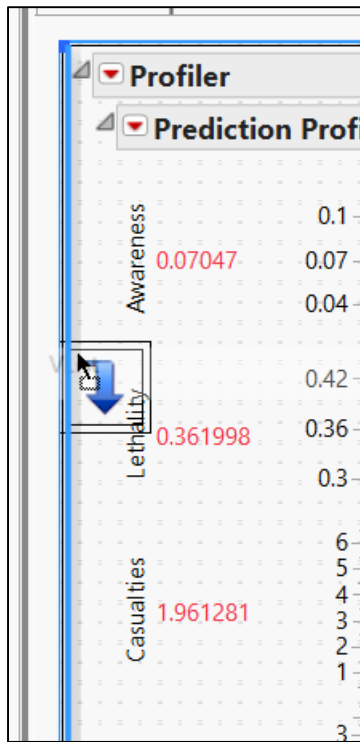
23. Click the Run button to reopen the application builder with the new references built in. You may close the previous instance of the application builder, and save the new Application Builder (as a new file, to be safe).



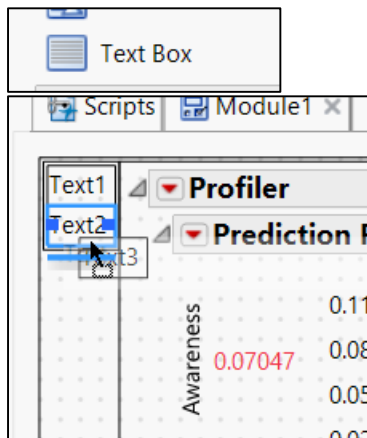
24. Close the “Script Window”.
25. Close the old Application Builder window (not the one that was just created).
26. In Module3 (the prediction profiler), right-click on *Report1* in the objects pane and select **Add Container** > **H List Box**.



27. Drag a **V List Box** (from the left panel of the Application Builder window) into the new **H List Box** to the left of the profiler



28. Drag three **Text Boxes** (from the left panel of the Application Builder window) into the new **V List Box** and fill them out as shown after setting both **Width** and **Wrap** to 200 for each of the boxes in the Properties Pane (on the bottom right of the Application Builder window).



Properties

Text Box

Variable Name: Text1

Position: X 0 Y 0

Font: Segoe UI, Plain, 9pt

Text: Text1

Bullet: ☐

Color:

Justification: Left

Rotation: Horizontal

Width: 200

Wrap: 200

Tooltip:

Scripts Module1 Module2 Module3

Text1
Text2
Text3

Profiler

Prediction Profiler

areness 0.11
0.07047 0.08

Enter the following text in the **Text** area of the properties pane. Selecting the **Bullet** box in the Properties pane creates a bullet.

To maximize the desirability function and save the results with a custom name:

- Click the red triangle next to Prediction Profiler and select Maximize Desirability
- Click the red triangle next to Prediction Profiler and select Factor Settings > Remember Settings

Properties

Text Box

Variable Name: Text1

Position: X 0 Y 0

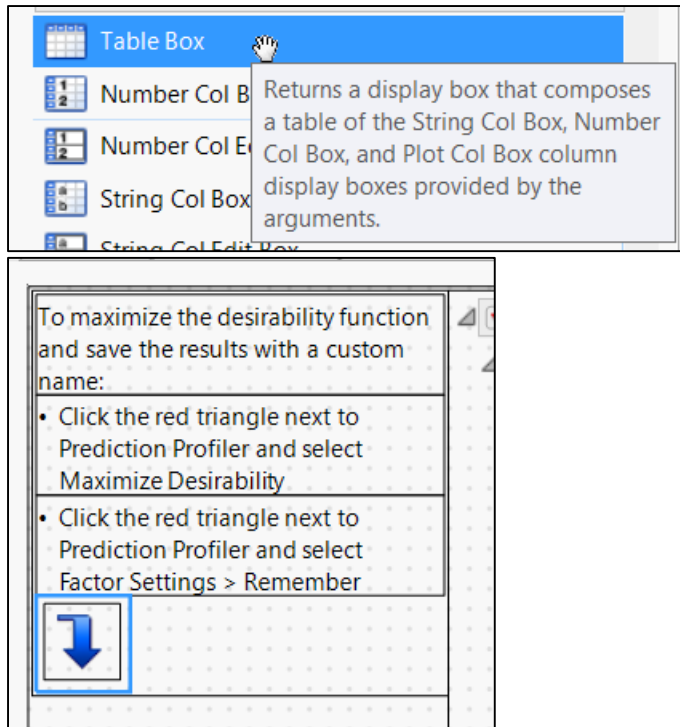
Font: Segoe UI, Plain, 9pt

Text: save the results with a custom name:

Bullet: ☐

Color:

29. Drag a **Table Box** into the **V List Box** under the **Text Boxes**.



30. Insert a **String Col Box**, a **Number Col Edit Box**, a **Number Col Box**, and a **String Col Edit Box** into *Table1* (the newly created **Table Box**).

To maximize the desirability function and save the results with a custom name:

- Click the red triangle next to Prediction Profiler and select Maximize Desirability
- Click the red triangle next to Prediction Profiler and select Factor Settings > Remember Settings

StringCol1	NumberEditCol1	NumberCol1	StringEditCol1
a	1	1	a
b	2	2	b

31. Click on the *StringCol1* box.

32. In the Properties pane for *StringCol1*, change the **Title** to *Response* and remove the items *a* and *b* from the list by selecting them and clicking the minus button.

Properties

Col Box

Variable Name: StringCol1

Position: X 0 Y 0

Title: StringCol1

Items: a, b

Properties

Col Box

Variable Name: StringCol1

Position: X 0 Y 0

Title: Response

Items:

33. Click the *NumberEditCol1* box.

34. In the Properties pane for *NumberEditCol1*, change the **Title** to *Weight* and remove the items 1 and 2 from the list.

The screenshot shows the 'Properties' pane for a 'Col Box' named 'NumberEditCol1'. The 'Variable Name' is 'NumberEditCol1'. The 'Position' is set to X=67 and Y=0. The 'Script' field is empty. The 'Title' is 'Weight'. The 'Items' list is empty, and there are buttons for adding and removing items.

