2014

SUSTAINABLE LIFETIME VALUE CREATION THROUGH INNOVATIVE PRODUCT DESIGN: A PRODUCT ASSURANCE MODEL

K. Daniel Seevers
University of Kentucky, kdanseevers@gmail.com

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

K. Daniel Seevers, Student

Dr. I.S. Jawahir, Major Professor

Dr. James M. McDonough, Director of Graduate Studies

SUSTAINABLE LIFETIME VALUE CREATION THROUGH INNOVATIVE PRODUCT DESIGN: A PRODUCT ASSURANCE MODEL

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By
K. Daniel Seevers
Lexington, KY
Director: Dr. I.S. Jawahir, Professor of Mechanical Engineering
Co-Director: Dr. Fazleena Badurdeen, Associate Professor of Mechanical Engineering

Lexington, Kentucky

2014

ABSTRACT OF DISSERTATION

SUSTAINABLE LIFETIME VALUE CREATION THROUGH INNOVATIVE PRODUCT DESIGN: A PRODUCT ASSURANCE MODEL

In the field of product development, many organizations struggle to create a value proposition that can overcome the headwinds of technology change, regulatory requirements, and intense competition, in an effort to satisfy the long-term goals of sustainability. Today, organizations are realizing that they have lost portfolio value due to poor reliability, early product retirement, and abandoned design platforms. Beyond Lean and Green Manufacturing, shareholder value can be enhanced by taking a broader perspective, and integrating sustainability innovation elements into product designs in order to improve the delivery process and extend the life of product platforms.

This research is divided into two parts that lead to closing the loop towards Sustainable Value Creation in product development. The first section presents a framework for achieving Sustainable Lifetime Value through a toolset that bridges the gap between financial success and sustainable product design. Focus is placed on the analysis of the sustainable value proposition between producers, consumers, society, and the environment and the half-life of product platforms. The Half-Life Return Model is presented, designed to provide feedback to producers in the pursuit of improving the return on investment for the primary stakeholders. The second part applies the driving aspects of the framework with the development of an Adaptive Genetic Search Algorithm. The algorithm is designed to improve fault detection and mitigation during the product delivery process. A computer simulation is used to study the effectiveness of primary aspects introduced in the search algorithm, in order to attempt to improve the reliability growth of the system during the development life-cycle.

The results of the analysis draw attention to the sensitivity of the driving aspects identified in the product development lifecycle, which affect the long term goals of sustainable product development. With the use of the techniques identified in this research, cost effective test case generation can be improved without a major degradation in the diversity of the search patterns required to insure a high level of fault detection. This in turn can lead to improvements in the driving aspects of the Half-Life Return Model, and ultimately the goal of designing sustainable products and processes.

Keywords: Sustainable Value Proposition, Product Half Life, Sustainable Lifetime Value, Adaptive Genetic Search Algorithm, Product Assurance

K. Daniel Seevers

April 23, 2014

SUSTAINABLE LIFETIME VALUE CREATION THROUGH INNOVATIVE PRODUCT DESIGN: A PRODUCT ASSURANCE MODEL

By

K. Daniel Seevers

Dr. I.S. Jawahir
Director of Dissertation

Dr. Fazleena Badurdeen
Co-Director of Dissertation

Dr. James M. McDonough
Director of Graduate Studies

April 23, 2014

I dedicate this dissertation to my parents, children, and loving wife.

ACKNOWLEDGMENTS

I wish to thank those who helped me through my journey to complete this dissertation.

It has been an honor to be advised by Dr. Jawahir, who brings such a wide scope of knowledge to the table. He has been integral to the advancement of research in the fields of Mechanical Engineering, Manufacturing Processes, and Sustainability. My time spent with Dr. Jawahir discussing these topics has been invaluable.

I would also like to thank my Co-Advisor, Dr. Badurdeen. Her feedback, hard work, and knowledge were outstanding aids in my journey. I offer thanks as well, to my other committee members, Dr, Keith Rouch and Dr. Thomas Goldsby, for their perspective, support, and insight.

The tools and education I have been blessed with throughout my career have been the direct result of talented people who have been willing to take the time to teach, provide feedback, and debate with me. Although there are too many to individually name, I thank them all.

Finally I thank my family. My world is built upon the foundation my parents, Ken and Doris Seevers, provided to me. My energy comes from the love and support of my wonderful wife Denise, and the joy my children, Kyra and Rachel, bring to me every day.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: Introduction

1.1 Background

Technology advancements and new innovations continue to fuel the fast pace of new product introductions available to consumers around the world. In 1965, Gorden E. Moore predicted the number of transistors on integrated circuits would double every two years (Moore, 2006). Today his relatively accurate prediction, Moore's Law, serves as a symbolic backdrop for the exponential growth of consumer electronics as well as design evolutions in the majority of industrial categories. With each new product introduction, consumers are presented with possibilities for increased productivity, improved communications and information flow, and improved quality of life (Malik, 2013; Friedman, 2005). But, with the ever increasing hunger for products that increase consumption of the worlds natural resources, questions arise of how to measure the benefits new technology brings to humankind vs. the potential wake of waste streams left in its path. The challenging concept is balancing the e-gain benefits from new technology vs. the e-waste of abandoned products (Figure 1).

The phrase "the world is now connected" refers to the explosion of electronic technology that allows consumers around the world to participate in the digital age of communications and computing (Lessig, 2002; Mulgan, 2011). With the advancement of satellites, cell towers, increased micro-processor speeds, advanced electronics and software, information and new solutions are connecting individuals across oceans. New solutions and product innovations in areas such as health, education, transportation, and engineering tools are enabling societal gains in all parts of the world. These **"**e-gains**"** are driving new benefits to international consumers. At the same time, the pace of the new technology advancements is growing exponentially and providing consumers with a constant flow of new choices. These choices are often at the expense of the current solution and are creating a waste stream of old hardware.

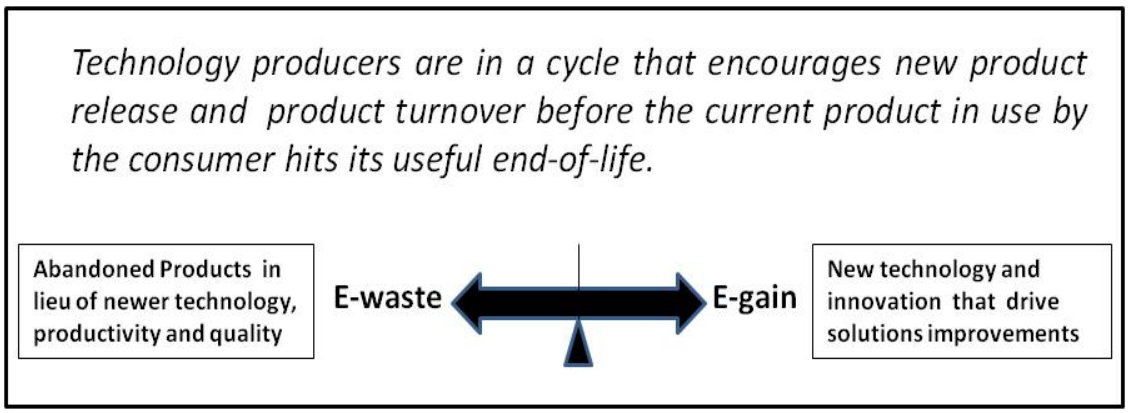


Figure 1.1: The sustainable product development conundrum

New business opportunities for material recycling or re-purposing have grown recently, yet simultaneously stock piles of consumer electronic waste have also grown. These bi-products have been labeled as **"**e-waste**".** There is a need for research that analyzes the drivers of a product lifetime in relation to the balance between e-gains vs. e-waste. Whereas the fuel for this conundrum comes from technology advancements and personal gains, engineers should pay attention to all forces that motivate consumers to abandon the use of a product before its designed useful end of life. These forces include product defects, excessive warranty costs or product downtime, and lack of function relative to new options in the market (Widmer et al., 2005).

In order to model or predict the success of a particular design in the field, one must look beyond the internal definition of product value, and integrate the reality of what the customer values. If you ask a businessman whether or not a particular product line is sustainable, the answer may lie in the context of financial gains they may extract from the customer relationship over time. This drives the analysis of Customer Lifetime Value in the effort to increase shareholder value. Ultimately, in order to affect the sustainability of a particular product design in the field, a broader perspective is required during the product development cycle to enable the creation of sustainable value over the lifetime of the product platform (Figure 1.2).

The traditional definition of customer satisfaction focuses on the ability of the producer to develop a solution in the form of a value proposition meeting customer expectation over time. One of the primary drivers in the proposition is the perceived value, a dynamic variable constantly affected by external factors. In order to improve product development to reduce the amount of e-waste relative to the e-gains of new technology, there is a need for research in the field of sustainable product development. This is accomplished by integrating sustainability concepts into product design tools in order to drive value over the life-cycle of the product platform and the lifetime of relationship with the consumer. This research defines the result of this activity as Sustainable Lifetime Value (SLV).

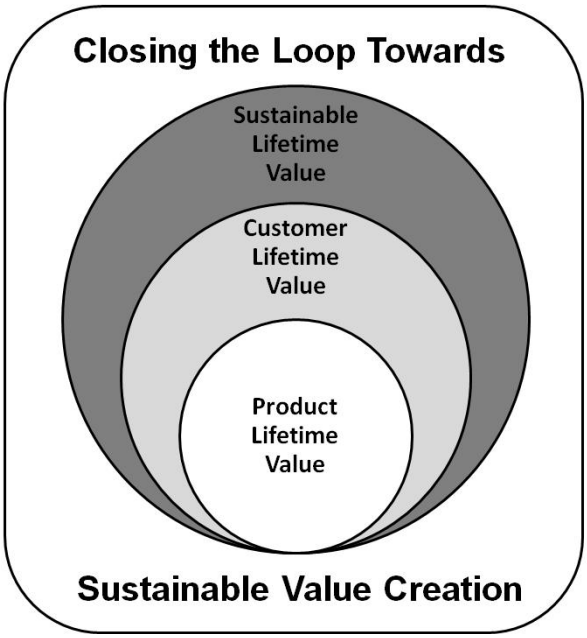


Figure 1.2: Product and Customer Lifetime Value models are core to Sustainable Lifetime Value Creation

1.1.1 Sustainable Lifetime Value Creation through the Design of Sustainable Products

In free enterprise business environments, the primary objective of corporations today is to maximize the return on investment (profit) subject to constraints. The company's shareholders are the ultimate residual claimant because they provide the required financial investment for development and operations (Jensen and Meckling, 1976; Eccles et al., 2012). For decades, there has been a debate on the effects of adopting corporate social responsibility and sustainability practice and the cost or benefit of these actions. Some scholars have argued that adopting such practices will destroy shareholder wealth (Friedman, 2007; Navarro, 1988). At the heart of these researchers' position is that employees of a business are responsible to their employers not to society. Recently, there is some evidence that companies that have a corporate culture that embraces sustainability may actually outperform similar companies in increasing shareholder value (Eccles et al., 2012; Linnenluecke and Griffiths, 2010). In reality, the dynamics of competing in a business world are complex and business models are constantly changing. The preferences of individual customers widely vary and corporations must choose the particular value proposition they will offer to the consumers and against the competition.

To assume the sustainable practices will only act as a tax can lead to takes a narrow view of the topic. In reality, it is possible to create a sustainable value proposition in a new product design (relative to the previous offering) that increases value to the *consumer and producer* and reduces environmental impact. By taking a broader view and integrating the drivers of sustainability into the product development process, sustainable lifetime value is created. Continued research and tool development is needed to aid the design engineer in bridging the gap between traditional financial models and models that take advantage of the time value of resources over multiple product life-cycles (generation to generation design improvements – see Figure 1.3).

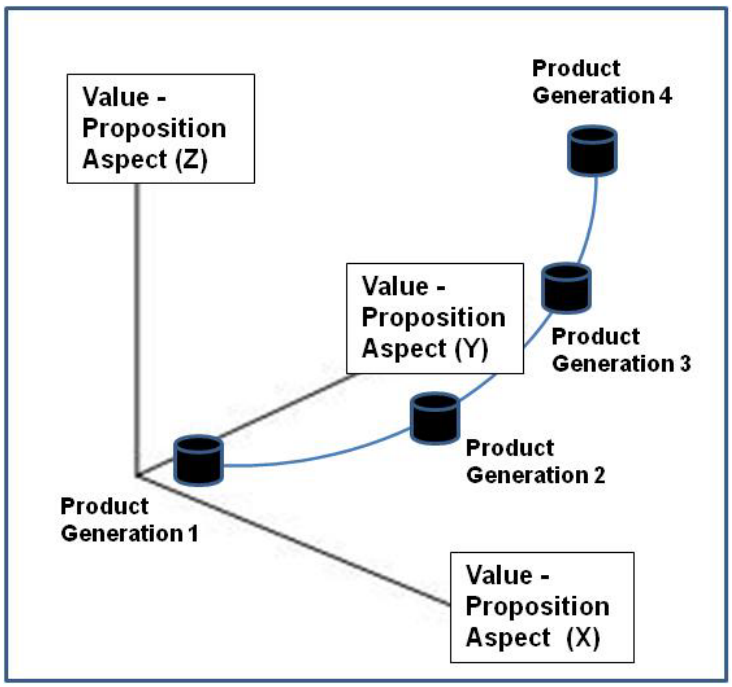


Figure 1.3: Driving the sustainability value proposition into future design generations

1.1.2 Sustainable Value Creation Models

From a return on investment and sustainability perspective, the long term goal of the development engineer is to increase shareholder value by improving generation to generation product designs focused on sustainable lifetime value . Unfortunately this is a dificult task due to the trade-offs within complex system designs. Competing values in combination with complex product definitions, make sustainability model development in the area of product design, dificult.

Research in the field of value creation and the development of sustainable products and processes should include the study of complex systems. In part, there is need to breakdown the complex problem into manageable aspects. Ueda et al. (2009) described the goal of Sustainable Value Creation as a complex problem. Beyond a producer creating an artifact that they feel the consumer will value, values are “co-created” through interaction among systems including natural systems. Longevity or product half-life is not only affected by design attributes such as specifications, but is also affected by

how society accepts and advocates for the new technology. Depending on societal trends, new product introductions can be either slow to succeed or the product can experience excess inertia. Ueda et al. (2009) also presented value creation models based on emergent systems and co-created decision making. They studied the relationships between natural, social, and artifactual systems. In related research, Tolio et al. focused on the complexity of economic, socio-political and technological dynamics (Tolio et al., 2010).

This dissertation research is in the field of sustainable products and process development with a focus on the key aspects that create sustainable lifetime value. One of the motivations for this research is to provide the engineering community a set of tools that bridge the gap between product design and financial deliverables. The first step is to redefine the traditional product value proposition to include the driving cost aspects of the major stakeholders in sustainable development (Figure 1.4).

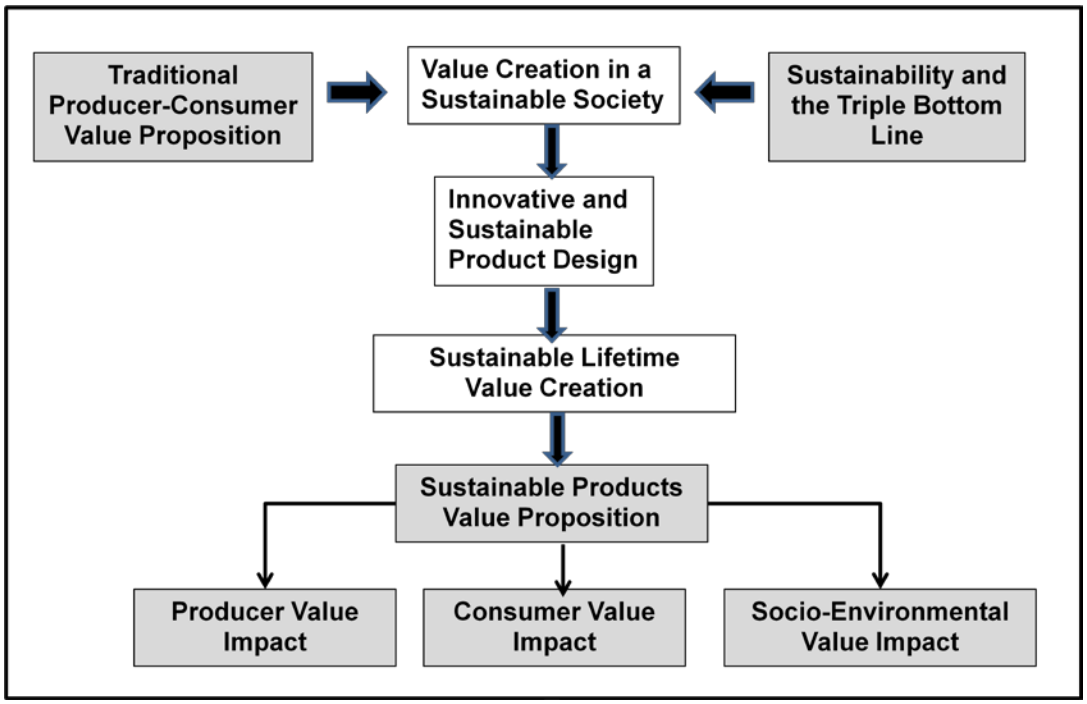


Figure 1.4: Sustainable Value Creation framework for products

Just as the introduction of new technology to society is a complex problem, the development and success of new products can also be complex. With the integration of new hardware, firmware and software, products today are integrated complex systems that provide solutions to consumers and society. Developing tools for the engineering community that reflect this reality is the key to bridging the gap between the time value of money and time value of resources. There is a need for additional tools in the engineering tool kit to help engineers transition the concept of a sustainable value proposition into the physical design.