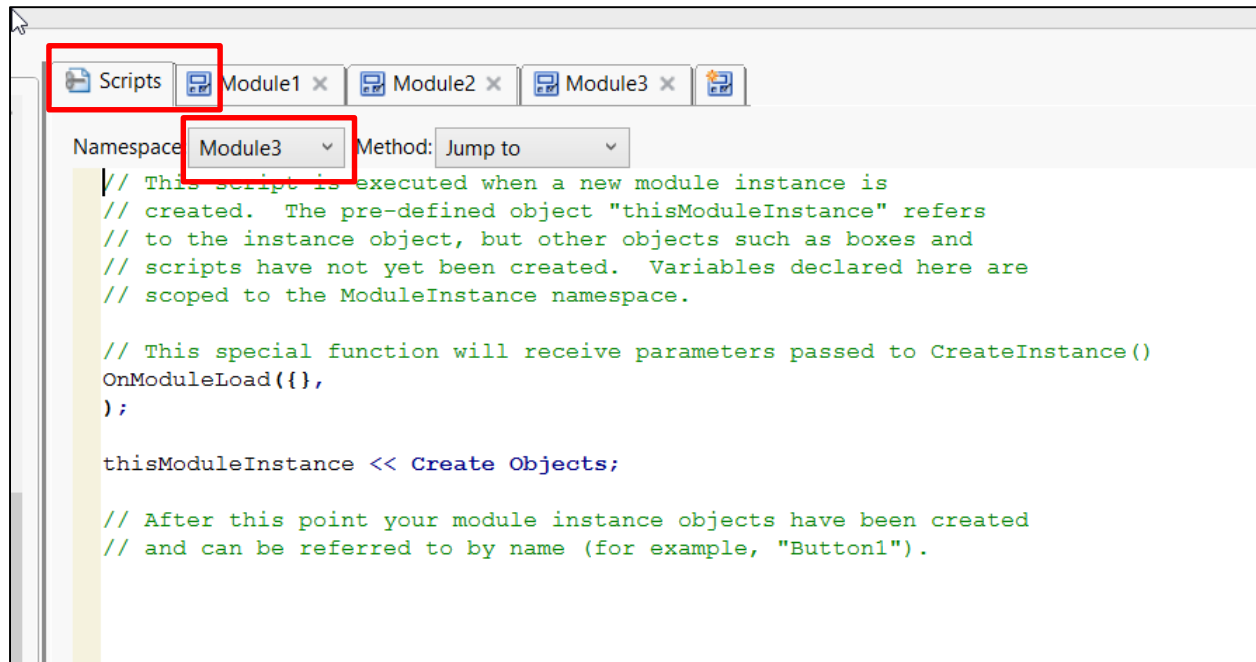
35. Click on the *NumberCol1* box.
36. In the Properties pane for *NumberCol1*, change the **Title** to *Proportion* and remove the items 1 and 2 from the list.
37. Set Variable Name to weight.prop.

The screenshot shows the 'Properties' pane for a 'Col Box' named 'NumberCol1'. The 'Variable Name' is 'weight.prop'. The 'Position' is set to X=123 and Y=0. The 'Title' is 'Proportion'. The 'Items' list is empty, and there are buttons for adding and removing items.

38. Click on the *StringEditCol1* box.
39. In the Properties Pane for *StringEditCol1*, change the **Title** to *Goal* and remove the items *a* and *b* from the list.

The screenshot shows the 'Properties' pane for a 'Col Box' named 'StringEditCol1'. The 'Title' is 'Goal'. The 'Items' list is empty, and there are buttons for adding and removing items.

40. Click on the Scripts tab and select Module3 to add code that will be run when the dashboard is first opened.



41. Enter the following code below the existing code and comments. This is the point at which we use the fact that the response names are contained in String Col Box (2) of the custom profiler (found in an earlier step).

```

//Link the factor settings across profilers
pred.profl << Link Profilers(1);

//extract the response names from the Custom Profiler
pred.profl<< custom profiler( 1 );
domain1.resp = Report(pred.profl)["Custom Profiler"][String Col Box( 2 )] << get;
pred.profl<< custom profiler( 0 );
//Grab the list of response names from the custom profiler

domain1.num = N Items( domain1.resp );
//Initially give equal weight to each response
//And set the default goal to max
For( i = 1, i <= domain1.num, i++,
    NumberEditColl << add element( 1 );
    weight.prop<<add element(1/domain1.num);
    if(contains(char(Column(domain1.resp[i])<<get property("Response
Limits")), "Minimize")==0,
        StringEditColl << add element("max"),
        StringEditColl << add element("min");
);
);
//record the current weights
NEC1.current = NumberEditColl << get;
//pouplate the response names
StringColl << set( domain1.resp );
//records whether the Simulate button has been pressed
sim.indicator=0;
//resize the prediction profiler graphs and
//set the response labels to horizontal
pred.profl<<Dispatch( {"Prediction Profiler"}, "Profiler", FrameBox( 1 ), {Frame Size( 35,
24 )} );
for(i=1, i<=domain1.num, i++,
    Eval(Eval Expr(pred.profl<<Dispatch({"Prediction
Profiler"},Expr(domain1.resp[i]),TextBox,{Rotate Text( "Horizontal" ), Set Wrap( 130
)})));
);
Function( {this, which},
    changed.value = StringEditColl << Get( which );
    window = Expr(
        New Window( "Error",
            <<Modal,
            Text Box( "Goal should be one of {max, min, middle}" ),
            Button Box( "OK" )
        )
    );
    If(
        changed.value != "max" & changed.value != "min" & changed.value
        != "middle",
        Eval( window );
        Stop();
    );
    //update the desirability functions
    //set.des1 is a button that will be created
    set.des1 << click();
);

```

The screenshot shows the 'Scripts' window in the Application Builder. The 'Namespace' is set to 'Module3' and the 'Method' is 'Jump to'. The script contains the following code:

```
// This script is executed when a new module instance is
// created. The pre-defined object "thisModuleInstance" refers
// to the instance object, but other objects such as boxes and
// scripts have not yet been created. Variables declared here are
// scoped to the ModuleInstance namespace.

// This special function will receive parameters passed to CreateInstance()
OnModuleLoad({},
);

thisModuleInstance << Create Objects;

// After this point your module instance objects have been created
// and can be referred to by name (for example, "Button1").

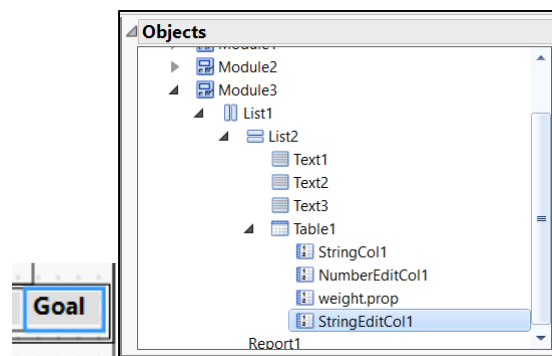
//Link the factor settings across profilers
pred.profl << Link Profilers(1);

//extract the response names from the Custom Profiler
pred.profl<< custom profiler( 1 );
domain1.resp = Report(pred.profl)["Custom Profiler"][String Col Box( 2 )] << get;
pred.profl<< custom profiler( 0 );
//Grab the list of response names from the custom profiler

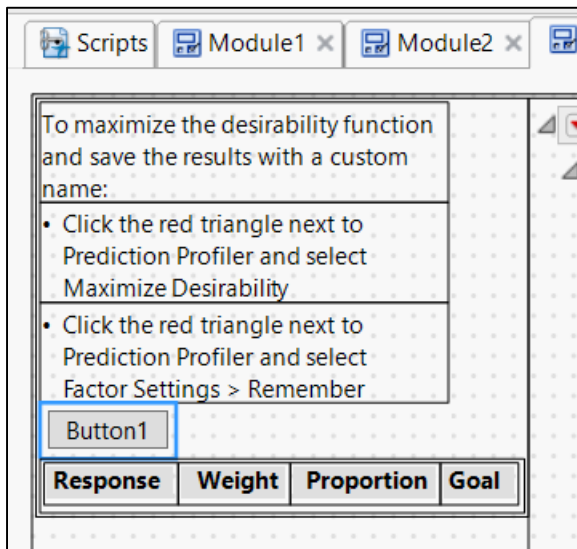
domain1.num = N Items( domain1.resp );
//Initially give equal weight to each response
//And set the default goal to max
For( i = 1, i <= domain1.num, i++,
    NumberEditCol1 << add element( 1);
    weight.prop<<add element(1/domain1.num);
    if(contains(char(Column(domain1.resp[i])<<get property("Response Limits")), "Minimize")==0,
        StringEditCol1 << add element("max"),
        StringEditCol1 << add element("min");
);
);
//record the current weights
MEC1.current = NumberEditCol1 << get;
```

42. Return to the Module3 tab.

43. Select the StringEditCol box titled *Goal*. For some reason, this is difficult to do. You may need to just select *StringEditCol1* from the **Objects** pane on the top right of the Application Builder window.

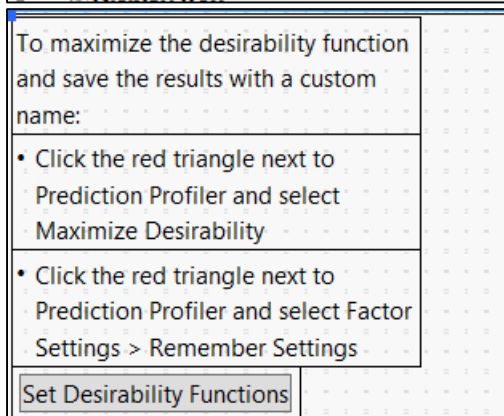
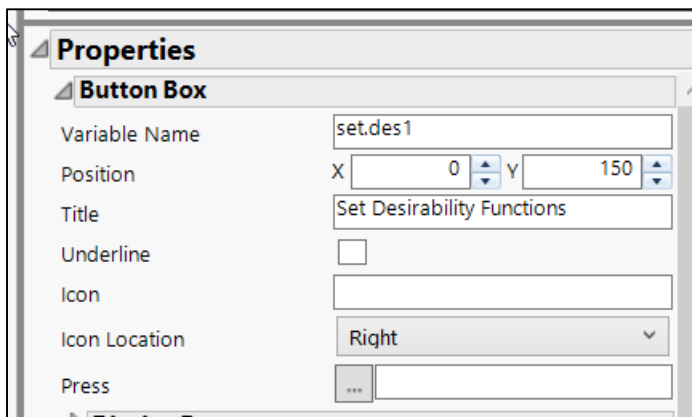


44. Add a Button Box below the **Text Boxes** in Module 3. This button will contain the code for maximizing the desirability function.



45. In the properties pane, set the **Variable Name** of the box to `set.des1`

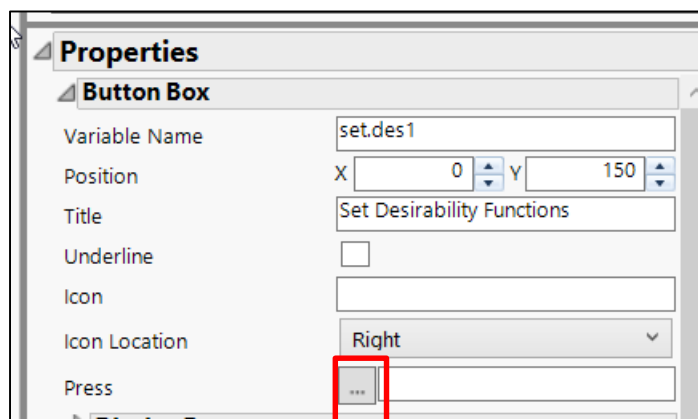
46. Set the title to *Set Desirability Functions*.



47. Enter the following in the **Press** area of the Properties pane for the set.des1 button. Note: click the grey



button to paste in the script. If you paste it into the white text box next to that button, it will only save the last line of the script.



```
names = StringCol1 << get;
upper.lim = {};
lower.lim = {};
For( i = 1, i <= N Items( names ), i++,
    lower.lim = Insert( lower.lim, 0 );
    upper.lim = Insert( upper.lim, 0 );
);

//loop over the contour profilers 1 through 4
//get the hi and lo limits and the corresponding response names from each Report
For( j = 1, j <= 4, j++,
    Eval( Eval Expr( report.names = Expr( Parse( "report(profiler" || Char( j )
|| ")" ) ) [String Col Box( 1 )] << get ) );
    Eval(
        Eval Expr(
            r2.ncb.lower = Expr( Parse( "report(profiler" || Char( j ) || ")"
) ) [Number Col Edit Box( 2 )] << get as matrix
        )
    );
    Eval(
        Eval Expr(
            r2.ncb.upper = Expr( Parse( "report(profiler" || Char( j ) || ")"
) ) [Number Col Edit Box( 3 )] << get as matrix
        )
    );
    For( i = 1, i <= N Items( names ), i++,
        Eval( Eval Expr( loc = Contains( report.names, Expr(names[i] ) ) ) );
        //when a match is found, the data filter is updated
        //the appropriate command for the data filter depends on whether there
are missing values
        //for the Hi or Lo Limits in the contour profiler.
        //Note: due to lack of readability of if-then statements in JMP, the
three conditions are
        //listed separately in three if-statements
```

```

        If( loc > 0,
            lower.lim[i] = r2.ncb.lower[loc];
            upper.lim[i] = r2.ncb.upper[loc];
        );

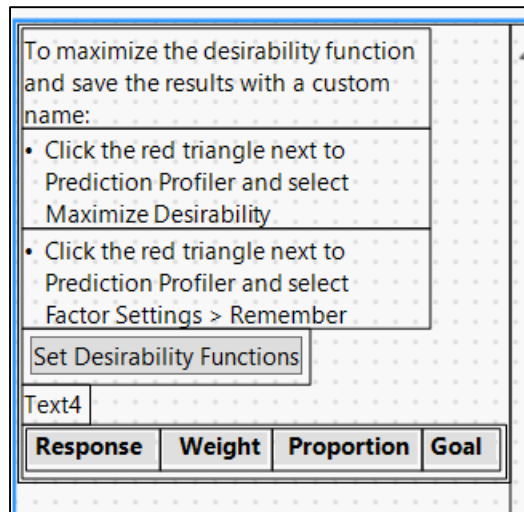
    );

);

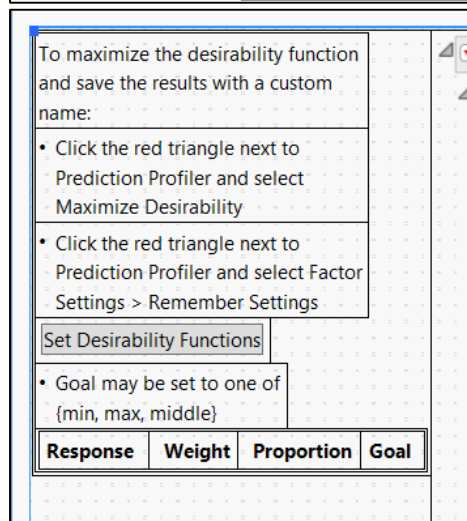
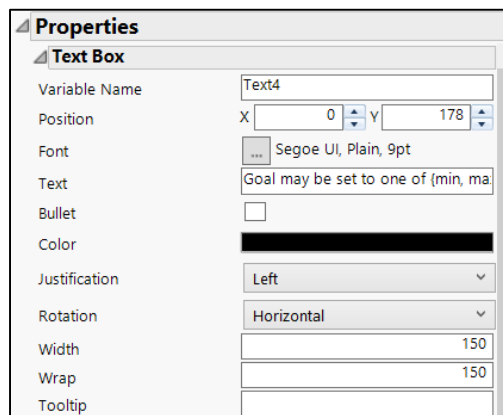
For( i = 1, i <= N Items( StringColl << get ), i++,
    var.name = StringColl << get( i );
    max = upper.lim[i];
    min = lower.lim[i];
    delta = 1e-6;
    goal = StringEditColl << get( i );
    If( goal == "max",
        Eval(
            Eval Expr(
                pred.profl << (Expr( var.name ) << Response Limits(
                    {Lower( min, 0.1 ), Middle( max, 1 ), Upper( max +
delta, 0.1 )}
                ))
            )
        );
    If( goal == "min",
        Eval(
            Eval Expr(
                pred.profl << (Expr( var.name ) << Response Limits(
                    {Lower( min - delta, 0.1 ), Middle( min, 1 ), Upper(
max, 0.1 )}
                ))
            )
        );
    If( goal == "middle",
        Eval(
            Eval Expr(
                pred.profl << (Expr( var.name ) << Response Limits(
                    {Lower( min, 0.1 ), Middle( (min + max) / 2, 1 ),
Upper( max, 0.1 )}
                ))
            )
        );
);
Report1[PictureBox(1)]<<reshow;