With the help of NGO's, industry representatives, and government employees, influence on the long-term effects of sustainable products has continued to increase in some industries. The potential for even greater value creation is not only possible, but necessary in order to improve sustainability in products from generation to generation. At the heart of this proposition is the creation of greater value between consumers and producers, to achieve societal and environmental benefits.

There is extensive research in the field of product development and customer satisfaction that analyze the potential profit of particular product design. For example, Customer Lifetime Value (CLV) models have been developed to help producers (companies) develop business models that analyze the profit per customer of a product line over the life of a platform (Reinartz and Kumar, 2003). This traditional model is built on the fundamentals of financial theory and the time value of money. In addition, another product development business model, the Return Map, calculates the return on investment vs. the cost and time to develop the new product (House and Price, 1991). These two standard product development feedback models are focused on the producer/consumer relationship, but lack the integration of the basic sustainable product development concepts.

This research is unique in that it deals with the fundamental weaknesses of these models from a sustainability perspective and focuses on identifying critical aspects of

the development process to integrate them into a framework that aids in the design of sustainable products.

1.2 Research Outline

This research is divided into two parts that lead to closing the loop towards Sustainable Value Creation in product development. The first section presents a framework for achieving sustainable lifetime value through a toolset that bridges the gap between financial success and sustainable product design. The second section applies the framework, focusing in on the roles verification, risk and resource management play in the development on sustainable products.

1.2.1 Closing the Loop towards Sustainable Lifetime Value Creation

Ultimately, the goal of this research is to create a framework for the engineering community that helps close the loop towards Sustainable Lifetime Value Creation in product design. Four primary elements are presented in a concept model focused on this goal (Figure 1.5).

1. **Key Stakeholders:** In order to create a sustainable value proposition that will lead to value creation, the primary stakeholders are identified.
2. **Product Development Process Drivers**: Six primary drivers of activity in the development process are identified for the engineering community, that affect the sustainable lifetime value metrics.
3. **Development Process Integration**: The six drivers are presented in an integrated format to emphasize the symbiotic relationship necessary for increasing the return on investment.

4. **Sustainable Value Life-Cycle Metrics**: A new set of metrics focused on time value of resources is identified that aids in the analysis of return on investment for sustainable product development.

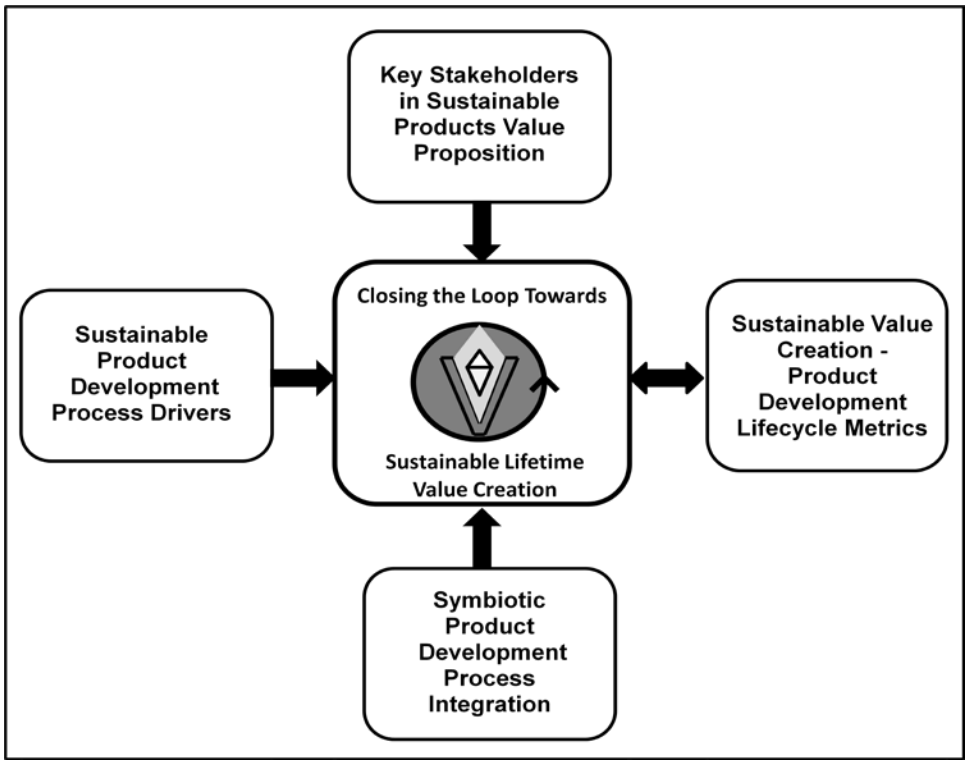


Figure 1.5: Concept model of the integrated sustainable product development framework

Fundamentally the study of product lifetime value is introduced as measured by the product half-life relative to related development metrics. This research establishes the Sustainable Products Half-Life Return Model to integrate data sets from product development life-cycles and the product platform lifetime. To bring focus to the problem, the concept of an expanded sustainable value proposition is introduced where the high impact drivers for each pillar of the proposition are identified. In doing so, the design engineer will have a set of metrics that will aid in value creation in generation-to-generation product development. Finally, a set of primary drivers in sustainable product development are integrated into a tool set designed to aid the engineering team during

the product development life-cycle. Robust design practices are critical to improving the lifetime value of a product design but the complexity of product verification and feedback during the design process can be just as important to Product Lifetime Value, Customer Lifetime Value and ultimately Sustainable Lifetime Value. The integration of risk and resource management, along with fault detection and mitigation are levers that drive improvement in the sustainable products design model.

Whereas the long term benefits of improving the sustainable value proposition will include the integration of total cost with social and environmental factors, research focused on the extension of product half-life, material utilization and development resource optimization will play a major role in sustainable product development.

1.2.2 Reliability Assurance Model for Sustainable Product Development

The second part of this research applies the aspects of the integrated product development framework for fault detection and mitigation process during the product development life-cycle. By introducing risk and resource management (cost) into the fault diagnosis process, improvements can be driven into the key metrics of the sustainable product Half-Life Return Model. This research identifies five aspects of the fault detection process and applies them into a model designed to improve the effects of test case development by the System Product Assurance Engineers. The results of this model are used to draw generalization about these effects on the creation of Sustainable Lifetime Value.

A critical, yet often overlooked aspect of product development is testing, verification and product assurance activities. Unfortunately some products, including consumer electronics, have become so complex that traditional product assurance and reliability engineering processes cannot adequately predict the system reliability, or average life of a product. With the integration of hardware with firmware and software, the number of

system combinations requiring traditional product verification testing is impossible. In essence, if the goal of the reliability engineer is to test every design combination, the problem becomes intractable. Today, some complex systems are shipped to customers with a projected failure rate at the start of production (Tassey, 2002). The societal costs of these escapes, along with the current expense rates of product verification, create the need for advancements in process and tool development.

Recently there have been advancements in research focused on fault detection and test case generation using heuristic techniques. (Cohen et al., 2003; Watkins et al., 2002; Baudry et al., 2005) These new fault detection algorithms are primarily in software development which does not present the same difficulty as verifying the combination of hardware, firmware and software. Because of the possibility of latent and interactive defects in hardware systems, as well as the potential for multiple defects related to one sub component in a complex system, subsystems and interactions must be continually monitored in the verification process.

In the fields of reliability engineering and system assurance, the science of test case (for fault detection) development, with problem resolution management vs. risk analysis and management, is typically managed independently with separate data and value streams. This gap prevents the opportunity to focus verification resources on the test combination with the highest potential payback. In addition, time to market and limited testing resources can be a critical factor that affects verification strategies.

The second part of this research is the development of a broader adaptive algorithm that can integrate the search for functional defects, interactive defects, and latent defects embedded in a complex system. In addition, this fault diagnosis process is focused on the characteristics of a complex system that integrates hardware, firmware, and software into one system to test. By introducing test case cost, a verification budget, and detected fault risk value into the algorithm, the ability to increase the lifetime value of the product and shareholder value of the producer will improve. By

focusing on the primary drivers of the Half-Life Return Model, the ability to create sustainable lifetime value is also enabled.

Whereas the long term benefits of improving the sustainable value proposition will include the integration of total cost as well as social and environmental factors, research focused on the extension of product half-life, material utilization and development resource optimization will play a major role in sustainable product development.

1.3 Chapter Summary

This dissertation is presented in two parts (Figure 1.6). The first part is focused on Sustainable Lifetime Value Creation in the pursuit of developing sustainable products. A concept model and analysis metrics are presented which can be adapted for specific industries. In Chapter two, first, a literature review is presented in the area of Sustainable Value Creation, product delivery, and background material. Next, tools are presented which are designed to aid the development engineer in the design and delivery of products that improve the sustainable value proposition. These concepts are packaged into an integrated framework to address Sustainable Lifetime Value Creation.

The second part of this dissertation is focused on some of the key drivers introduced in the integrated framework. In particular, it introduces the use of feedback during the development life-cycle to increase the lifetime value of the product. Chapter three is dedicated to the problem definition and hypotheses used to research and design a solution that assists the development team in the verification of product designs. Chapter four presents a literature review and supporting background on feedback and the application of the integrated framework. Chapter five presents an adaptive genetic search algorithm designed to improve complex system fault detection modeling and application of the integrated framework. Chapter six presents a case study that exercises the search algorithm and sustainable value proposition metrics. In the simulation, a designed experiment is used to evaluate the independent affects and interdependence of the controlled test case model variables. These results are

presented in Chapter seven along with discussion. Chapter eight concludes the dissertation with a summation and discussion of the potential for future work.

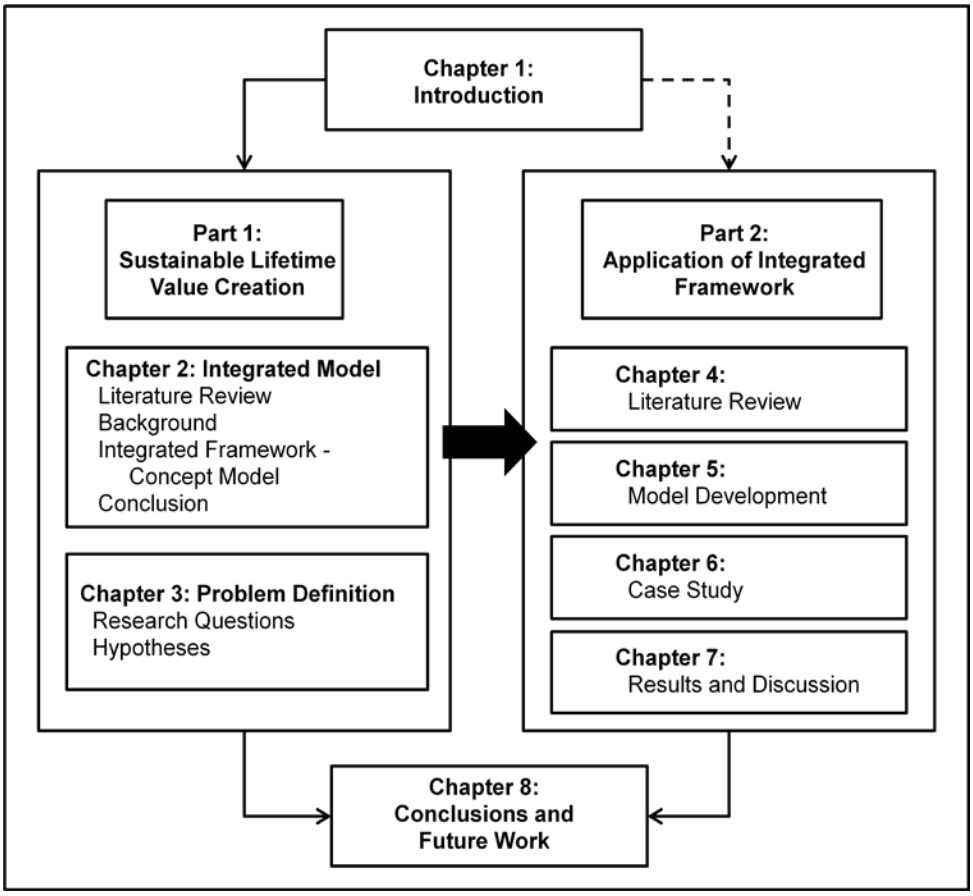


Figure 1.6: Dissertation chapter outline

Part 1: Sustainable Lifetime Value Creation

Chapter 2: Sustainable Lifetime Value Creation: Integrated Model

This chapter is focused on creating a model that bridges the gap between product development and value creation to aid in the design of sustainable products and processes. The first section provides a literature review and background information on the broader concepts of sustainable product development and the gaps experienced by development engineers in their attempt to bridge the relationship between sustainable concepts and new product delivery processes. The second section identifies the key aspects of a new value proposition and integrates those aspects into a framework designed to aid in maximizing the return on investments associated with product development aimed at the proposition. The goal of this research is to introduce the concept of the time value of resources into the delivery process to drive toward the creation of sustainable lifetime value. This creation is accomplished by integrating sustainability concepts into product design tools to drive value over the life-cycle of the product platform and the lifetime of the relationship with the consumer.

2.1 Literature Review

2.1.1 Sustainable Development

With continuous growth in the world's appetite for new products and services, the rapid consumption of the earth's natural resources and the pressure of this growth on social and environmental systems, fuels the desire and need for research in sustainable product development. The effects of this research on the actions and processes of specific industries and corporations are varied depending on the region, type of product, and individual motivation of the organization. Drivers of process change include regulatory requirements, corporate social responsibility, or even physical reminders

such as the reality of post-use waste streams such as e-waste. Inherent in the problem solving processes used to create new solutions, engineers do not set out to develop products that add undo waste. In fact, corporate product development processes seek delivery efficiencies and final designs that meet the expected value proposition. The concept of sustainable product development is logical, but the definition and understanding of this concept is varied and can be confusing to the engineering community. This confusion is due to the subject of sustainability being a broad and multidisciplinary topic.

System Dynamics

Early research in the field of system dynamics were born out of the recognition that the consumption of the world's natural resources and the effects of industrial growth on ecosystems were at a pace that would eventually outpace the supply. In an effort to cross boundaries, global think tanks and organizations began to discuss the topic. For example, a group called "The Club of Rome" released a report titled "Limits to Growth" to draw attention to related issues (Peccei, 1981; Meadows et al, 1972). In this report, a multidisciplinary group identified five variables (world population, industrialization, pollution, food production, and resource depletion) that should be analyzed within one system in order to gain insight into a model that would aid in future development. Systems theory research attempted to create models to represent these complex problems. Jay Forrester is credited with developing models in system dynamics and was invited by The Club of Rome to attempt to model the five variables identified in their report (Forrester, 1971). He created a mathematical model intended to predict the behavior of the complex interactions in the five variables over time. These models were originally titled World1 and World2 and, since that time, additional research has focused on improvement of the model. This line of research was aimed at modeling the behavior of vast systems over time, but did not necessarily provide a clear link between economic and social benefits of development.

Definition of Sustainable Development

One of the most recognized attempts to draw attention to sustainable development was initiated by the General Assembly of the United Nations in a report by the World Commission on Environment and Development, titled “Our Common Future” (Burton, 1987). In this report, it was pointed out the the words “environment” and “development” had definitions and connotations that could lead to narrow interpretation. In addition, the commision implied that the two words should be considered inseparable because the ”environment” is where we live and “development” is what we all do to improve our lot within the environment. Development within an environment can be described as an economy. Therefore, the commision defined sustainable development as economic developmment that meets the needs of present generations without compromising the ability of future generations to meet their own needs (Vágási et al., 2003). In order to bridge the gap between the Brundtland Commission and business and economic theory, J. Elkington introduced the term “The Triple Bottom Line” as an advancement of the traditional financial business term “ the bottom line”, wherein the end corporations are in business to make profits (Elkinington, 2004).