```

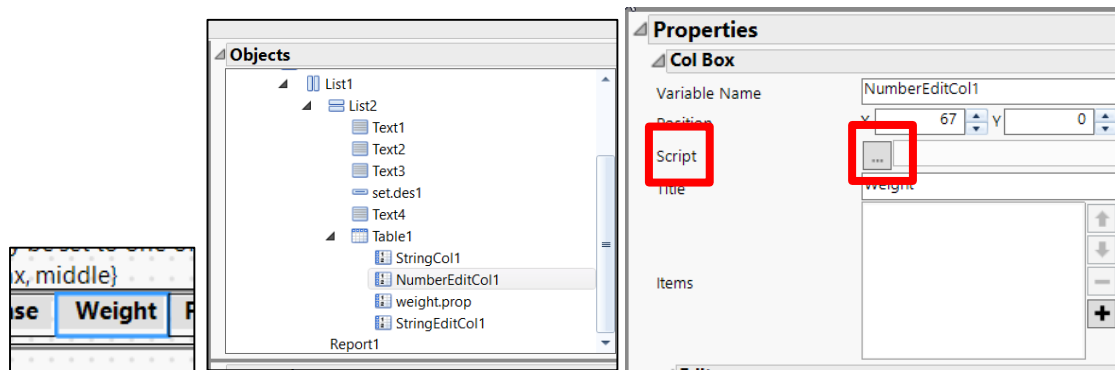
48. Insert a Text Box below the **Set Desirability Functions** button.



49. Set the width and wrap of the new Text Box to 150 in the Properties pane, and set the text to “Goal may be set to one of {min, max, middle}”



50. Select the *Weight* column. Again, this may be easier to do by selecting *NumberEditCol1* from the Objects pane at the top right of the Application Builder.



51. In the **Script** area (in the properties pane for *Weight*), click the grey button to enter the following code that will automatically run whenever the weights are changed.

```
Print( input.values = NumberEditCol1 << Get() );
Print( "Old Values" );
Print( old.values = NEC1.current );
window = Expr(
    New Window( "Error",
        <<Modal,
        Text Box( "Weights should be >0" ),
        Button Box( "OK" )
    )
);
neg.ind=0;
For( i = 1, i <= N Items( input.values ), i++,
if(input.values[i]<0,neg.ind=1);

);
If( neg.ind==1,
    Eval( window );
    NumberEditCol1 << Set( old.values );
    Stop();
);

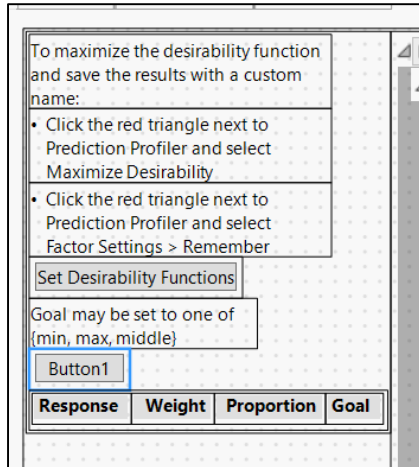
NEC1.current = input.values;
For( i = 1, i <= N Items( input.values ), i++,
    var.name = StringCol1 << get( i );
    var.value = input.values[i];
    Eval(
        Eval Expr(
            pred.profl << (Expr( var.name ) <<
                Response Limits( {Importance( Expr( var.value ) )} ))
        )
    );
);

weight.matrix=NumberEditCol1 << Get as matrix;
weight.sum = sum(weight.matrix);
weight.scale=weight.matrix/weight.sum;
weight.prop<<set values(weight.scale);
```

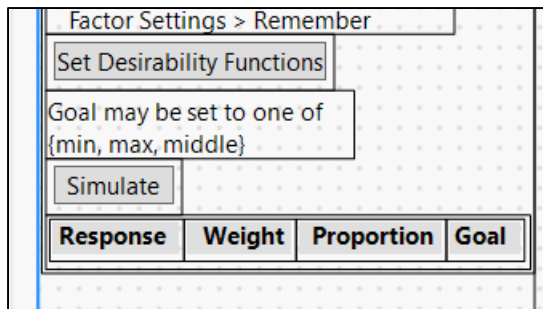
52. Save the dashboard. Run it to make sure that there are no errors.

Monte Carlo

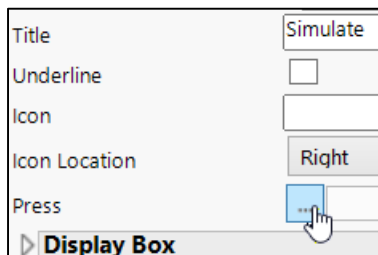
1. Add a Button Box to Module3 just above the **Response** column.



2. In the Properties pane for this button, change the title to “Simulate”



3. In the Properties pane of the Simulate Button Box, select the grey button next to the **Press** option to enter the following script. Note that some of the referenced variables (e.g. `::importlimitsbutton1`) are created in later steps.



```

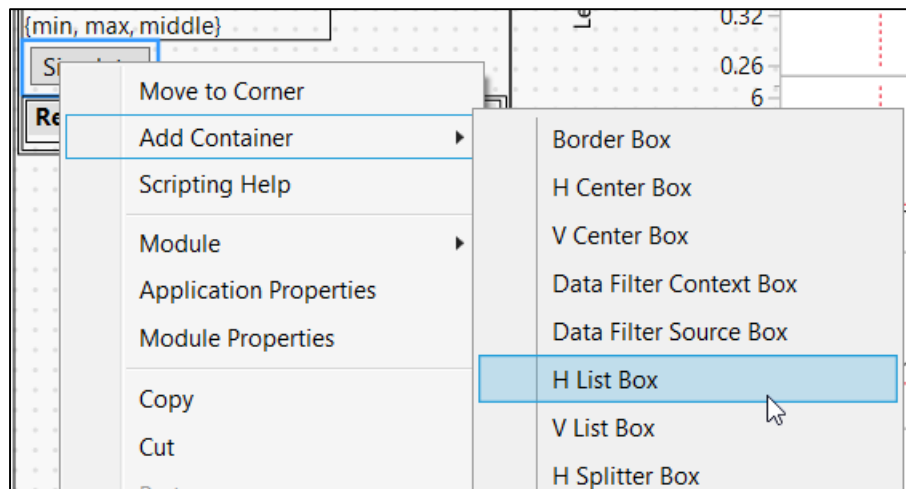
//Enable the buttons in the Contour Profiler Dashboard
::importlimitsbutton1 << enable( 1 );
::exportlimitsbutton1 << enable( 1 );
//record that the button has been pressed
sim.indicator=1;
//Turn on the custom profiler
pred.profl << custom profiler( 1 );
//Extract the factor names from the custom profiler display tree
domain1.factors = Report( pred.profl )["Custom Profiler"][String Col Box( 1 )] << get;
//save the responses (as a global variable, as indicated by ::)
::domain1.resp = Report( pred.profl )["Custom Profiler"][String Col Box( 2 )] << get;
//turn off the custom profiler
pred.profl << custom profiler( 0 );
//enable the Profiler Simulator
pred.profl << Simulator;
//Select a uniform distribution for each factor
For( i = 1, i <= N Items( domain1.factors ), i++,
    Eval( Eval Expr( (pred.profl << Simulator( Factors( Expr( domain1.factors[i] )
        << Random( Uniform() ) ) ) ) ) ) ) )
);

n.runs=number.of.runs<<get;
//Click the simulate Button
pred.profl << Simulator( Automatic Histogram Update( 1 ), N Runs( n.runs ), Simulate );
/* Click the Make Table button */ Report1["Simulate to Table"][Button Box( 1 )] << Click( 1
);

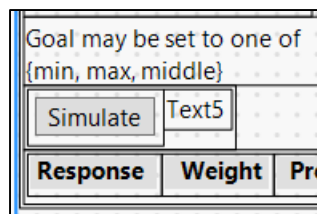
/* Assign a handle and a (global) name to the resulting table */
::dt.sim1 = Data Table( 1 );
//select all of the response columns
For( i = 1, i <= N Items( ::domain1.resp ), i++,
    Eval( Eval Expr( Column( ::dt.sim1, Expr( ::domain1.resp[i] ) ) << Set Selected( 1 ) )
)
);
dt.sim.selected1 = ::dt.sim1 << get selected columns;
::dt.sim1 << Set Name( "Simulated Table" );
//create the scatterplot matrix
Eval( Eval Expr( ::spml = ::dt.sim1 << Scatterplot Matrix( Y( Expr( dt.sim.selected1 ) ),
Nonpar Density( 1 ) ) ) ) );
//create a data filter for the simulated table
::dt.sim.filter1 = ::dt.sim1 << data filter;
::dt.sim.filter1 << set Show( 1 );
//reference the report (display tree) of the data filter
::dt.sim.filter.report1 = ::dt.sim.filter1 << report;
//no JSL analog to the Add button, so just click it virtually
::dt.sim.filter.report1[Button Box( 4 )] << Click( 1 );
//update the new data filter with the current contour plot limits
::exportlimitsbutton1 << click( 1 );
//wait for the previous commands to finish
wait(0);
//hide the simulated data table
::dt.sim1<<show window(0);
//disable the Profiler Simulator
pred.profl << Simulator(0);
//record that the button has been turned off
sim.indicator=0;

```

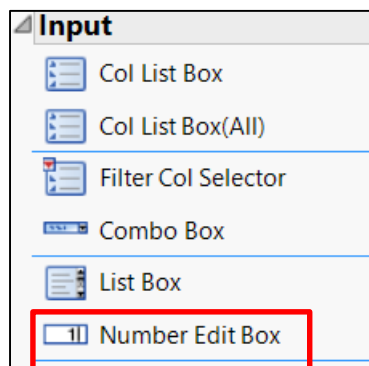
4. Right-click the **Simulate** button and select **Add Container > H List Box**.



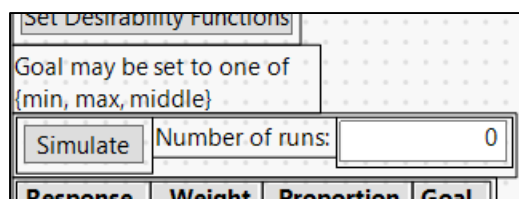
5. Drag a **Text Box** into the H List Box to the right of the **Simulate** button.



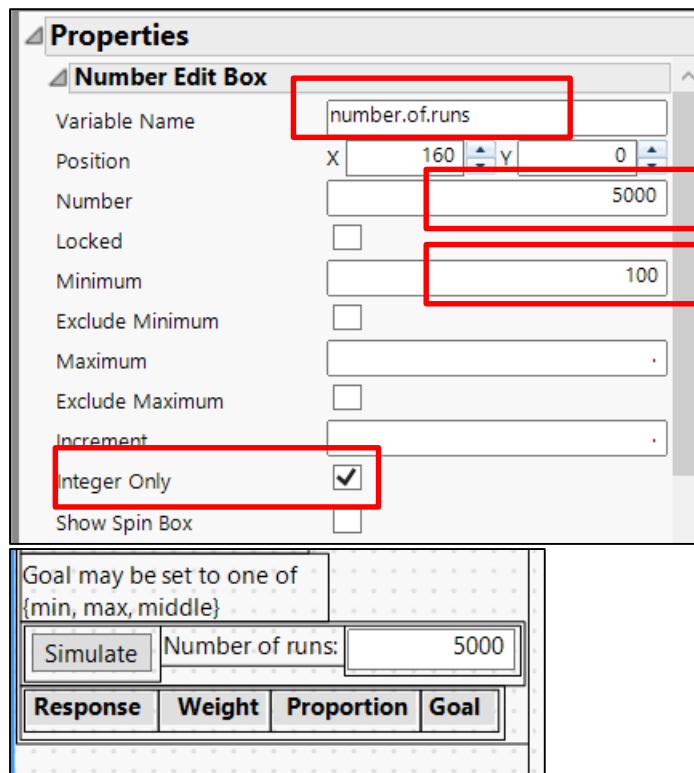
6. Change the **Text** entry of the Text Box in the Properties pane to “Number of runs:”.



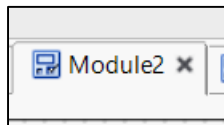
7. Drag a **Number Edit Box** into the H List Box to the right of the text box.