Financial management literature began to integrate traditional economic and financial goals with environmental and social concerns. The triple bottom line creates a triad of macroeconomic concerns between social, economic, and environmental aspects (McDonough & Braungart, 2002) (Figure 2.1). These early attempts to bridge the gap between business and sustainability integrated economic concepts into the discussion, but still lacked a cohesive bridge between sustainability and new product development. There is a need for further research and tools to aid the engineering community in closing the loop between the life-cycle of product development and broader life-cycles of the environment and society.

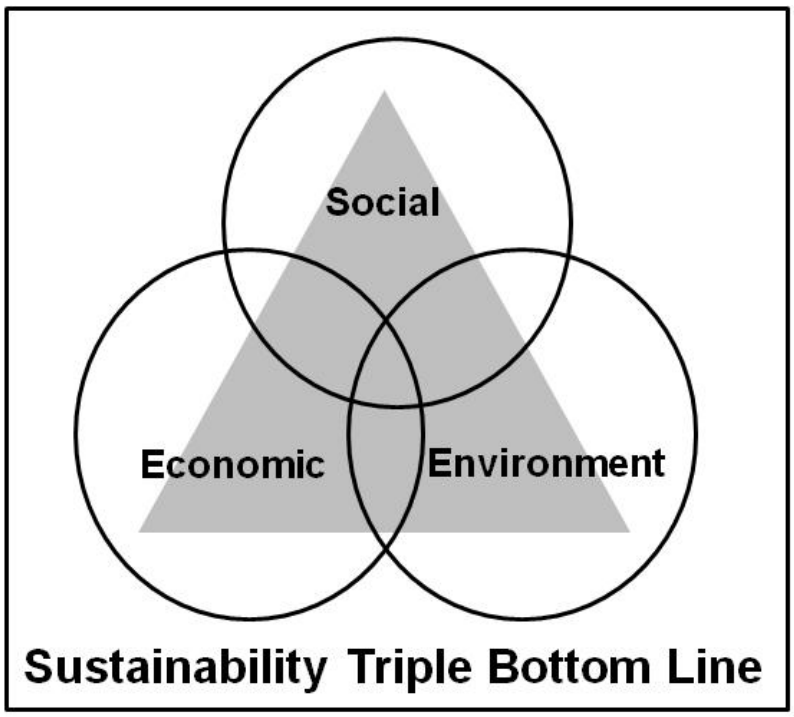


Figure 2.1: The triple bottom line of sustainable development (Elkinington, 2004)

Sustainable Manufacturing

Whereas the Brundtland Commission established a broad definition of sustainable development, there has been focused research in the definition of sustainable manufacturing and product development (Jayal et al., 2010). Adapting descriptions from the US Department of Commerce and the National Council for Advanced Manufacturing, Jawahir and Jayal (2011) conducted research in the field of sustainable manufacturing that was focused on the development of sustainable products and processes. While maintaining and/or improving the product and process quality**,** the earlier definition of sustainable manufacturing is expanded to cover the following five expectations:

- demonstrate reduced ***negative environmental impact,***
- offer improved ***energy and resource efficiency,***

- generate ***minimum quantity of wastes,***
- provide ***operational safety,*** and
- offer improved ***personal health***

while maintaining and/or improving the product and process quality.

These focus areas serve as a general list for a development team to consider during the design life-cycle. In addition to the list above, research on taking an extended view of the product life-cycle can also aid in the sustainability of products and processes.

Early attempts to focus on closing the loop on a product life-cycle were coined with the term the "3R's". People were encouraged to reduce their consumption, reuse their products if possible, and finally recycle the material at the end of the product life (Gehin et al., 2008). The majority of product development research centered on the 3R's is focused on lean and green manufacturing (Metta, 2011). However, in order to drive towards sustainable manufacturing, innovation-based approaches that extend the 3R's further are encouraged. These approaches include introducing the capability to recover end-of-life products and materials, redesign and remanufacture the next generation products over multiple life-cycles and utilize the recovered materials (Figure 2.2), (Joshi et al., 2006; Jawahir and Dillon, 2007).

This research seeks to extend the concept of integrating innovative drivers into the development process in order to assist the engineer in designing more sustainable products.

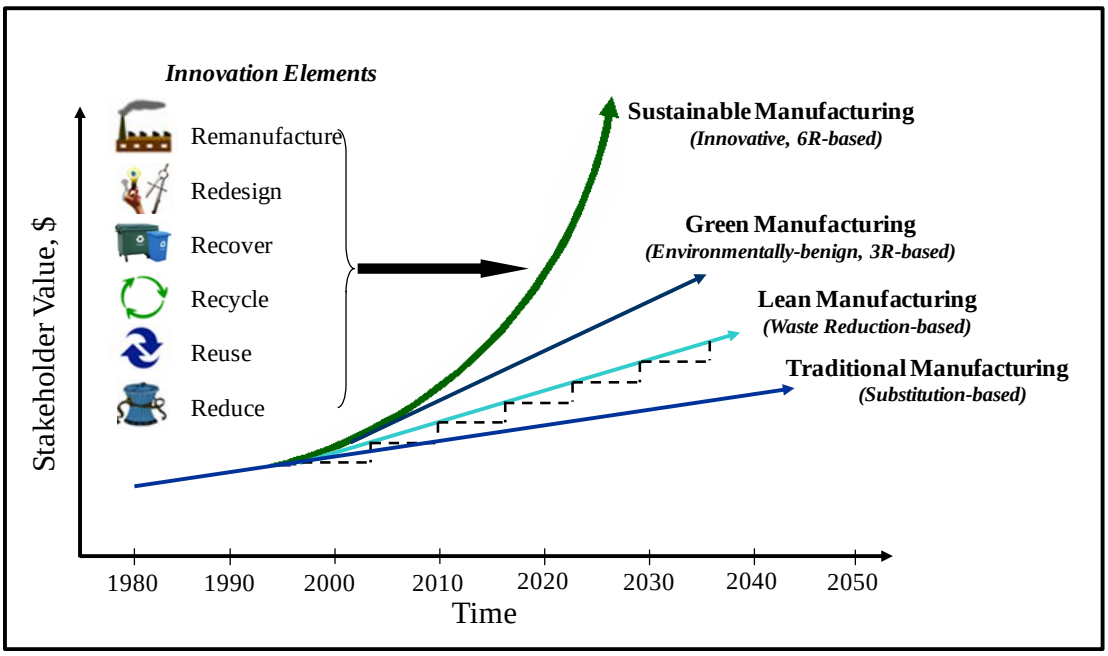


Figure 2.2: The "6R's of Sustainable Manufacturing are designed to increase stakeholder value (Jawahir and Dillon, 2007)

E-WASTE and Product Utilization

As a matter of strategy, engineers do not set out to design new products for the sake of creating waste. In fact, producers face a new product development conundrum: Technology producers are in a cycle that encourages new product release and product turnover before the current product used by the consumer hits its useful end-of-life. In order to draw attention to the research necessary to help improve the development of sustainable products and processes, particularly from a waste stream perspective, the perceived value should be well-understood and addressed.

A familiar saying in commercial enterprise is “time is money”. Courses in engineering economics introduce financial concepts to their design engineers as they contemplate cost, expense, and time as part of their overall solution. Given this background, engineers may struggle to develop the financial connections between time, money (commerce), and sustainability. In its basic form, finance theory uses interest (rates) to form the fundamental concept of time value of money. (Crosson and Needles, 2008)

When developing new tools in sustainability for the engineering community, this basic concept still holds, but further analysis is warranted around the term value. From an engineering perspective, this research broadens the concept to “time value of resources” and lifetime value.

The ultimate measure of the lifetime value of a product and consumer satisfaction may be the actual utilization of the product over an extended period of time. The study of profit over a product life-cycle through customer utilization and revenue is sometimes referred to as Customer Lifetime Value- CLV (Berger and Nasr, 1998). This dissertation draws attention to the gap between research addressing Customer Lifetime Value and Sustainable Value Creation. By focusing on improving the sustainable value proposition to the consumer and meeting the intended design of the development team, the product utilization (reducing early product withdrawal) and warranty rates will improve. This broader scope of sustainable product development will assist in the improvement of inefficient consumption and help reduce the creation of its byproducts, such as electronic waste.

To illustrate the effects of early product withdrawal, the study of the half-life of product families is introduced (Figure 2.3). The half-life is defined as the point where half of the products sold within a product platform (model family) have been retired and are no longer used in the market. The graph presents models of relative half-life estimates for various types of material goods (Seevers et al., 2013).

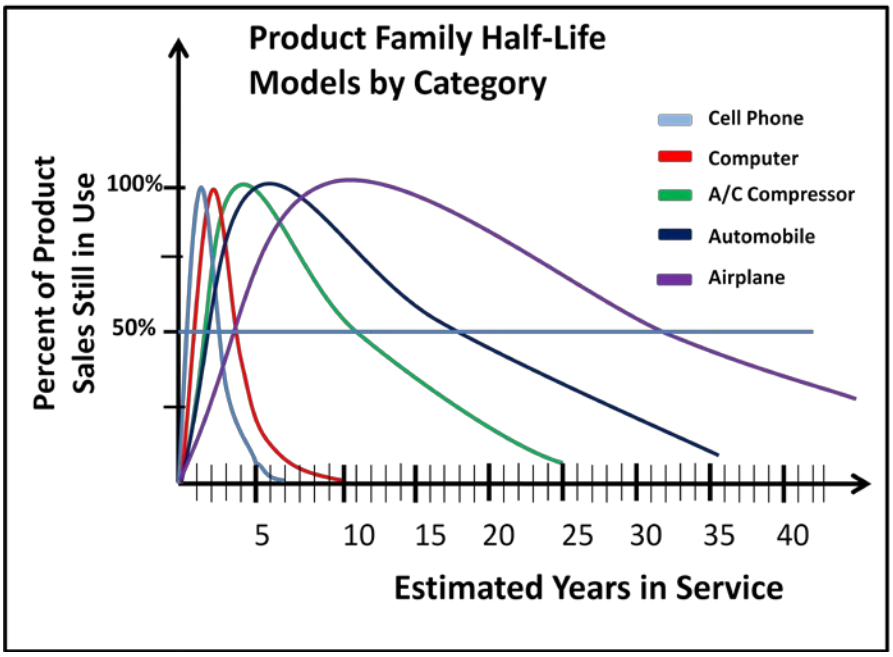


Figure 2.3: Relative product half-life estimates of selected product families (Seevers et al., 2013)

The chart exposes the challenges consumer electronics producers and other high technology industries face, where it is possible that the half-life of a product family is shorter than the time it takes to develop the product. When product half-life data is superimposed on product financial models, even greater insight on the potential risk of early product abandonment is possible. The details behind these dynamics can aid in research toward the development of sustainable products and processes. The ability to predict product life due to changing market conditions is an important aspect of sustainable product design and manufacturing. There is a gap in research addressing modeling product life that takes into consideration the sustainable value of the products from the perspective of society and the environment.

2.1.2 Traditional Half-Life Modeling

Customer Turnover and Product Churn

Just as research in the field of sustainable manufacturing examines the value of a product over potentially multiple lifetimes, research in business administration studies the value of retaining customers over time. Studies show that the initial cost to attract new customers is typically higher than the cost to retain customers. This phenomenon is why customer satisfaction is critical to long term profitability of businesses, particularly those whose value proposition includes annuities or contract services (Reinartz and Kumar, 2003). In their research, Reinartz and Kumar (2003) identify the impact of customer relationship metrics on lifetime profitability and create a model to analyze resource allocation and balance between competing expenses in marketing organizations for optimal profit. The industry term used to identify a customer who ends their financial relationship with a producer and stops utilizing their product is called Churn. Conversely, when a producer adds a new customer (a consumer of their product line), this is referred to as Lift**.** Churn and Lift models are a primary method for businesses to predict their future profit and shareholder value in relation to their product portfolio and value proposition (Fader and Hardie, 2007).

As the new product development conundrum points out, customer and product churn creates a strain on both producers and the environment. Abandoned products are taken out of service before the intended useful life has been achieved. This problem is heightened in the electronics fields due to many reasons, but technology advancements continue to force reconsideration of the potential value proposition of the old product vs. the next generation. Beyond consumer electronics, the creation of waste streams due to churn can affect many other industries too. Because of this, there has been increased research in the field of predictive churn models in relationship to the projected Customer Lifetime Value for the producer.

Braun and Schweidel (2011) point out that there are two major research categories related to this topic. The first category focuses on Customer Lifetime Value in relationship to the time when a customer decides to terminate a relationship (Fader et al., 2009; Rosset and Neumann, 2003). Models are created with the intent to calculate the time before churn. The researchers also point out that these models do not focus on why they move on, just that they have moved on. The other category of research in this marketing field focuses on reasons why a customer may choose to stop use of the product or service (Schweidel et al., 2008). While these models are useful, they do not account for the complexity of competition in the market place and do not address the issue of competing causes for churn. Braun and Scheidel focus their research on linking the different reasons for which customers churn to the value they provide to the producer. In essence, they develop probability of surviving (a given time period) with multiple risks for organizations to calculate the expected Customer Lifetime Value of their product line. The logic used to create this model, starting with survival probability due to a single event working up to the survival probability due to competing events, is shown in Figure 2.4 and described below (Braun and Scwheidel's (2011)).

Survival Probability

The premise for calculating the lifetime profit a business may achieve for a particular product line starts with connecting the survival rate of keeping the customer set. The survival probability (*S*) is a function of hazard rate (*H*) of the studied data set over time (*t*).

Expected Customer Lifetime Value

In order to manage the data, the survival of the customer set is broken down into time segments. After each time frame, the number of customers remaining is calculated. The expected is therefore dependent on survival of each period.

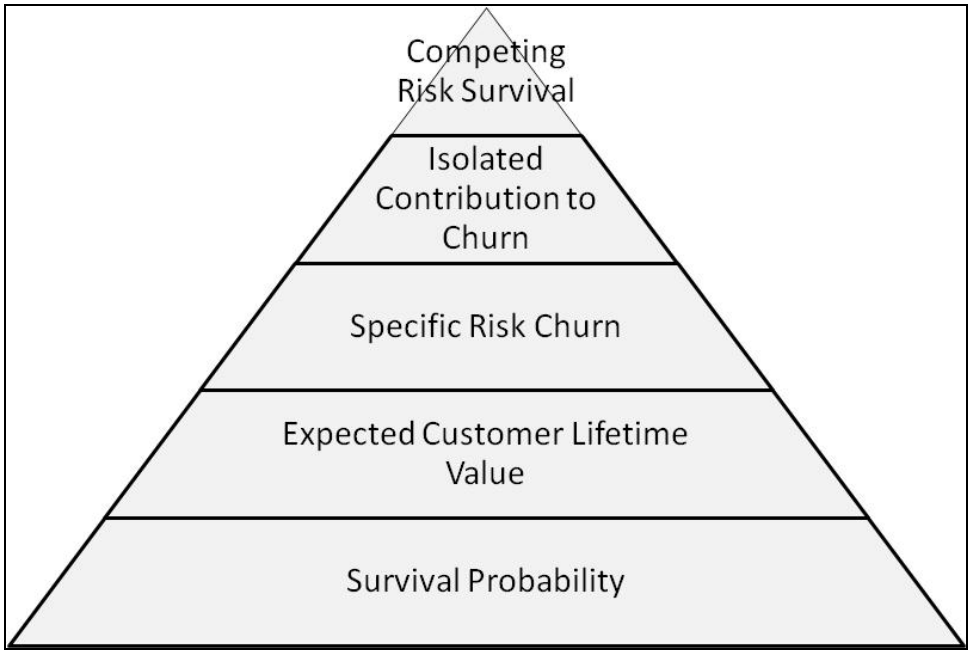


Figure 2.4: The progression of logic to establish CLV via multiple churn risks

Specific Risk Churn

A customer may choose to stop using a particular product for a number of reasons. The ability to break a problem down into multiple effects increases the quality of a model. The model generates information about the effects of delaying churn attributable to a specific risk.

Isolated Contribution

The likelihood of a single contribution to a single individual is the final building block to modeling the potential risk to profit due to a lost customer.

Competing Risks Survival

Ultimately the probability of survival with competing risks is derived from the logic of the previous sections. Braun and Scwheidel's (2011) research derived the mathematical equations for this logic.