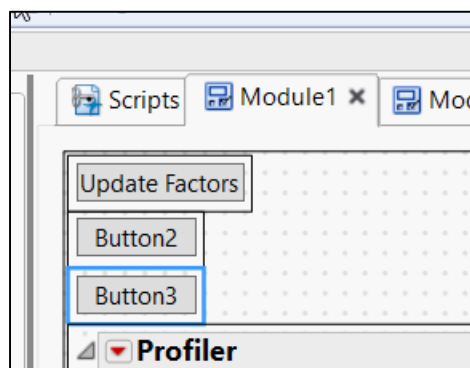
8. In the Properties pane of the Number Edit Box, set **Variable Name** to *number.of.runs*, **Number** to 5000, set **Minimum** to 100, and select **Integer Only**.



9. Click the Module2 tab.

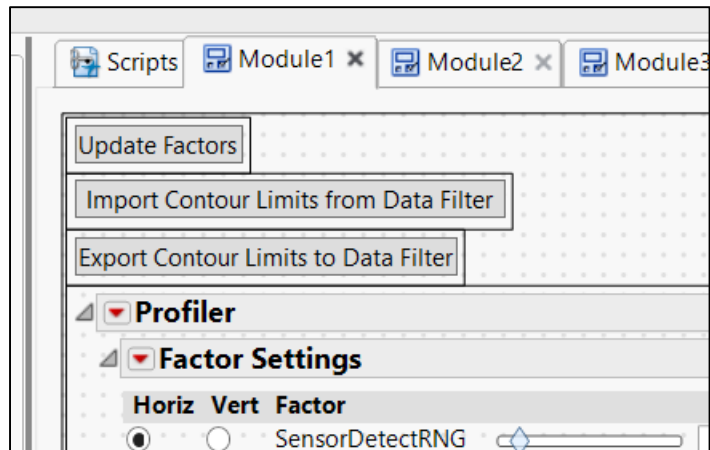


10. Under the **Update Factors** button, drag two new Button Boxes into the Lineup Box above the contour profilers.



11. Set the title of the top box to "Import Contour Limits from Data Filter" in the Properties pane.

12. Set the title of the bottom box to "Export Contour Limits to Data Filter" in the Properties pane.



13. Select the *Import Contour Limits from Data Filter* Button Box.
14. In the Properties pane for the *Import Contour Limits from Data Filter* Button Box, select the grey button in the **Press** area and enter the following script.

```

//Extract the upper limits from the data filter
upper.lim = {};
For( i = 1, i <= N Items( ::domain1.resp ), i++,
    Eval( Eval Expr( upper.lim = Insert( upper.lim,
::dt.sim.filter.report1[Number Edit Box( Expr( 2 * i ) )] << get ) ) )
);
//Extract the lower limits from the data filter
lower.lim = {};
For( i = 1, i <= N Items( ::domain1.resp ), i++,
    Eval( Eval Expr( lower.lim = Insert( lower.lim,
::dt.sim.filter.report1[Number Edit Box( Expr( 2 * i - 1 ) )] << get ) ) )
);
//Extract the response names from the data filter
names = {};
For( i = 1, i <= N Items( ::domain1.resp ), i++,
    Eval( Eval Expr( names = Insert( names, ::dt.sim.filter.report1[Text Box(
Expr( 3 * i ) )] << get text ) ) )
);
//loop over the contour profilers 1 through 4 (profiler1 simply contains the factor
settings)
//get the hi and lo limits and the corresponding response names from each Report
For( j = 1, j <= 4, j++,
    Eval( Eval Expr( report.names = Expr( Parse( "report(profiler" || Char( j
)||")" ) ) [String Col Box( 1 )] << get ) );
    Eval( Eval Expr( r2.ncb.lower = Expr( Parse( "report(profiler" || Char( j
)||")" ) ) [Number Col Edit Box( 2 )] << get as matrix ) );
    Eval( Eval Expr( r2.ncb.upper = Expr( Parse( "report(profiler" || Char( j
)||")" ) ) [Number Col Edit Box( 3 )] << get as matrix ) );
    For( i = 1, i <= N Items( names ), i++,
        Eval( Eval Expr( loc = Contains( report.names, Expr( names[i] ) ) ) );
        //update the corresponding values in lower.lim and upper.lim
        If( loc > 0,
            r2.ncb.lower[loc] = lower.lim[i];
            r2.ncb.upper[loc] = upper.lim[i];
        );
    );
    //update the hi and low limits in the contour profilers
    //make sure to use "set values" and not "set"
    Eval( Eval Expr( Expr( Parse( "report(profiler" || Char( j )||")" ) ) [Number
Col Edit Box( 2 )] << set values( r2.ncb.lower ) ) );
    Eval( Eval Expr( Expr( Parse( "report(profiler" || Char( j )||")" ) ) [Number
Col Edit Box( 3 )] << set values( r2.ncb.upper ) ) );
);

```

15. Click OK.

16. In the Properties pane for the *Export Contour Limits to Data Filter* Button Box, select the **Press** area and enter the following script.

```

//Extract the current upper limits from the data filter
upper.lim = {};
For( i = 1, i <= N Items( ::domain1.resp ), i++,
    Eval(
        Eval Expr(

```



```

        upper.lim = Insert(
            upper.lim,
            ::dt.sim.filter.report1[
                Number Edit Box( Expr( 2 * i ) )] << get
        )
    )
);
//Extract the current lower limits from the data filter
lower.lim = {};
For( i = 1, i <= N Items( ::domain1.resp ), i++,
    Eval(
        Eval Expr(
            lower.lim = Insert(
                lower.lim,
                ::dt.sim.filter.report1[
                    Number Edit Box( Expr( 2 * i - 1 ) )] << get
            )
        )
    )
);
//extract the response names from the data filter
names = {};
For( i = 1, i <= N Items( ::domain1.resp ), i++,
    Eval(
        Eval Expr(
            names = Insert(
                names,
                ::dt.sim.filter.report1[Text Box( Expr( 3 * i ) )] <<
                get text
            )
        )
    )
);
//loop over the contour profilers 2 through 5 (profiler1 simply contains the
//factor settings)
//get the hi and lo limits and the corresponding response names from each
//Report
For( j = 1, j <= 4, j++,
    Eval(
        Eval Expr(
            report.names = Expr( Parse( "report(profiler" || Char( j )||")" )
        )
    )
);
    String Col Box( 1 )] << get
);
    Eval(
        Eval Expr(
            r2.ncb.lower = Expr( Parse( "report(profiler" || Char( j )||")" )
        )
    )
);
    Number Col Edit Box( 2 )] << get as matrix
    Eval(
        Eval Expr(

```

```

r2.ncb.upper = Expr( Parse("report(profiler" || Char( j )||")" )
)
Number Col Edit Box( 3 )] << get as matrix
)
);
For( i = 1, i <= N Items(names ), i++,
Eval(
Eval Expr(
loc = Contains(
report.names,
Expr(names[i] )
)
)
);
//when a match is found, the data filter is updated
//the appropriate command for the data filter depends on whether
//there are missing values
//for the Hi or Lo Limits in the contour profiler.
//Note: due to lack of readability of if-then statements in JMP,
//the three conditions are
//listed separately in three if-statements
If(
loc > 0 & !(Is Missing( r2.ncb.lower[loc] )) &
Is Missing( r2.ncb.upper[loc] ),
lower.lim[i] = r2.ncb.lower[loc];

Eval(
Eval Expr(
::dt.sim.filter1 << (
Filter Column(
Column( ::dt.sim1, Expr( names[i] ) )
) << Where(
Column( ::dt.sim1, Expr( names[i] ) ) >=
lower.lim[i]
))
)
);
);
If(
loc > 0 & !(Is Missing( r2.ncb.upper[loc] )) &
Is Missing( r2.ncb.lower[loc] ),
upper.lim[i] = r2.ncb.upper[loc];

Eval(
Eval Expr(
::dt.sim.filter1 << (
Filter Column(
Column( ::dt.sim1, Expr( names[i] ) )
) << Where(
Column( ::dt.sim1, Expr( names[i] ) ) <=
upper.lim[i]
))
)
);
);
);

```

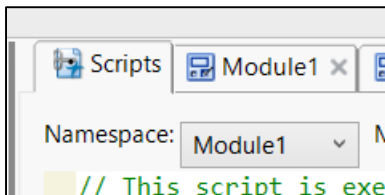
```

If(
  loc > 0 & !(Is Missing( r2.ncb.upper[loc] )) & !(
    Is Missing( r2.ncb.lower[loc] )),
  lower.lim[i] = r2.ncb.lower[loc];
  upper.lim[i] = r2.ncb.upper[loc];
  Eval(
    Eval Expr(
      ::dt.sim.filter1 << (
        Filter Column(
          Column( ::dt.sim1, Expr( names[i] ) )
        ) << Where(
          Column( ::dt.sim1, Expr( names[i] ) ) <=
            upper.lim[i] &
          Column( ::dt.sim1, Expr( names[i] ) ) >=
            lower.lim[i]
        ))
      )
    )
  );
);

```

17. Click on the Scripts tab.

18. Select *Module1* from the namespace dropdown.



19. Paste the following code at the bottom of the script.

```

::importlimitsbutton1=Button2;
::importlimitsbutton1<<enable(0);
::exportlimitsbutton1=Button3;
::exportlimitsbutton1<<enable(0);

```

```

Report1[number col edit box(1)]<<sib prepend(Number Col Box( "Min",mat.min),horizontal);
Report1[number col edit box(1)]<<sib append(    Number Col Box( "Max",mat.max),horizontal);

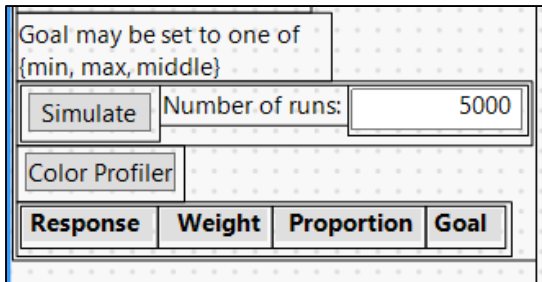
::importlimitsbutton1=Button2;
::importlimitsbutton1<<enable(0);
::exportlimitsbutton1=Button3;
::exportlimitsbutton1<<enable(0);

```

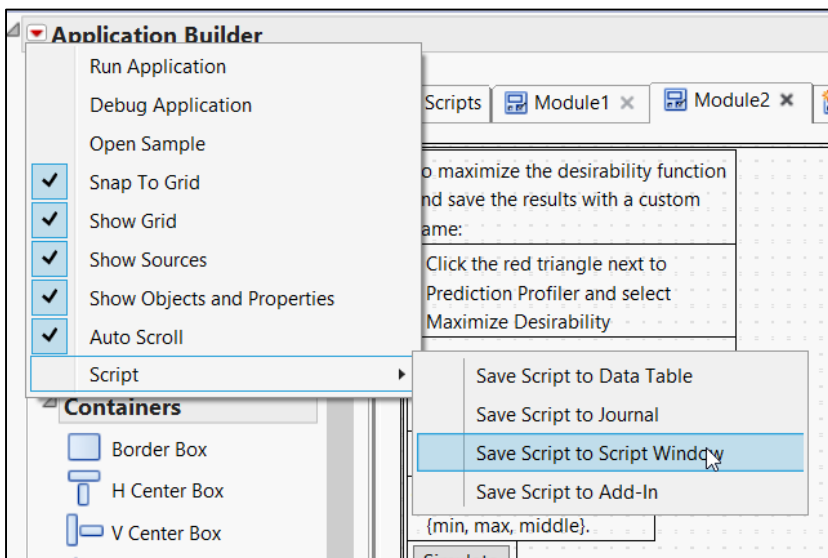
Color Profilers

1. Return to the Module 3 tab.
2. In Module 3, drag a new Button Box below the **Simulate** button.

3. Rename the Button Box “Color Profiler” in the Properties pane.



4. The next steps involving the Script Window are optional if the dashboard is being re-built for the MeanATKSamplev4 data set. They show how the code should be changed if building the dashboard for a new data set. Make sure that the **Script Window** is not currently open. Click the red triangle next to Application Builder and select **Script > Save Script to Script Window**. Unlike in previous steps where we referenced the script source of the application, we will not be clicking the run button this time to open a new Application Builder environment: we simply need to extract some of the code from the dashboard source.



5. Search for the string “pred.prof1 =” (without quotes, note the space before the equal sign).

```

StringEditCtrl = String Edit Box(
Report1 = Platform(
  DataTable1,
  pred.profl = Profiler(
    Y(
      :Awareness,
      :Lethality,
      :Casualties,
      :Sensory Weight,
      :Rifle Weight,
      :Radio Weight
    ),
    Profiler(
      1,
      Desirability Functions( 1 ),
      :Awareness << Response Limits(
        {Lower( 0.02, 0.066 ), Mid

```

6. The list of columns in the **Y()** section will be pasted into the resp.list variable in the code block below (step 12).
7. The list of columns in the **Reorder X Variables()** section will be pasted into the factor.list variable below.

```

),
:Rifle Weight << Response Limits(
  {Lower( 11.5, 0.9819 ), Middle( 13.75, 0.5 ), Upper( 15, 0.066 ), Goal( Minimize ), Importance( 1 )}
),
:Radio Weight << Response Limits(
  {Lower( 15, 0.9819 ), Middle( 21, 0.533461538461 ), Upper( 27, 0.066 ), Goal( Minimize ), Importance( 1 )}
),
Reorder X Variables(
  :SensorClassifyRNG,
  :RifleRNG,
  :RadioDelay,
  :SensorDetectRNG
),
Term Value(
  SensorDetectRNG( 2, Lock( 0 ), Show( 1 ) ),
  SensorClassifyRNG( 1.5, Lock( 0 ), Show( 1 ) ),
  RadioDelay( 7.5, Lock( 0 ), Show( 1 ) ),
  RifleRNG( 1.5, Lock( 0 ), Show( 1 ) )
)
),