Challenge to model correlation between churn and sustainability

Previous research by Braun and Schweidel (2011) provides a promising model of projecting Customer Lifetime Value with the possibility of multiple causes of churn. The researchers' definition of Customer Lifetime Value is only from the producers' perspective with regard to potential lifetime profit per a given customer. This narrow definition does not take into consideration sustainability aspects such as the cost of this churn to consumers, society, or the environment. In essence, if a consumer "churns" and moves on to another solution, the waste stream of the abandoned solution is considered a sunk cost. Sunk costs are expenses that are not recoverable, are attached to opportunity costs from past decisions, and are not considered relevant to future decision making (Schmalensee, 2004). This definition is strictly from a financial business perspective and may under appreciate the time value of resources from a consumer, societal, or environmental perspective.

In order to expand the definition of value creation to include the longer term perspective of product development over potentially multiple physical life-cycles, research in the field of value creation must include the driving aspect of sustainability. The research in this dissertation will create a framework to transition from Customer Lifetime Value to Sustainable Lifetime Value (SLV). An integrated framework of sustainable product development drivers is presented in order to address the long term value proposition. The effects of this research are presented in a model that integrates financial product success with the longevity of the product life and measured by the product half-life.

2.1.4 New Product Delivery and Return on Investment

Within free enterprise markets, many companies have been identified who are focused on increasing shareholder value and simultaneously focused on sustainability and corporate social responsibility. There are certification programs that exist, such as

Energy Star and Blue Angel, that give producers the opportunity for brand recognition in return for meeting higher standards for product designs that help consumers and the environment (Brown et al., 2002; Hemmelskamp and Brockmann, 1997). In addition, opportunities are identified for investors who seek out corporations that perform well relative to performance metrics in economic, environmental, and social categories. The Dow Jones Sustainability Index (DJSI) and the United Nations Global Compact 100 stock index are two examples of indices that rank stocks based on performance in environmental and social issues as well as financial results.

Research in the field of Environmental Economics is a branch of sustainable development that is focused on the effects of the economy on the environment. In David A. Anderson's book titled Environmental Economics and Natural Resource Management , he takes the position that the economy is a subset of the environment (Anderson, 2013). In related work, Whitehead and Haab (2012) suggest adding the external costs of production to the internal bill of materials. From an economic theory perspective, the added cost (nicknamed a pollution tax) will, in effect, raise the cost of the product and, therefore, lower the demand. These models are focused more on material consumption and do not analyze the three mutual aspects of the sustainable value proposition (consumer impact, producer impact, and socio-environmental impact). The success of a product with new technology is a complex problem, which may or may not take into consideration the environmental effects. By looking at the mutual value between the three driving aspects, product success can drive sustainable lifetime value and shareholder value through improved return on investments.

A Balanced Approach to Product Design

There is another line of research that draws attention to the integration of ecology and the environment into the product development process. Because of the complexity of designing products for a sustainable world, engineers are required to make competing trade-offs in the development process. Traditional tools developed to aid the engineer

in decision tradeoffs and generation-to-generation design comparisons include the Quality Function Deployment and The Pugh Concept Selection Methodology (Prasad, 1998; Hauser; Kerscher, 1993; Pugh and Clausing, 1996). Recently, efforts have been made to introduce the concept of sustainability into these types of tools which are mainly focused on life-cycle integration into the visual tools (Ramani et al., 2010; Devanathan et al., 2010). These new tools are considered to be in the field of eco-design but do not necessarily focus on the aspects of sustainable value in the product development process. In order to take advantage of concept selection type sustainable product development tools that also integrate value creation analysis, the competing factors of this complex problem should be identified.

The Return Map

In free enterprise markets, the majority of new technology and product development is conducted with the goal of profit, portfolio growth, and customer satisfaction. There are tools used by development community program managers that aid in the financial analysis of a product line to provide feedback to the design teams. An example of such a tool was developed by Charles H. House and Raymond L. Price. They labeled it “The Return Map” (Figure 2.5). This tool integrates the concepts of development time (expense) along with sales. Emphasis is placed on break even points and return on investment (House and Price, 1991).

One of the primary reasons House and Price developed this tool was to provide marketing R&D and manufacturing a common standard of measurement to shorten the development cycle for improved return on investment (ROI). In essence, the Return Map captures both money and time in one space. Three primary sets of data (investment expense, sales revenue, and profit) are plotted out on a timeline. The primary data sets are:

Product Research Time: The initial time (investigation phase) and research expense used to investigate new technology and potentially new product to deliver to the market.

Product Development Time: This is defined as the "Time-to-Market**"** or the amount of time (and development expense) required developing the concept into the final design ready for production.

Investment Expense: Producers may choose to track research expense separate from product development expense, but, in general, the investment expense is the amount of money required to fully develop and produce a product for market. Some producers track the physical product material expense (a.k.a. bill of material – BOM expense) separate from the development expense.

Sales: In the Return Model, sales are tracked as total revenue in the product family sales.

Profit: Profit is the positive gain from sales (revenue) after subtracting for all expenses.

Break-Even-Time (BET): Starting the clock when the team begins the investigation of a new product, the breakeven point is where the cumulative profit line crosses over the cumulative investment expense line. The breakeven time is, therefore, the amount of time elapsed during the product life-cycle before the cross over point is met.

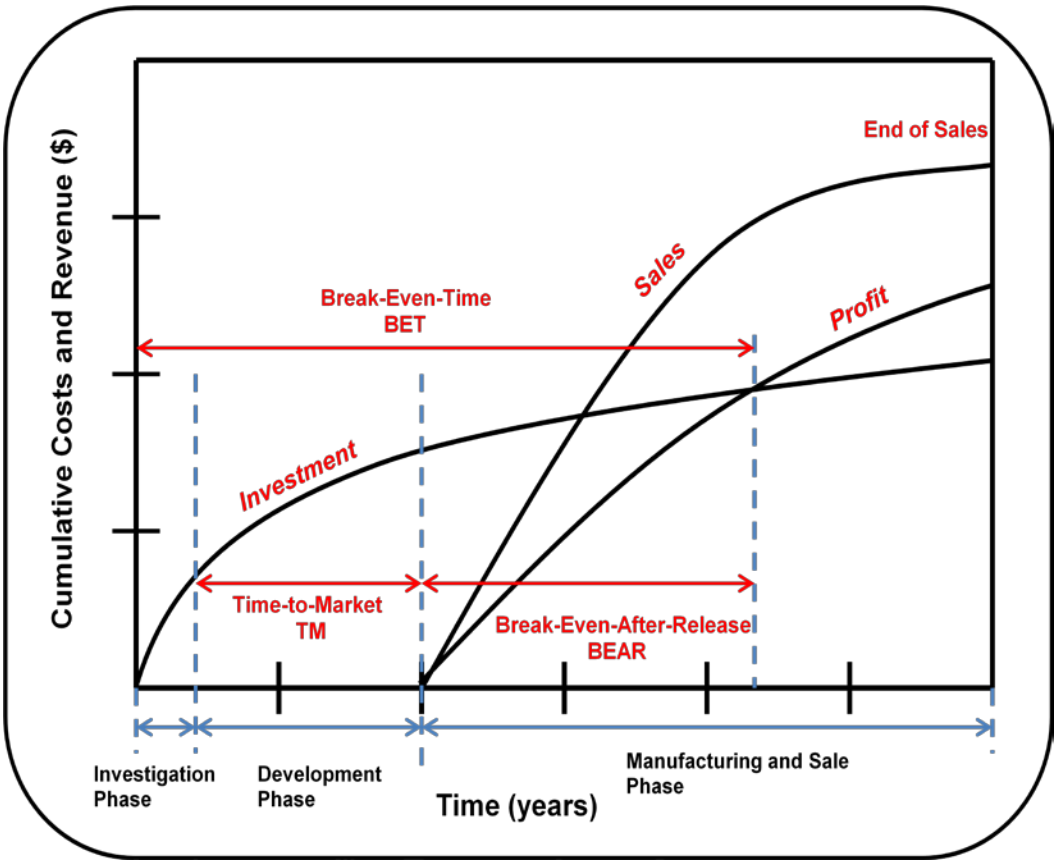


Figure 2.5: The product development Return Map by House and Price (1991)

Break-Even-After Release (BEAR): Likewise, break-even-after-release refers to the amount of time (and investment expense) elapsed between the start of engineering development and the breakeven point. This is a critical metric in the effort to drive improved collaboration between R&D, marketing, and manufacturing as well as reducing development time.

Return Factor (RF): This is the calculation of profit divided by the investment at any specific point in time after the product has started production (SOP) and the beginning of sales.

With focus on producer collaboration and driving shorter development cycles, the Return Map is a valuable development tool designed to aid the engineering community in increasing portfolio value for their investors. As worldwide competition continues to

put greater pressure on engineers and, ultimately, the product life-cycle, viewing data from both a money and time perspective provides insight into the producer value proposition.

Product Half-Life and Sustainability

Since the development of the original model by House and Price (2011), greater focus and public demand is being placed on issues of sustainability and the development of sustainable products and processes. One of the drawbacks of the Return Map model is that time ends on the last day of product sales. The model does not take into account the value of the product line in future time, which should be integrated into the sustainable value proposition. This dissertation introduces a model that focuses on the total life-cycle of the product family in order to draw attention to the key aspects that can be fed back into the overall solution. One of the primary metrics used to track the success of a product over time is the product half-life. This is the point in time where half of the products sold within a product platform (model family) have been retired and are no longer used in the market. Product half-life is a form of measuring customer churn during the life-cycle of the product platform.

In order to extend the producer value proposition into the field of sustainability, total life-cycle costs for the consumer and indirect socio-environmental costs are integrated into the formula. As noted earlier, this is not an easy task. In the field of product development, many organizations struggle to create a value proposition that can overcome the headwinds of technology change, regulatory requirements, and intense competition in an effort to satisfy the long-term goals of sustainability. By focusing on the half-life of the product in the field, producer collaboration, and developing life-cycle time/expense, progress will be achieved in the pursuit of sustainable product development.

Product Delivery Process

Producing sustainable products requires an integrated approach between product, process, and system design (Jayal et al., 2010). Just as process plans are created and followed in an effort to run efficient manufacturing facilities, development teams also follow a process in order to design products that drive the team towards maximum return on investment. This section focuses on product delivery processes used by engineering teams.

Ultimately, the success or failure of a product rests on the ability of the engineering team to coordinate and, at times, govern activities to deliver a quality design that meets or exceeds the value proposition. From a business perspective, quality, cost, and delivery (QCD) are still the cornerstones by which an engineering team will be measured (Akao, 2004).

One of the more recognizable product development systems is built on the fundamentals of QCD, surrounded by a total quality management (TQM) mindset. Taiichi Ohno is considered the father of the Toyota Production System, which is the basis for broadly accepted lean manufacturing methods (Womack et al., 2007; Kennedy and Ward, 2003). The cohesiveness of this culture within the Toyota Corporation was so successful and standardized that it became integrated into the product development process. In their research, Morgan and Liker (2006) point out that the lean manufacturing methods were taught around the world and no longer afforded the Toyota Company exclusive reliance on its benefits. They also point out that Toyota, at the time of their writing (2006), still had an advantage because lean methods were integrated into their product delivery process. Whereas the Toyota Corporation is known for controlled growth, some models described the need for advanced focus on competing values of innovation to succeed in markets that have advancing technology and competition (Thakor et al., 2000; Cameron, 2006). The difficulty with relying on

incremental improvement is that the process will not allow the development team to look beyond the tangential change in order to consider step function improvements.

It is important to point out that in free enterprise markets, the forces of intense competition and technology growth will place a continuous forcing factor on what is considered best practices. Yamaji et al. (2011) point out that, in the midst of rapid globalization and worldwide quality competition, Japanese manufacturers are struggling for the realization of "simultaneous achievement of QCD". They define it as the reduction of the product development life-cycle, continued assurance of high quality, and production at low cost. Even the most recognized companies will continue to feel competitive pressure to reduce development time and expense, but, at the same time, delivery of the highest value proposition to the consumer. When this is achieved, the longevity of the product platform is increased and drives toward Sustainable Value Creation and investment portfolio growth.

Beyond lean and green manufacturing, focus needs to shift toward integrating sustainability into every aspect of the product delivery process (PDP). Just as every corporation or development team could be described by their own specific culture, every team also has their own development process. It is unrealistic to subscribe to one development process for all industries; each producer has a different set of goals, resources, risk aversion level, competitive environment, maturity, and capital investment intensity. That being said, each team is still driven by a core set of financial, physical, and coordinated activities integrated in to one superset referred to as the product delivery process (PDP).

From an engineering perspective, a process implies a set of actions that may or may not be interdependent to transform a system (Martin, 2000). It is assumed with a repeatable process that the output of the system is predictable when given a particular set of inputs and actions. Some processes, like cooking recipes, are passed along generation to generation, but the intention is to keep the process exactly the same. The reality of large scale producers, who experience worldwide competition for their

product line, is the product development process must be continuously improved. This is not to imply that the best way to shorten the time it takes to deliver a product design is to cut corners on the development process. Yet, this can be the natural result of such a goal. In their research authors Wilson et al. (1996), draw attention to the need for study of the product development process along with the pitfalls of ignoring the issue. The researchers describe the term “organizational amnesia”, whereby the majority of the product knowledge and delivery process is contained within individuals instead of a systematic, company-wide process. The risk of this type of culture includes the lack of sharing of best practices and the potential loss of process knowledge if the employee leaves the team. Their work is an example of research focused on developing an integrated, functionally balanced product development process.

One of the most traditional product development process methods is based on breaking the delivery process into major phases (Crowe and Feinberg, 2001). Typical process phases in order include: conceptual (idea) development; concept evaluation; development and design consolidation; system test; and, finally, product manufacturing. To add process structure to these phases, checkpoints at the start of each phase are conducted. These phase gates (also called stage gates), along with the specific design practices within the phases, together can be considered a product development process. This type of development process is also traditionally referred to as the waterfall process.

One of the most popular practices designed to shorten the product development process is referred to as “Concurrent Engineering” (Ma et al., 2008). The goal of concurrent engineering is to integrate functional areas involved within the delivery team earlier in the process. This includes members from areas such as manufacturing and service which traditionally did not get involved until later in the process. By addressing functional needs earlier and potentially avoiding re-design or bottlenecks, the elapsed time to deliver the final system is reduced.

These types of delivery processes are well established and have served their purpose well. One of the major drawbacks of the waterfall delivery method is the lack of emphasis on the role that design verification plays early in the process. Traditional models do not emphasize system verification until the final phase before the hand off to the manufacturing team. As the complexity of new product designs continue to grow, especially with the integration of software and firmware with hardware, the ability to discover system interaction faults in a short amount of time or with limited testing budgets, is becoming an intractable problem.

2.1.5 Current Model Limitations

As focus on worldwide resource consumption in congruence with the headwinds of intense competition and product turnover grows, engineers are looking for a tool set that bridges the gap between traditional design methods and sustainability. While no two industries or environments are the same, research is necessary to develop a number of new models and tools to aid the engineering community.

Standard product development feedback models, including those presented by Reinartz and Kumar (2003) and House and Price (1991), are focused on the producer/consumer relationship, but lack the integration of the basic sustainable product development concepts. A common limitation of current product development models is that return on investment is narrowly focused on the time vale of money and does not take into consideration the time value of resources over potentially multiple product life-cycles. In fact, many financial models consider a customer that chooses to stop using a product as a sunk (investment) cost to the producer and do not consider the burden that product churn can place on producers, consumers, society, and the environment (Schmalensee, 2004). In the Return Map - Product Development Feedback model by House and Price (1991), time and the value of the product ends on the day of the last product sale. Their model does not consider the potential value of the product over the

entire lifetime of the product platform or the effects that the lifetime value have on the development life-cycle business model.

Traditional product churn models fail to break the problem down beyond the current product sales cycle, lack analysis of the rate of product churn relative to the amount of useful product life remaining, and do not take into consideration the complexity of high technology products and competing fault types that reduce the useful life of a product in the field. This research is also focused on the role that verification plays on the creation of value during the product development process. A limitation of traditional test and verification strategies is the treatment of fault detection and risk management as two separate processes, thereby limiting the ability to use risk and resource consumption in the verification process. The product development process is enhanced when the design team considers the sustainable value proposition aspects within each design life-cycle. Ultimately sustainability is achieved by closing the loop and repeating the process over the product lifetime (Figure 2.6).

2.2 Concepts Relevant to Closing the Loop towards Sustainable Value Creation

Creating sustainable products, especially an in an area that has exponential growth in new technology, is a complex task. Engineers struggle to make the connection between sustainability, the value proposition, and shareholder portfolio value growth. By introducing the concepts outlined in this section, this research is building a foundation for closing the loop toward Sustainable Value Creation.

Producers are making decisions that need to take into account the continuity of current successful solutions vs. new technology. This foundation is built on the goal of integrating producer, consumer, and socio-environmental aspects into one value proposition. On this foundation, research must continue to develop new tools for the "engineering tool box" to guide them in the development process.

In order to expand the definition of value creation to include the longer term perspective of product development over potentially multiple physical life-cycles, research in the field of value creation must include the driving aspect of sustainability. The research in this dissertation will create a framework to transition from Customer Lifetime Value to **Sustainable Lifetime Value (SLV).**

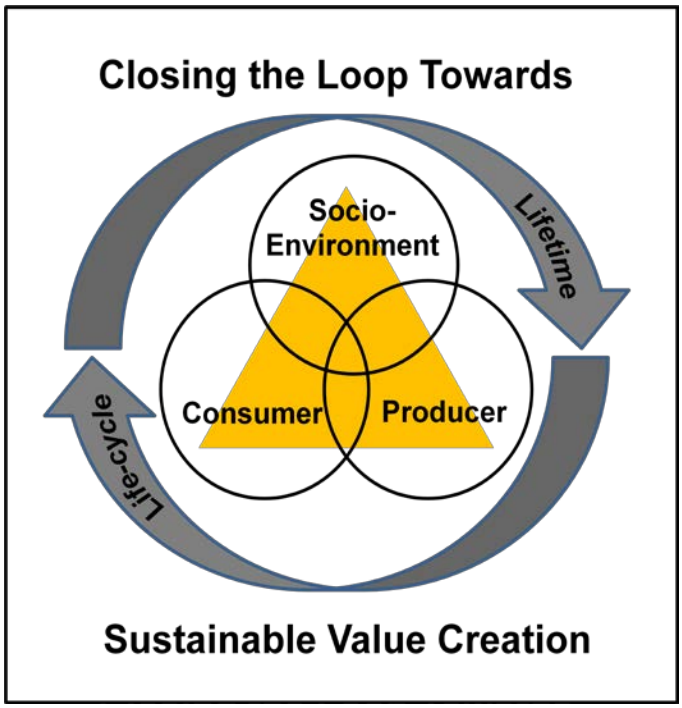


Figure 2.6: Focus on product Life-Cycle and Lifetime in Sustainable Value Creation

2.2.1 Sustainable Value Creation

According to an ASME survey focused on trends related to sustainability in product development, the overriding reason why corporations integrate sustainability factors into their designs is government regulations (Autodesk, 2009; Rosen, 2013). This report surveyed engineers to obtain reasons they would consider sustainability in their product designs. In additon to regulations, rising energy costs and client demand rounded out the top three motivating factors for developing more sustainable products. Only 16 percent of respondents reported the potential for improved return on investment. In a

similar survey conducted by the MIT Sloan Mangement Review and the Boston Consulting Group, which focused on integrating sustainability into the developmnet process, 45% of respondents reported that they expected higher operational cost to take away from profits. Thirty three percent cited that the administrative costs of sustainability programs would create additional losses (Kiron et al., 2013). The survey results show that to keep the attention of the design engineer when developing next generation products or grab the attention of the consumer in the purchase of their next solution, sustainable value must be reviewed from their individual and mutual perspectives.

2.2.2 Green Products and Marketing

The decade of the 1970's is remembered for the effects of energy on economies around the world. As a result, focus on environmental business, management, and marketing began to increase. In the years to follow, it was often considered an added expense to focus on environmental management within a business and the name "The Green Wall" was coined to describe the creation of roadblocks to manage issues successfully (Wolff, 1996). Essentially, the word green was used to describe business activities aimed at financial actions considering the environment in the process. To some consumers, green marketing was an effective tool to create demand for environmentally focused products. Although markets grew, they were still considered a niche and not necessarily mainstream.

Recently, there has been an increase in research centered on sustainable value. Beyond green marketing, by focusing on sustainability, businesses were beginning to realize that there could be a win-win scenario in the development of sustainable practices (Laszlo, 2008).

2.2.3 Sustainable Lifetime Value in Product Design

The triple bottom line (TBL) attempts to bridge the two worlds of sustainable development and business. In the center of this concept is the requirement for reconciliation of environmental, social, and economic demands within the context of development. While the engineering community is familiar with the TBL, many struggle to project the concepts onto their own work. In order to put focus on sustainable value, this research identifies the overlapping benefits between the producer, consumers, and the socio-environment. This additional set of pillars is referred to as the Sustainable Product Value Drivers (Figure 2.7).

In order to bring clarity to the concept of value from a sustainability perspective, the first step is to consider value and respective propositions, from the perspective of the key stakeholders. For example, new industries in **green products and marketing** have been created for consumers who seek out environmentally conscious products. The proposition for consumers who seek these types of product seek "green value".

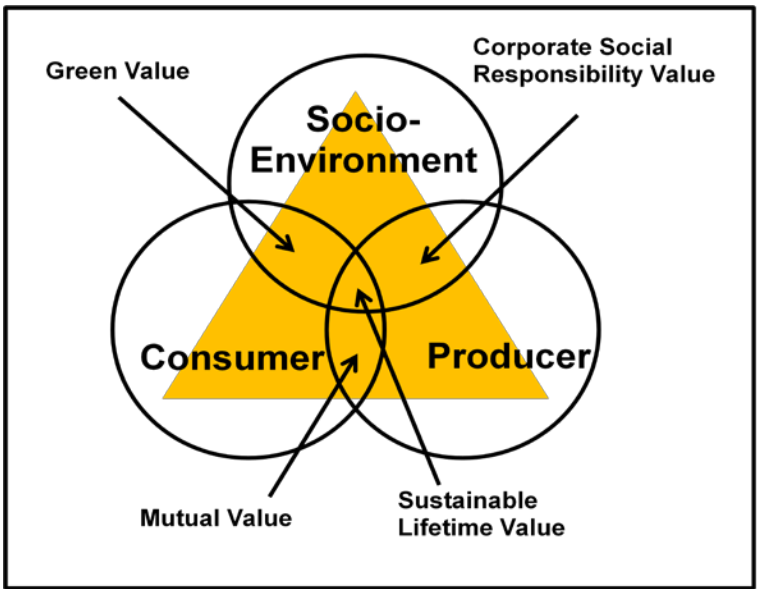


Figure 2.7: Sustainable Product Value Proposition Drivers

Producers are motivated to show their social and environmental value through **corporate social responsibility reporting** (CSR). Consumers and producers often work together to create **mutual value** focused on solutions that reduce workflow and resource consumption. Producers calculate the net profit they obtain from customers over the life of a product and call this the **Customer Lifetime Value (CLV).** Yet, many engineers lack the tools or foresight to break down the new product design process into the driving metrics that would simultaneously seek new value creation for the consumer, producer, and the socio-environment. In order to indentify the driving aspects of Sustainable Value Creation through product design, long-term value must be examined from each perspective. This examination of long-term value will lead to the calculation of lifetime value of a product from a broader perspective. **Sustainable Lifetime Value (SLV)** creation in the design of products can be achieved by integrating all three value drivers into a sustainable value proposition that seeks mutual benefits for producers, consumers, society, and the environment, without taking away from future generations.

Producer Value: In order for producers to be profitable, designers strive to develop products that meet customer needs at acceptable production and delivery cost – thereby creating a mutual value proposition. Product use and life are the key deliverables.

Consumer Value: Potential Customers seek out innovative solutions that meet their needs. In doing so, consumers weigh these potential solutions against the total cost of purchasing and owning the product.

Socio-Environmental Value: From a sustainability perspective, new products or solutions that improve the health and well being of society without affecting the ability of future generations to meet their needs.

These concepts are not difficult when studied on an individual basis, but creating solutions that optimize the three key pillars of the value proposition is difficult. In fact, as the world becomes more competitive, the headwinds that development engineers

face continue to complicate their ability to achieve the desired goal of sustainable development. For example, manufacturing losses, abandoned design platforms, and early product retirement are all examples of waste stream that create losses to producers, consumers, society and the environment. Certainly, research focused on lean manufacturing and green marketing can help improve the bottom line. But, in order to have the greatest impact on the long-term development of products and processes, focus should be on developing a sustainable products value proposition that integrates sustainability innovation elements into the product design value proposition. These elements carry the design concepts beyond the traditional 3R's of reduce, reuse and recycle, to include recovery, redesign, and remanufacture (Jayal et al., 2010). This research focuses on the driving aspects of sustainable product design that affect the value proposition between the consumer, producer, and the socio-environment.

2.3 Sustainable Lifetime Value Creation: Integrated Model

In this section, a framework for achieving the goal of Sustainable Value Creation is presented and three focus areas are identified. The framework is designed to help engineers close the loop towards sustainable product development. First, the study of product half-life relative to development metrics introduces the opportunity for improvement. Next, to bring focus to the problem, the concept of an expanded sustainable value proposition is introduced. The high impact drivers for each pillar of the proposition are identified. In doing so, the design engineer will have a set of metrics that will aid in the optimization of value creation in generation-to-generation product development. Finally, a set of primary drivers in sustainable product development are integrated into a tool set framework.

2.3.1 Sustainable Product Half-Life Return Model

A common paradigm in product development is to focus only on the physical device or base unit the customer obtains in the purchase of a new product. Development engineers can be so focused on the delivery of their particular design or sub system to the manufacturing team that they fail to take a broader view of the product definition. In the same manner, producers often assume the final day of product sales (the day of the last product family purchase) as the end of the product life-cycle. Traditional financial models, such as "The Return Map," highlight the last day of product sales as the end of time in calculating the return on investment. When the concept of the sustainable value drivers is integrated into the equation, it is evident that the life-cycle of a product reaches a broader definition, as well as the product definition itself.

By focusing on the broader value proposition presented to the customer, the definition of the deliverables that are integrated into the product design may also include items such as customer replaceable sub-units, warranty material, operating systems, and solution software. A broader product definition will enable the analysis of the lifetime of a product from a sustainability perspective.

The first model presented is the sustainable product Half-Life Return Model (HLRM). It is intended to shed light on primary metrics that will enable the development of financially successful products with improved sustainability attributes.

The HLRM focuses on variables that affect long term portfolio gains and losses, which is central to Sustainable Value Creation. It integrates two important data streams into one map:

1. The net profit & loss curve for the product family over the life-cycle
2. The total number units sold that are still in use over time. This curve is also used to track product use degradation and the product half-life point.

Profit and Loss Life-Cycle Curve

The first data set tracks the product family net profit-loss (net P&L) curve over time (Figure 2.8). The key points on the life-cycle curve are described below.

Start of Development - In the beginning of the product life-cycle, producers invest in the development of the product and the early expense drives the net P&L negative (i.e. currently the investment is a negative gain). The SOD date marks the first day of development expanse.

Expense Turn Around Point - After the start of production and sales which bring in revenue and presumably net profit per sale, the P&L curve, which had been trending in the negative direction, eventually slows down. At the point where net unit profits are greater than net unit expenses, the expense turn-around point is met.

Break Even Point - The first time total profit is equal to total expenses, the breakeven point is met.

Program Life Profit and Loss - The net profit or loss of the product family is at the end of the total life-cycle. When the total life-cycle of a product family is taken into consideration, there may be expenses a producer incurs that are required to transition a product at end of life. These expenses could include collection, clean-up, or program scrap charges. Because of this, there may be another inflection in the overall P&L curve over time

Beginning with the concept generation phase in the beginning of product development, expenses are incurred in the process of the design and delivery of a product. Figure 2.8 presents the net profit or loss of the product over its life-cycle. After the start of sales, the intake of revenue may begin to offset expenses and a profitable product will eventually pass the break-even point and generate net profits over the product life. The data presented in the figure represents one product life-cycle. The engineering

community takes advantage of a product delivery process in order to improve the P&L life-cycle curve for a given product platform.

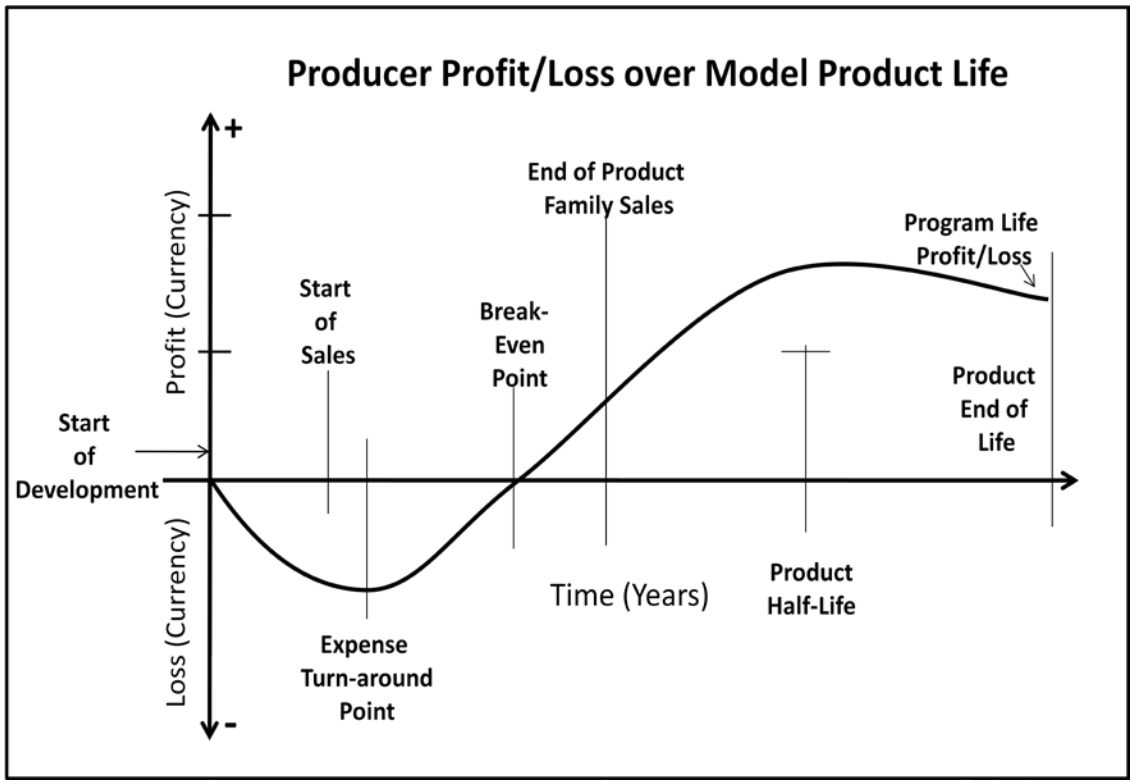


Figure 2.8: Producer profit/loss over model product life chart

Product Half-Life Curve

The second data set presented in Figure 2.9, tracks the total number of product family units that are currently in use after the start of product sales. As a product family churns (Reinartz, 2003) (where some amount of the original systems originally sold are not in use anymore and essentially retired), they are removed from the half-life curve. It is important to note that in some industries it may be possible that some of the initial units sold have been retired before the last unit is sold, even with products that have a very short half-life. The data presented in the figure represents one product life-cycle.

The engineering community seeks to analyze the value proposition in order to improve the product half-life. The key points on the life-cycle curve are described below.

Start of Sales - The point where products are available for purchase by consumers

End of Product Family Sales - The point where the last unit of a product family is sold to consumers

Product Half-Life Point – The point where half of the overall products sold to consumers has been retired and is no longer in use.

Product End of Life– The point where all products have been retired and are longer in use.