```

8. Close the Script Window without saving. Nothing was modified.
9. Return to Module 3.
10. Select the Color Profiler button.
11. Click the grey button next to **Press** in the Properties pane (for the Color Profiler button).
12. Enter the following script. When building the dashboard for a different dataset, the highlighted code defining res.list and factor.list needs to be changed. These values can be found in the source of the Application Builder file (described in the previous steps).

```

//need to manually copy these column names from the pred.profl profiler in the
//application builder script
resp.list = { :Awareness,
              :Lethality,
              :Casualties,
              :Sensory Weight,
              :Rifle Weight,
              :Radio Weight};

//convert the column references to string column names
resp.name.list = {};
For( i = 1, i <= N Items( resp.list ), i++,
    resp.name.list = Insert( resp.name.list, resp.list[i] << get name )
);
//need to manually copy these column names from the pred.profl profiler in the
//application builder script
factor.list = { :SensorClassifyRNG,
                :RifleRNG,
                :RadioDelay,
                :SensorDetectRNG};

//convert the column references to string column names
factor.name.list = {};
For( i = 1, i <= N Items( factor.list ), i++,
    factor.name.list = Insert( factor.name.list, factor.list[i] << get name )
);
wait.window = New Window( "Running", Text Box( "Running. Please wait.
This window will close automatically when the coloring is complete.
Expected wait is ||char(ceiling(N Items( factor.name.list )*0.25+3))||" seconds." ) );
//create a temporary, invisible table that will be used to hold the modified factor
//settings. The formulas in the response columns are used to calculate the slope of
the
//line connecting the endpoints in each frame box.
dt2 = DataTable1 << Subset( invisible, Suppress formula evaluation( 0 ), Selected
Rows( 0 ), Rows( [1] ), Selected columns only( 0 ) );
wait(.1);
//add rows. There are two rows for each factor
dt2 << add rows( 2 * N Items( factor.list ) - 1 );
//wait(0) tells JMP to wait until the previous operation has completed before
moving on
Wait( .1 );
//extract the current factor settings from the profiler
factor.settings = {};
For( i = 1, i <= N Items( factor.list ), i++,
    factor.settings = Insert( factor.settings, Report1[Number Edit Box( i )] <<
get );
    wait(0);
);
//print the current factor settings to the log file
Print( "factor settings" );
Print( factor.settings );
//make sure the factor columns are unlocked, and set all of the rows
//in the temporary data table to the current factor settings
For( i = 1, i <= N Items( factor.list ) * 2, i++,
    For( j = 1, j <= N Items( factor.name.list ), j++,
        Column( dt2, factor.name.list[j] ) << lock( 0 );
    );

```

```

        Column( dt2, factor.name.list[j] )[i] = factor.settings[j];
        wait(0);
    )
);

factor.min = [];
factor.max = [];
//slope.matrix will hold the slopes of the lines connecting the endpoints
//of response profiles in the matrix of graphs displayed by the profiler
slope.matrix = J( N Items( resp.list ), N Items( factor.list ), 0 );
wait(0);
//extract the minimum/maxium factor values
For( i = 1, i <= N Items( factor.name.list ), i++,
    factor.min = V Concat( factor.min, Round( Col Min( Column( DataTable1,
factor.name.list[i] ) ), 3 ) );
    factor.max = V Concat( factor.max, Round( Col Max( Column( DataTable1,
factor.name.list[i] ) ), 3 ) );
    wait(0);
);
Print( "factor min" );
Print( factor.min );
Print( "factor max" );
Print( factor.max );
//each subsequent pair of rows in the temporary table is set to the
//current factor levels, with one of the factors set at its minimum/maximum
//values
For( j = 1, j <= N Items( factor.name.list ), j++,
    Column( dt2, factor.name.list[j] )[2 * j - 1] = factor.min[j];
    Column( dt2, factor.name.list[j] )[2 * j] = factor.max[j];
    wait(0.25);
);
//the wait function is necessary to make sure that the data table has time to
update the
//formulas for the response columns
Wait( 3 );
//calculate the slopes of the lines in each frame box, and save to the
corresponding
//location in slope.matrix
For( i = 1, i <= N Items( factor.name.list ), i++,
    For( j = 1, j <= N Items( resp.name.list ), j++,
        slope.matrix[j, i] = Column( dt2, resp.name.list[j] )[2 * i] - Column(
dt2, resp.name.list[j] )[2 * i - 1];
        wait(0);
    )
);
Print( "slope.matrix" );
Print( slope.matrix );
//nr and nc are the number of rows and columns in the matrix of graphs in the
profiler
//(the +1 corresponds to the desirability function graphs)
nr = N Items( resp.list ) + 1;
nc = N Items( factor.list ) + 1;
//the frameboxes are numbered down the columns in the profiler
//this matrix key tells us which frame box correponds to which
//component of the slope matrix

```

```

if(sim.indicator==0,framebox.key = Transpose( Shape( 1 :: (nr * nc), nc, nr )
);wait(0));
if(sim.indicator==1,framebox.key = Transpose( Shape( 1 :: ((nr+1) * nc), nc, nr+1 )
));
//shading.mat will be a scaled version of slope.matrix
//shading values of .99 produce a color that is light/transparent, while
//shading values around .45 are dark/opaque
shading.mat = J( nr - 1, nc - 1, 0 );
wait(0);
//within each row (within each response), the factors with the biggest slopes
//are shaded darker, while those with smaller slopes are lighter
For( i = 1, i <= (nr - 1), i++,
    max.in.row = Max( Abs( slope.matrix[i, 0] ) );
    Eval( Eval Expr( shading.mat[i, 0] = Expr( 0.99 - (Abs( slope.matrix[i, 0] )
/ max.in.row) * .55 ) ) );
    wait(0);
);
Print( shading.mat );
//the next loop colors the frameboxes
//the "goal" variable checks whether the goal is to maximize/minimize
//each response. Factors with positive (negative) slopes that are being maximized
(minimized)
//are colored green. Factors with negative (positive) slopes that are
//being maximized (minimized) are colored red. Zero slopes are colored black.
//If "goal" is set to "middle", the frame boxes for that response are white
For( i = 1, i <= (nr - 1), i++,
    For( j = 1, j <= (nc - 1), j++,
        goal = StringEditColl << get( i );
        wait(0);
        If( slope.matrix[i, j] < 0 & goal == "max" | slope.matrix[i, j] > 0 &
goal == "min",
            Eval(
                Eval Expr( Report1[framebox( Expr( framebox.key[i, j] ) )])
<< set background color( HLS Color( 0, Expr( shading.mat[i, j] ), 1 ) ) )
            );
            If( slope.matrix[i, j] > 0 & goal == "max" | slope.matrix[i, j] < 0 &
goal == "min",
                Eval(
                    Eval Expr(
                        Report1[framebox( Expr( framebox.key[i, j] ) )]) <<
set background color( HLS Color( .346, Expr( shading.mat[i, j] ), 1 ) )
                    )
                );
            If( slope.matrix[i, j] == 0,
                Eval( Eval Expr( Report1[framebox( Expr( framebox.key[i, j] ) )])
<< set background color( HLS Color( 0, 0 ) ) ) );
            If( goal == "middle"&slope.matrix[i, j] != 0,
                Eval( Eval Expr( Report1[framebox( Expr( framebox.key[i, j] ) )])
<< set background color( "white" ) ) );
            );
    );
);

```



```
close(dt2,nosave);  
wait.window << closewindow;
```

13. Save the Application Builder environment with a .jmpappsource extension to preserve the Application Builder environment, or with a .jmpapp extension for distribution.