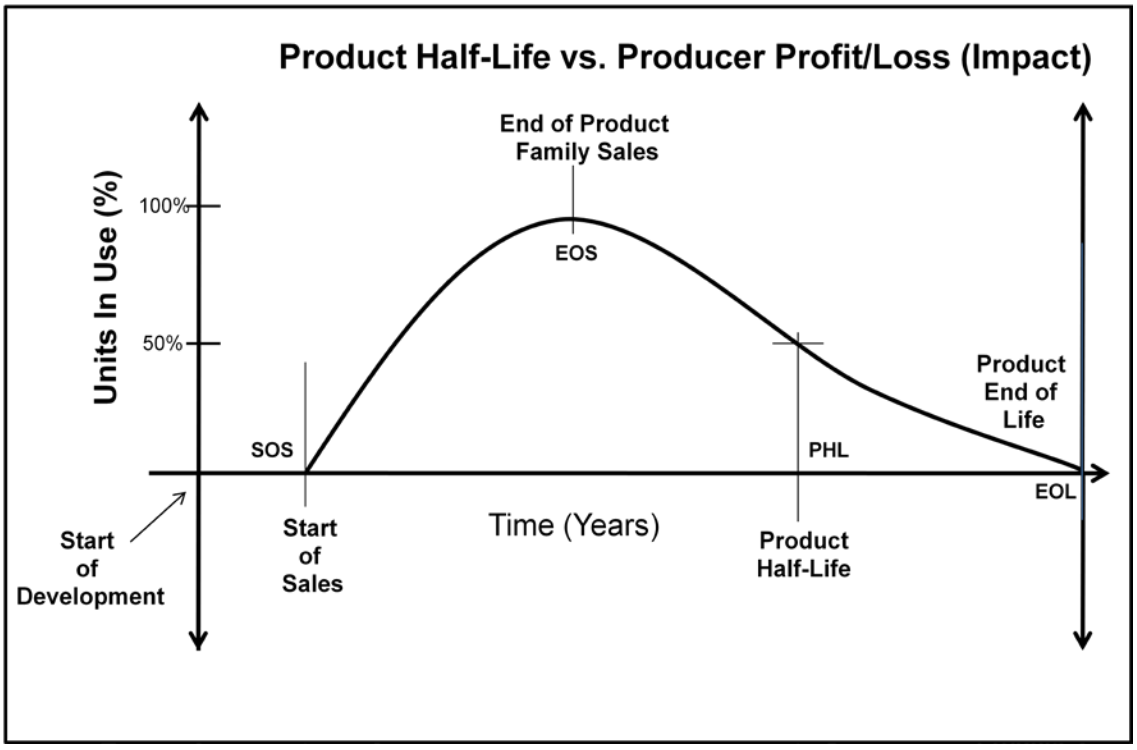


Figure 2.9: Product half-life vs. producer profit/loss chart

Product Half-Life Return Model

Producers can gain insight into the goal of Sustainable Value Creation by integrating data from the product profit and loss curve (Figure 2.8) on top of the data from product half-life tracking (Figure 2.9) as shown in Figure 2.10. Essentially producer profit and loss (especially expenses related to the development phase and the end-of-life phase) is monitored relative to the longevity of the entire product family life-cycle. The goal of Sustainable Value Creation should take into account producer product expense, consumer life-cycle expense, and socio-environmental drivers.

In addition to the key traditional metrics identified in the study of return on investment, an additional set of metrics is identified that expand the view of Sustainable Value Creation. By integrating the product half-life data on top of the producer P&L curve over time, a greater appreciation of the key elements that extend the goals of the consumer, as well as the producer, are highlighted, which, in turn, affects socio-environmental targets. The primary metrics in the Half-Life Return Model are:

- Start of Development (SOD)
- Start of Sales (SOS)
- Expense Turn Around Point (ETP)
- Break Even Point (BEP)
- End of Sales (EOS)
- Product Half-Life Point (PHL)
- End of Product Life (EOL)

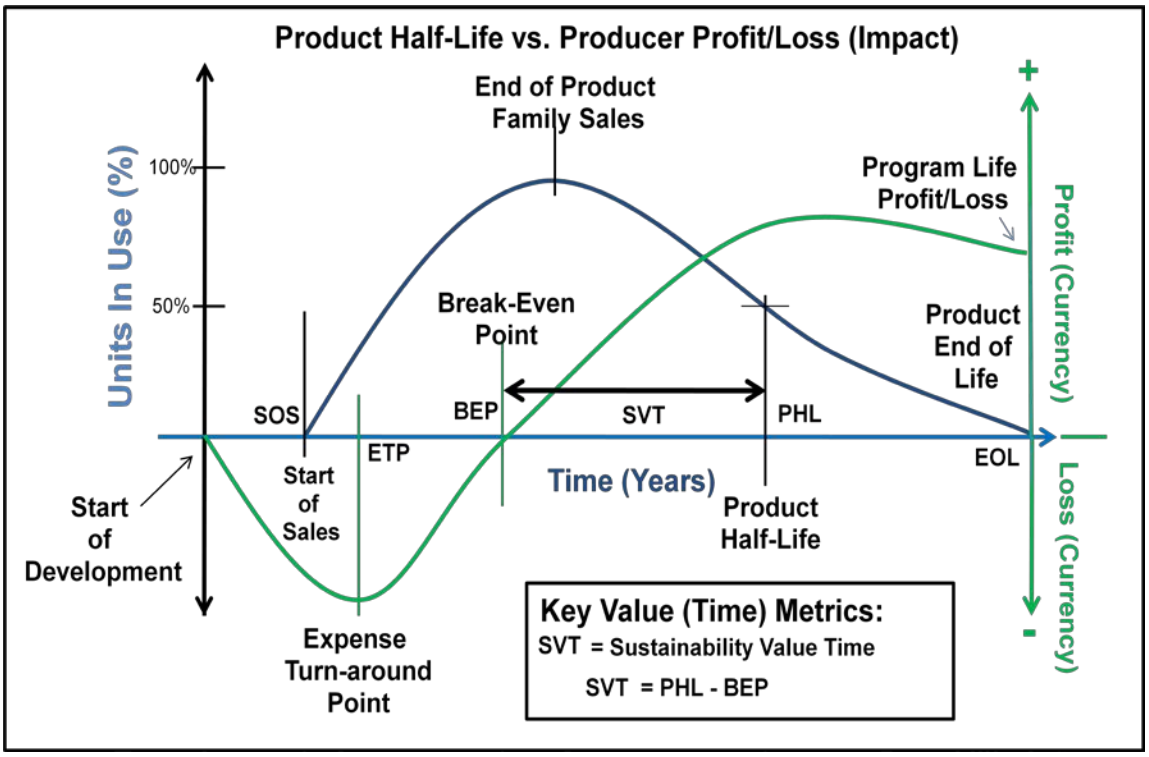


Figure 2.10: Sustainable Lifetime Value Creation Tool #2: Half-Life Return Model

The traditional return factor ratios remain interesting at any given point of the financial curves but, from a sustainability perspective, the product half-life point (PHL) relative to the producer investments draws attention to the return of investment from a total life-cycle perspective.

A potentially sensitive metric is defined as the **Sustainability Value Time (SVT)**.

By integrating the product half-life data (traditionally associated with the product value proposition) within the product profit and loss data (traditionally associated with the product delivery process - PDP), the engineering design team can monitor the potential value creation or loss due the affects of many variables. The SVT is calculated by subtracting the breakeven point (BET) from the product half-life point (PHL). The breakeven point is a relative reflection of the initial return on investment of the development expense of a product. The product half-life point is a relative reflection of the long term viability of a product family. By comparing the Sustainability Value times

(SVT) of relative products to each other, the engineering community will become more in tune to the drivers of Sustainable Value Creation.

Sustainability Value Time

$$SVT\ (time) = PHL\ (time) - BEP\ (time) \qquad (2.1)$$

The model curves, shown in Figure 2.10, represent the ideal state where it is assumed a product family will be successful and drive an overall profit over the life-cycle. In many products, especially in industries with continuous technology advancements, the HLRM curves may look much different. In fact, many high technology products have a negative value for the SVT metric, where the product half-life point is crossed before the producer has passed the break-even point on the P&L. These types of products may be at the highest risk for early abandonment. A negative SVT number not only projects higher risk for the producer but may also indicate the potential for higher risk of loss by the consumer, society, and the environment.

Possibly, the most detrimental scenario from a socio-environmental perspective are new products or family platforms that are deemed a failure in the market, never reach (positive) profit, and leave behind abandoned hardware, test models, and sunk cost investment (losses) by both consumers and producers.

There are additional scenarios that highlight the study of combining financial data with product life data. For example, the process of setting the market price for a product line is a science and a strategy by itself. From a pure economic perspective, supply, demand, and utility play roles in the price that is set by the producer to maximize profits. Simple supply demand curves are essentially setting a price at a point in time. Lower prices may drive higher sales which is one of the variables on the Half-Life Return Model. But, if the long term viability of the platform is in jeopardy (for example a fire sale on a product line that may be discontinued), the half-life of that product may be relatively shorter than a similar product and, therefore, less sustainable.

Half-Life Return Model Goals

From an engineering tool kit perspective, these development process and relative design comparisons should be considered to achieve the following goals:

Maximize

- Sales
- Program P&L
- Product Half-Life (PHL)
- Sustainability Value Time (SVT)

Minimize

- Expense Turn Around Point (ETP)
- Start of Sales (SOS)
- Break Even Point (BEP)

The most significant focus on the metrics that are targeted for minimization should be the development process. Improved collaboration, concurrent engineering, and improved processes are examples of techniques that could aid in minimizing ETP, SOS, and BEP. By focusing on the sustainable value proposition, the development team will be able to improve the goal of maximizing Sales, Program P&L, PHL, and SVT. Similar to traditional return on investment type ratios, additional metrics that add focus on Sustainable Value Creation are outlined below.

There are many factors they may influence the definition of financial success for any given producers. For example, a corporation who seeks a growth strategy for their investors may place priority on revenue growth. The ration of Product Sales ($) divided by the Sustainability Value Time can provide insight into the growth over time for a given product platform (Eq. 2.2).

Value Time can provide insight into the growth over time for a given product platform (Eq. 2.2).

$$\frac{\boldsymbol{Sales}\ (\$)}{\boldsymbol{SVT}(\boldsymbol{time})} \tag{2.2}$$

For a company who in focused on return on investment (potential a value based corporation), the ration of P&L divided by the Sustainability Value Time can provide insight (Eq. 2.3).

$$\frac{\boldsymbol{Profit\ or\ Loss}\ (\$)}{\boldsymbol{SVT}\ (\boldsymbol{time})} \tag{2.3}$$

Simplified versions of the equations can be calculated using the product half-life instead of the product Sustainability Value Time.

Another paradigm of influence for some engineers is the belief that the Bill of Material cost (BOM cost) for a product must increase if the design is modified for sustainability. By focusing on the longer term ratio of the Bill of Material (BOM) cost divided by the product half-life, a new perspective on the value of designing for the full life-cycle can be introduced (Eq. 2.4).

$$\frac{\boldsymbol{Bill\ of Material\ Cost}(\$)}{\boldsymbol{PHL}\ (\boldsymbol{time})} \tag{2.4}$$

The race continues between the e-gain benefits of new technology and the research for new tools that will aid in the long term development of more sustainable products and processes. A central goal of this research is to begin to build a new paradigm for development engineers, a paradigm that sheds light on the realization that product designs can be more sustainable from both a financial and environmental perspective. By focusing on the main drivers of each sustainable value proposition aspect, the development community improves their role in creating truly sustainable value.

The sustainable products value proposition seeks a balanced approach toward the integration of total cost of ownership, social and environmental improvements, and an expanded definition of product life drivers.

2.3.2 Sustainable Product Value Proposition

One difficulty in developing a common set of aspects in the design of sustainable products and processes is the need to integrate a wide array of drivers into one common analytical metric set. In the process of identifying the driving aspects of the sustainable products value proposition, categories that have the highest impact from a value perspective are identified. In this process, value is viewed as the potential for new utility relative to its cost. In order to have the highest impact on the long-term goals of sustainability, generation-to-generation product designs should seek to improve each pillar of the driving aspects at the same time.

A common paradigm of development engineers is the assumption that the bill of materials must increase in order to create solutions that accomplish goals such as extending life, meeting regulations, or lowering the cost for the customer to operate. In order to break down this paradigm, detailed drivers for each aspect are identified to provide a broader perspective to the key stakeholder of the value proposition. The first step of this process is to broaden the definition of costs into a total life perspective. The concept of the total cost of ownership (TCO) has been presented in many forms, including research and tools designed for the IT industry. (Bace and Rozwell, 2006; Ellram, 1993). From a financial perspective, TCO represents the direct and indirect cost to purchase and utilize a product for the consumer. The sustainable products value proposition expands the set of total cost drivers. Therefore the primary aspects that drive producer expenses and potential benefits can be identified as follows:

Producer Impact: Cost of Product Development and Delivery

In general, consider the cost of these metrics to be relative to the specific product design points chosen to meet the expected targets.

1. **Bill of Material Expense** – Typically, the primary focus of the development engineer, from an expense perspective, is the bill of material. The bill of material is the cost to physically manufacture the product.

2. **Relative Design concepts of delivered function, specifications, and solutions** – In an effort to meet customer expected quality levels, features, and functions, the engineering team creates the design specification that describes the expected outcome of the system. Typically, higher tolerances and tighter specifications can cost more to produce, but the customer may be willing to pay for it.

3. **Mean time between failure and Intervention** – The most common measure of system reliability is the **mean time between failures**. The uptime of equipment can affect productivity beyond the individual user if the product is involved with any type of work flow. As system complexity, as well as competition increase, another reliability-based metric has become critical for the development community. **Mean time between interventions** is also a measure of product up time, but it assumes that the system needs attention from the user (and not a warranty call). Examples in this category include the following: clearing systems hangs/jams, changing supplies, or updating the system. Complex solutions in the future will have longer lasting Sub-systems and will have intelligent operating and embedded systems.

4. **Cross Platform Compliance within Product Families** – This category is focused on the typical struggles that producers face in the quest for satisfying individual customer needs vs. the financial benefits of focusing on the convertibility or the commonality of components or Sub-systems between platforms. The ability to

convert products already produced increases the value and flexibility of the supply chain team. Increasing the use or re-use of common Sub-systems reduces the amount of development and verification resources required to design the product. This aspect is not only one of the key drivers that producers can use to reduce the cost of their value proposition, but it also applies directly to the improvement of the product family longevity, a key component of the environmental pillar.

5. **Generation-to-Generation Product Compliance** – The focus of this category is on enabling the producer to use existing infrastructure and intellectual property in the development of the next generation solution. Likewise, it enables the customer to use existing infrastructure and intellectual property in the transition and integration of the next generation system. Extending the platform of a product family through generation-to-generation compliance can have one of the most positive effects on designing sustainable products. This aspect is simple in concept but becomes difficult when integrating challenges from competitive designs and considering the tendency of engineers to invent new systems because they can.

6. **Product Life Extension or Retirement –** Product life extension or retirement can be a cost stream or an opportunity for re-designing or re-manufacturing the product for retirement or extended use. Either way, the development engineer takes end-of-life product aspects into consideration in the overall design. The ultimate expense for a producer can come from a consumer abandoning the use of a product before its useful end-of-life.

Customer Impact: Costs and Benefits to the Customer

Ultimately, in free enterprise markets, the consumer is the focal point of new products and the longevity of competing designs. Customers seek out solutions when they realize

benefits relative to the cost of the product. Therefore the primary cost and benefits to the cusomer can be identified as follows:

1. **Benefit of New Innovation and Solution Improvements –** This metric is counter to the others in that this driver is viewed as the aggregate benefts gained by obtaining the new solution. Benefits of new innovation and solution improvements can be quantified through a variety of sources, such as productivity gains, improved quality or reduction in material consumption.

2. **Cost to Purchase, Install and Prepare for Use –** Beyond the initial box cost, many consumers fail to include the cost to install and create the infrastructure for new products. These costs include the training and learning curve required to fully utilize the new solution. Many products are abandoned early due to a mis-match in customer expectations or skill levels.

3. **Cost of Consumables –** This expense stream covers the material or supplies needed to maintain the utility of the solution. They are typically referred to as customer replaceable units (CRU's).

4. **Cost of Maintenance and Product Intervention –** Consumers expect products to work but understand interventions and maintenance of the system might be required. Yet, there is a cost to perform these activities that include expenses beyond the person performing the activity. Often workflow downstream is affected by the downtime of devices.

5. **Cost of Warranty Repairs -** This cost is a combination of warranty expense for the customer and producer, as well the cost the consumer faces with product down time. In order to protect themselves, many customers purchase extended warranties as a precaution in case of unexpected failures.

6. **Cost of the End of Current Life-Cycle –** Beyond the cost of product dispossal, there are often expenses in the activities that lead to the purchase of new

equipment and the removal and possible accelerated capital expense write-off of previous equipment.

Social and Environmental Impact: Indirect Cost of Product Compliance and Natural Resource Consumption

In the process of developing new products, good stewardship of our natural resources is now recognized as a cost savings opportunity, in addition to what more potential customers are expecting to review in the purchasing cycle. Standard reporting and certification processes are integral to the development model. Therefore the primary cost and benefits that are related to social or environmental aspects can be identified as follows:

1. **Total Energy Consumption to produce and operate –** Tracking the consumption of utilities in the manufacturing process is prudent. Focusing on the effects that energy consumption has on the product design often yields opportunity for increased quality or yield. In addition, consumers now track the energy consumption of products, and it is often a critical specification for customer purchase requirements.

2. **Total water consumption to produce and operate –** Energy consumption has been the central focus for engineers who seek to design for the environment. Now water consumption is also a critical aspect as the world's fresh water supplies become more acute.

3. **Product and Material Safety Compliances –** Most products require safety and material certification and approvals. In addition, depending on the product line, there can be a number of specific certifications required to sell to targeted consumers. These specific certifications could include, energy, electromagnetic compatibility (EMC), acoustic, or other aspects of products that affect society and the environment.

4. **Corporate Social and Environmental Activities and Reporting –** The health and safety of employees and consumers is usually a first priority for producers. In addition, many corporations consider taking a proactive approach to social and environmental issues as a benefit to the overall value proposition. Today, many consumers look to producers to pass along sustainability-based metrics as part of the product delivery process.

5. **Industry specific certifications –** In addition to mainstream certification and regulatory requirements, many industries have specific regulatory requirements that are aimed at the unique social and environmental aspects of their products.

6. **Collection and Product Disposal -** Many new regulations require producers to reclaim or, at least, play a role in the handling of products at the end-of-life.

Relative Value Metrics

When integrated into one set of driving aspects, the engineering team is presented with a visual tool that identifies potentially competing cost drivers (Figure 2.11). In a market with worldwide competition, the value of any one particular sustainable value proposition metric is relative to the competitive offerings and societal impacts. Therefore, these values should be considered as dynamic and focus should be placed on continually monitoring a particular value proposition (in the form of a product offering) relative to the best of breed for each individual metric. For the value proposition comparison tool, a scale of 1 to 10 is used to rate each driving aspect compared to the product in the field that is considered the best of breed for that particular value. In order to promote continuous improvement, the best of breed is given a set value of eight across the board. When considering potential designs for next generation product offerings, surpassing the current best of breed value proposition would be rated a relative score of 9 or 10. This system is designed to rate each driving aspect independently from each other. In other words, the best of breed for each metric could be on several competing products. With that being said, a hypothetical score of a total

best of breed for a product offering would be 144 points (multiply each aspect (18 total) by 8). See case study in Section 6.1 for further details.

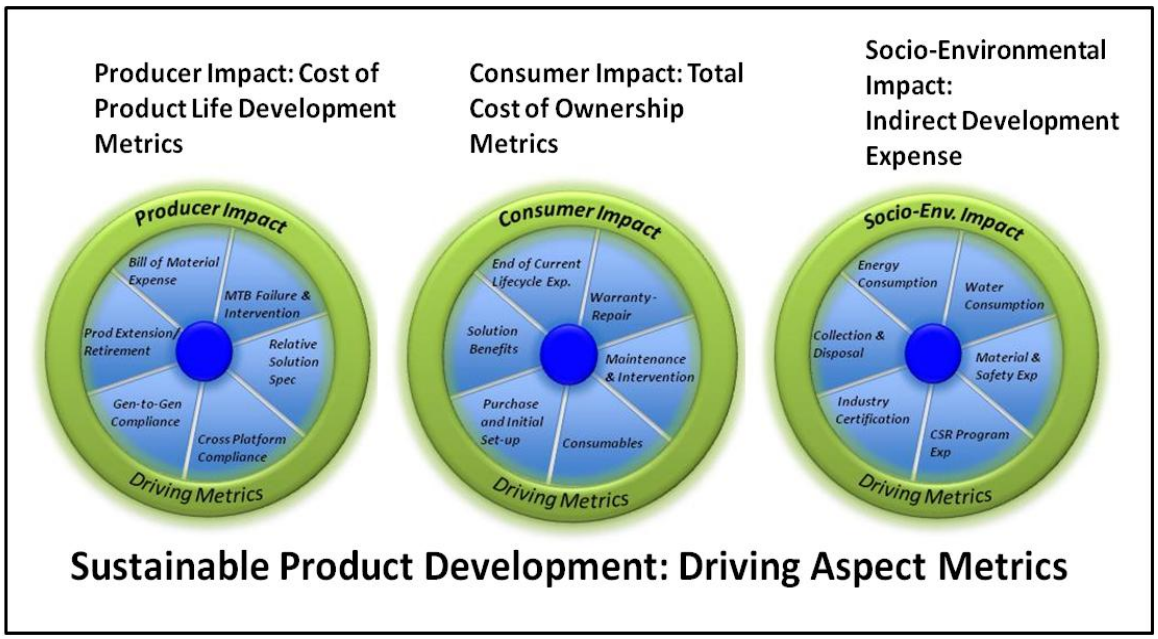


Figure 2.11: Visual tool designed to compare current design to the industry best of breed in each metric

2.3.3 Sustainable Product Development Drivers: Integrated Framework

In order to drive toward the final goal of Sustainable Value Creation through innovative product design, it is important to establish a clear relationship between product development processes and sustainability. There are many forces that a design team must account for in the process of developing specification and ultimately the final design of a complex system. At the same time, the design team must integrate consumer requirements and the effects of competitive offerings. In order to drive a longer term perspective to product development, the concept of the sustainable value proposition was introduced. The sustainable value proposition identifies 18 detailed drivers divided into three primary aspects which are the following: producer value impact, consumer value impact, and socio-environmental impact.

Many development teams may have a goal to increase the sustainability of their product portfolio but struggle connecting the relationship between financial drivers of a potential new design concept vs. the potential improvement from an environmental or societal perspective. The introduction of the product Half-Life Return Model was presented to integrate financial data with the success and longevity of products over an entire life-cycle. There are many reasons why a consumer may abandon the use of a product in the field and move on to a new solution platform. In addition to new technology, product quality, total cost of ownership, and work flow interruption are examples of drivers that accelerate the amount of product churn in the field, which, in turn, drives the product half-life to a shorter value.

Sustainable Product Development Drivers

In this section, six primary drivers are identified that will aid the development team in designing sustainable products and processes. These drivers are presented in an integrated framework designed to place focus on the mutual goal of closing the loop toward Sustainable Value Creation.

The topic of best practices design is broad and there has been significant amounts of research in areas related to processes that help producers improve financial metrics, deliver a design to market quicker, or even integrate quality to the value proposition (Chan and Wu, 2002; Clausing and Clausing, 1988). Although engineers are becoming more familiar with sustainability topics, the need for improved tools that integrate the benefits of sustainability into the product delivery process is important to address. In order to accomplish this, a broader perspective of the value proposition, the effective working environment, and consumer benefits is required. In addition, an integrated framework that accounts for producers, consumer, and socio-environmental needs will serve as the foundation for a greater understanding of the development of sustainable products and processes.

The six drivers in this section are able to stand alone in contributing value relative to development best practices, but when integrated into one conceptual framework, they enhance the ability to drive the engineering team toward long term Sustainable Value Creation.

The primary sustainable product development drivers are:

1. **Value Creation** – This topic has been described in detailed in Section 2.2. By integrating the concepts of sustainability into the producer-consumer value proposition, the mutual satisfaction over the entire product life-cycle and potentially multiple product life-cycles can be improved.

2. **Robust Design** – Typically the cornerstone of research related to improving development return on investment, robust design best practices are just as prevalent with regard to the design of sustainable products.

3. **Verification Feedback** – Whereas robust design is front and center with regard to developing products, the role of verification is ultimately one of the most important aspects of the development process from a Sustainable Value Creation perspective. Just as unchecked consumption is a concern from a socio-environmental perspective, the development of products without a robust feedback system can be just as dangerous.

4. **Risk Management** – Risk management is the first of two primary drivers intended to improve the stewardship of consumer, producer, and environmental resources. Risk management recognizes there may not be one single solution to any problem. By drawing attention to the process of risk management, engineers will increase their ability to produce higher valued added decisions.

5. **Velocity of Workflow** –Velocity of workflow refers to the relative speed (and direction) a development team cycles through their respective workflow. Just as

continuous improvement is expected in product designs, it is also required in the development process itself.

6. **Resource Optimization** – Taken out of context, engineers often assume resource optimization is simply learning how to deliver the new design with less. In fact, when sustainability concepts are introduced, resource optimization takes on a much broader meaning.

When these six drivers are integrated into the same framework, the design engineer is presented with the foundation that will improve the development process, which guides the team towards closing the Sustainable Value Creation loop (Figure 2.12).

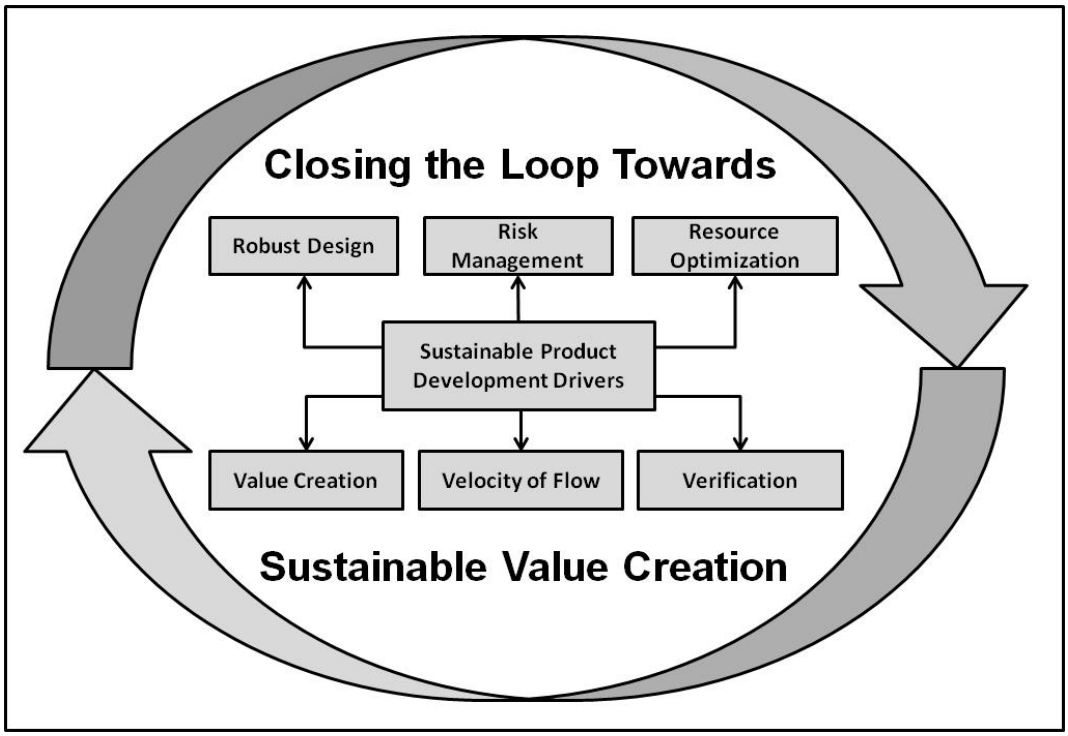


Figure 2.12: Six primary aspects identified that will help drive sustainable product development

Integrated Framework Part 1: Sustainable Value Creation and the Value Proposition

The sustainable products value proposition drivers serve as the capstone for the sustainable product development integrated framework. As described in section 2.2.2, market opportunities in sustainability are present, but they are typically focused on one aspect of the sustainable value triad.

Reflecting on the three driving aspects, generation to generation design concepts can be evaluated relative to the three primary "cost" drivers. The first aspect is the producer impact, which is a view of the cost to develop and deliver the new product design. The second aspect is the customer impact, which is the total cost (including benefit) incurred by the consumer. Finally, the third aspect is the socio-environmental impact, which is the indirect cost of product compliance and natural resource consumption.

Integrated Framework Part 2: Robust Design, Verification and Velocity of Workflow

The second section of the Integrated Framework for sustainable product development incorporates the primary drivers of Robust Design, Verification Feedback, and Velocity of Workflow into the same model. The implication is that these three aspects have the greatest potential for Sustainable Value Creation when they are viewed interactively and within a symbiotic relationship (Figure 2.13).

Just as concurrent engineering encourages earlier involvement of all cross functional team members in the delivery process, verification engineers should be involved with the design and delivery of the product from the earliest stages. There is research that draws attention to the major steps of design and test in the development process (Smith and Reinertsen, 1991). For example, a perspective presented by Tom Abbott describes the "pyramids of product development" where a design process pyramid stands next to (but separate from) a test process pyramid (Abbott, 1988). The design

pyramid is described as a top-down process that starts with designing the system, following with the architecture, high level design, low level design, and, finally, the detailed design. The test pyramid is described as a bottom-up process by, first, starting the verification process with individual components and then working through the Sub-systems up to the final system test. A model that recognizes that development and verification processes are related was developed by the US Department of Transportation and is referred to as the "V" model because the design to verification process follows along in a "V" pattern (Eppinger and Browning, 2012). The development process starts with customer requirements and follows a top-down path from the system down to the component level. Similarly, the verification process starts at the bottom of the V with component qualification and working up through Sub-systems and finally system integration. The model presented in this dissertation adds to the "V" model by focusing on the continuous process of feedback into the design and the integration of risk management and resource optimization into the model.

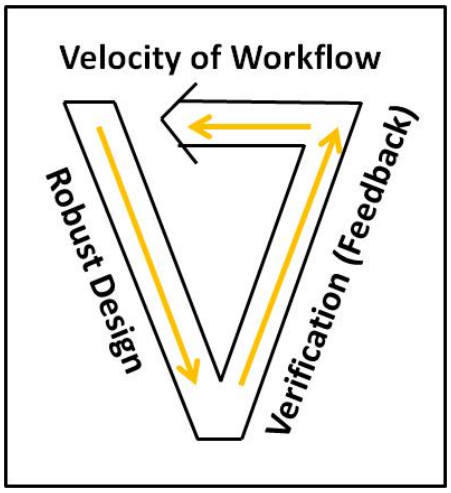


Figure 2.13: Foundation for product development life-cycle

The most important aspect of this model is working through the interfaces on each leg of the V and resisting the urge to jump ahead in the process. (Figure 2.14) For example, an activity that can lead to misleading data and consume verification resources unnecessarily is the desire to take subsystem design modifications and place them into a

system test before conducting the preliminary verification activities within the Sub-systems, models, and components. It is analogous to jumping to the back of a book to see the answer but it will not be in context. A basic example of potential drain on this verification process and velocity is to find an independent fault in the system test. Theoretically, independent faults should be discovered in a sub-system test before the design change is promoted to the final system design status. The mindset of the engineer should be to work their way down the leg of the robust design phases but continue to deal with detected faults and improvements by promoting the improved designs into the higher phase of the verification process. This should only occur after it has passed the interface criteria. For example, the engineer should not promote a design change to a component of a subsystem until it has passed the component verification criteria. The same rules apply to the promotion of Sub-systems up the system integration test phase until after the subsystem has passed not only component verification but also parameter verification. By focusing on the process, verification resources and development time can be preserved.

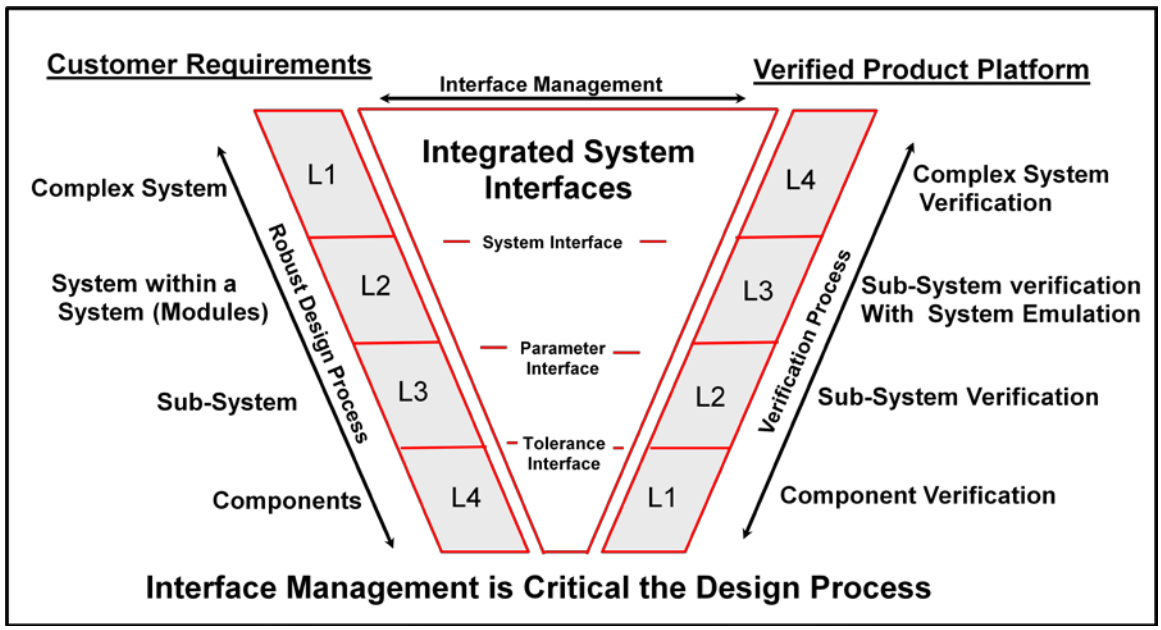


Figure 2.14: The integration of the verification process into the development process is critical to velocity of workflow

In order for this process to be successful, the design and engineering team need to pay close attention to the interfaces between Sub-systems and components. As the goal is to improve the system reliability in the most efficient manner possible, it is essential that faults be detected as close to the original source and as soon as possible as they are injected into the system. Therefore, the velocity of the process to verify designs at the interfaces for potential promotion up into the final system design affects both development resource consumption as well as optimal system reliability growth.

In addition to focus placed on stratified verification, it is important to point out this should be a continuous process. By breaking the verification process into levels, the tendency to wait for all Sub-systems to be promoted to the same system level before beginning the verification process is reduced. Ultimately, in order for the development team to meet the goals of the sustainable value proposition and improve the Half-Life Return Model metrics, they must learn how to embrace feedback and the dynamics of the reliability growth process during the development life-cycle. Simply put, focus should be placed on product delivery and verification processes that allow the engineer to learn how to fail faster. Before an engineer can learn how to fail faster, they must first learn how to fail.

The Integrated Framework stresses the need for integrated feedback throughout the development process, which will be defined as the product delivery workflow. In order to improve on the time and resource consumption during the delivery process, the velocity of the workflow should be studied.

Velocity of Workflow

Velocity of workflow refers to the relative speed (and direction) a development team cycles through their respective delivery process. Just as continuous improvement is expected in product designs; it is also required in the development process itself.

Focus and research on the cadence of the delivery workflow has recently had significant growth in the area of software engineering. Similar to complex hardware products that integrate many related subsystems, enterprise level software development can rely heavily on structured multi-layered programs. Complex software programs typically have thousands of lines of code and, when a sub-section of code is modified, it must be verified before it can be promoted into the current customer level version of code (Cohen, 2010). This verification process is becoming increasingly important as the push for quicker development cycles is amplified. Today, many software development teams have transitioned to working in small "Scrum" teams which is described as an agile development process. The focus of agile/scrum teams is to deliver new functionality in the software through more frequent and smaller iterations.

Creating an environment of quick learning cycles is the primary goal of agile development; creating smaller cross-functionally integrated teams that are focused on the next deliverable and, thereby, creating a development platform where there is less chance for error. In addition, the quality/test engineer is integrated into the agile team and is expected to create the test cases as the code is written (developed). The focus is on controlled changes and value added activities that are only promoted into the customer shippable code after it has been verified. In addition, the team focus is on speed and process efficiency. As a team, the incremental design changes which are to be developed, verified, and integrated into the product are identified. One of the primary benefits of the agile / scrum process is the quick incremental development release cycles called sprints. This type of development process produces steady incremental value added changes to the system.

With short design sprints, the product development team can react to incremental changes in technology and customer demand which builds off the current platform. This drives sustainable value from the producer's perspective.

One of the drawbacks to the agile/scrum process is it may be difficult to inject major system changes or step function additions to the value proposition. In addition, by

integrating the verification engineers directly into the scrum team and focusing only on the next incremental design improvement, there is a chance the team may lose focus on the overall system effectiveness or lose the objectivity of an independent tester in order to keep peace within the team. There is typically a delicate balance between all of the driving factors in the sustainable value proposition that will ultimately affect the key metrics of the Half-Life Return Model. The role of the product assurance engineer is to look at the system from a holistic perspective, including the reliability growth over the development life-cycle. This perspective should not only look for the best solution for the producer but for the consumer and the socio-environment .

Integrated Framework Part 3: Risk Management and Resource Optimization

In many industries, worldwide competition is accelerating as much as technological change. In fact, constant improvements in technology are aiding in the acceleration of worldwide competition and vice versa. In a sense, technology advancements and competitive growth are feeding each other. Because of this phenomenon, the risk a producer takes when investing in developing a new product continues to grow. This is especially true if the development team is using the same process and criteria generation to generation.

Research centered on the design of sustainable products and processes is sometimes met with assertions that the drivers of improvements are not new or unique. For example, it has been stated that the field is simply an extension of lean manufacturing. The issue with this perspective is the lack of integration of the individual concepts into an interactive framework. By drawing attention to the velocity of workflow within the verification and robust design process, the pre-manufacturing aspects of the HLRM are addressed. In order to increase the effects of the product design process on the sustainable value proposition, two additional drivers are integrated into the model

which will attempt to transition the team beyond manufacturing waste and into a view of product performance over potentially multiple life-cycles.

The final sustainable product development drivers are risk management and resource optimization (Figure 2.15 (a)). Development organizations understand the benefits of robust design, verification feedback, risk management, and resource optimization individually, and some measure the linear effects these aspects may have on their delivery process. The development of sustainable products and processes requires new tools that are multi-disciplinary and end to end in perspective. By integrating risk management and resource optimization directly into the feedback process, the development team will have a broader set of tools and information in order to improve the final deliverable (Figures 2.15 (b) and 2.15 (c)).

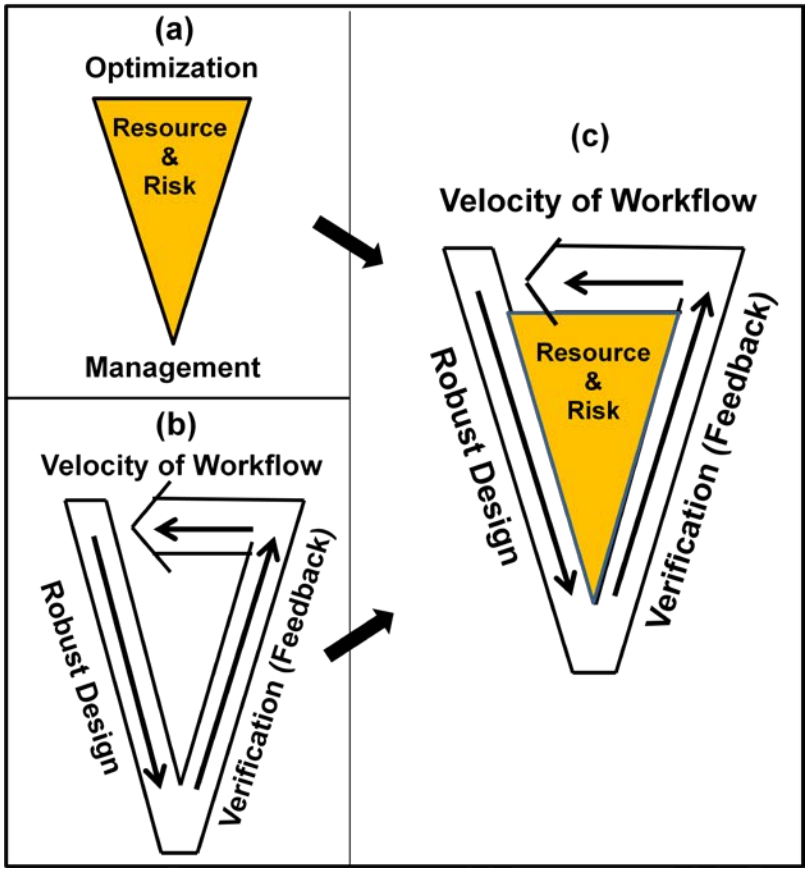


Figure 2.15: The integrated foundation for the sustainable product development tool kit

The integrated framework is presented as the foundation for the primary drivers in the engineering tool kit that will aid in the development of sustainable products. In order to drive toward the final goal of Sustainable Value Creation through innovative product design, it is important to establish a clear relationship between product development processes and sustainability. When you take the individual perspectives of producers, consumers, and stewards of society and the environment, it can seem like a daunting challenge to create a set of metrics and driving aspects that optimize mutual goals. To shed light on the relationship between product life-cycles and development life-cycles, the Half -Life Return Model was introduced. The model introduces key metrics that have the highest sensitivity toward the mutual goals outlined in the sustainable value drivers. From these key metrics, the Sustainable Products Value Proposition was introduced to drive the primary metrics of the HLRM into the cost centers of the producers, consumers, and socio-environment. With these two models, the engineer can make generation to generation design decisions at a more informed level. Finally, six primary drivers of sustainable development are introduced into a model in which the integration of the driving aspects of sustainable product development is presented in such a way that a broader perspective is taken in product design.

2.4 Summary

In order to have an effect on the long term process of product development so that future generations are not faced with the poor decisions of the past, a broader multidisciplinary engineering approach is required. The integrated Product Development Framework, shown in Figure 2.16, provides the development engineer with this broader perspective.

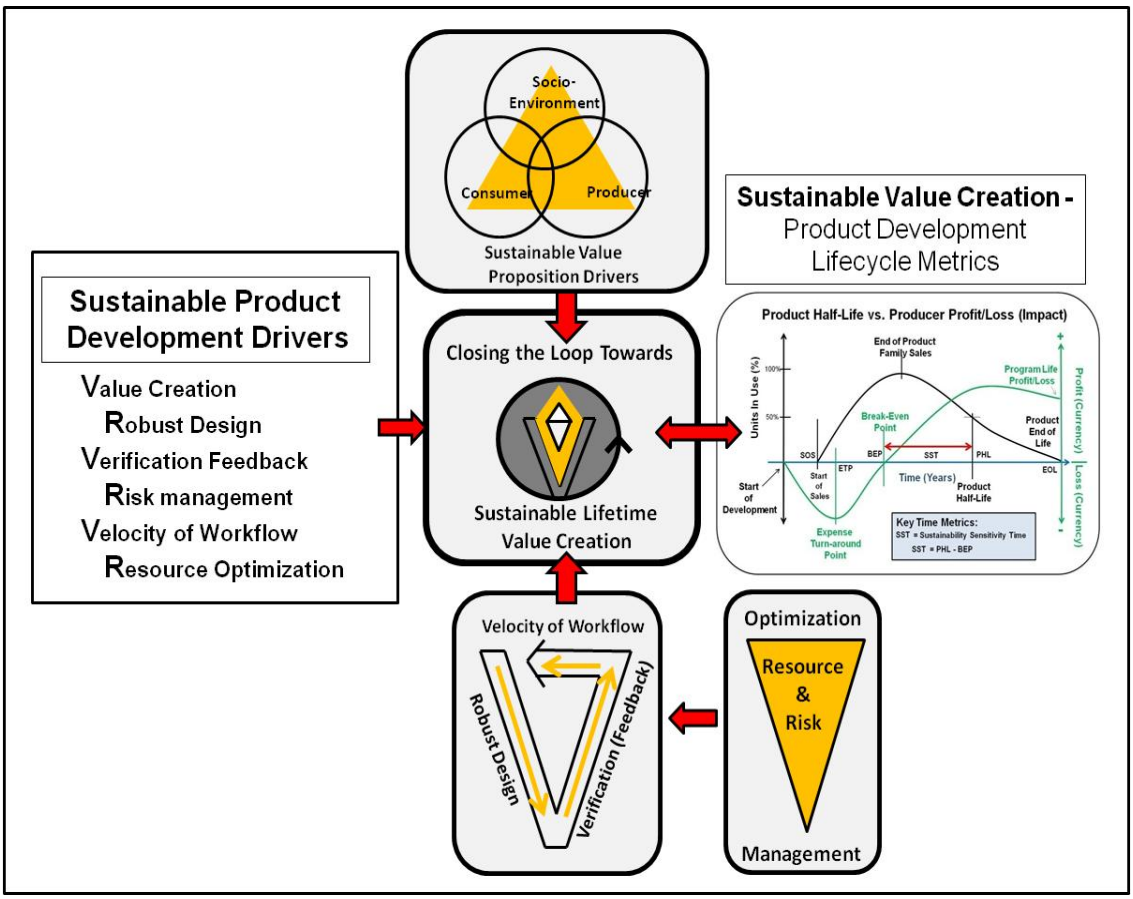


Figure 2.16: The integrated sustainable product development tool kit is designed to maximize the affects of the Half-Life Return Model

The first part of this dissertation was designed to provide the engineering community an integrated framework that bridges that gap between financial success and sustainable product design. The long term goal of this research is to provide the foundation for tools that can be developed to aid in the development of sustainable products and processes. In doing so, one result will be the ability to focus on a more sustainable value proposition between producers and consumers. The driving aspects of this Sustainable Value Proposition were introduced.

By first drawing attention to the value proposition between producers, consumers, society, and the environment, the engineering community has a logical base to build upon in the journey to design sustainable products. The next step identified in this research continues to build the bridge between financial and sustainable product

design. By integrating the product profit and loss data with product utilization field data, a connection is made between financial success and product lifecycle success. The Half-Life Return Model presented in this chapter is designed to provide feedback to producers in the pursuit of improving the return on investment for the expanded set of stakeholders.

Whereas the goal of this research is two-fold, the second part of this dissertation is focused on more in depth application of the key drivers introduced in the integrated framework. In particular, the introduction of higher value feedback during the development life-cycle, in order to increase the lifetime value of the product. Chapter three is dedicated to the problem definition and hypotheses used to research and design a solution that assists the development team in the verification of product designs.

Chapter 3: Problem Definition

3.1 Introduction

In the pursuit of improving the product assurance process in order to develop more sustainable products and processes, a greater understanding is necessary regarding the relative role value between the producers, consumers, society, and the environment. Today, with the rapid acceleration of new technologies, products, including consumer electronics, have become very complex but affordable to societies around the world. With these advancements, two primary issues are developing. First, with rapid growth and turnover of new technology and products, heavy consumption and early abandonment of products has put a strain on society and the environment. From a sustainability perspective, the accelerated growth of higher technology products has generated the following conundrum. Consumer electronics producers are in a cycle that encourages product turnover (new product release) before the current product in use hits the designed end-of-life. Second, in competitive markets, products are sometimes rushed to the retail shelves before the systems are completely verified. This rush also leads to material waste and further reduction in the full utilization of the originally designed product life.

At a time where momentum for sustainable products and processes is building, consumer electronics (a growing market segment) continues to draw attention to the pitfalls of increasing consumption. From a societal growth perspective, the spread of new technology via consumer electronics has been extraordinary (Dupont, 2010). At the same time, electronic waste (e-waste), a byproduct of this growth, continues to grow at an exponential rate and can be a real threat to the environment and society if left un-checked (Chen et al, 2010).

One avenue for increasing understanding and research addressing this problem would be to focus on value from a sustainability perspective. For example, one may debate the potential benefits of a technical product such as a microchip delivers vs. the potential environmental harm that the manufacturing processes may cause. By framing this debate in terms of relative value, new perspectives could be developed to aid in the construction of models and tools for improved products and manufacturing processes. (Williams and Ayres, 2002)

3.2 Product Assurance Adaptive Search Model: Problem Statement

With continued technology and manufacturing process advancements, some products, including consumer electronics, have become so complex that traditional product assurance and reliability engineering processes cannot adequately predict the system reliability or average life of a product. This inadequacy includes the ability to develop test case strategies that are designed to verify the product, with a limited amount of time and resources.

With test case development and data analysis as the primary byproduct of system assurance, a significant amount of cost and resource requirements of product development is in the verification and validation of the design (often referred to as testing and quality assurance) (Godefroid et al., 2005; Albrecht, 1979; NIST, 2002). A critical, yet often overlooked, aspect of product development is testing, verification, and product assurance activities. With the integration of hardware with firmware and software, the number of system combinations requiring traditional product verification testing is not feasible. In essence, if the goal of the reliability engineer is to test every design combination, the problem becomes intractable.

Even with relatively large resource allocations dedicated to this part of the development process, the possibility of escapes can be large. Today, some complex systems are shipped to customers with a projected failure rate at the start of production (Tassey,

2002). The societal costs of these escapes, along with the current expense of design verification, create the need for advancements in process and tool optimization. This need is evident in the increasing number of product recalls and difficult to explain system failures (Valdes-Dapena, 2011; Bunkley, 2011; Maytag, 2010). To complicate the role of development engineers further, in addition to function and reliability, engineers must also integrate a growing number of local and federal requirements into the product. This increases the resource burden on the product assurance teams.

Ultimately, the goal of the product assurance team is to optimize the fault detection and elimination process and minimize system risk to drive maximized customer satisfaction levels. Products that are more reliable and meet customer needs will also improve the sustainability of the product. Maximizing reliability and customer needs can become a difficult job because the assurance engineer is typically constrained on the amount of physical resources and time to reach these goals at an acceptable confidence level.

One of the most difficult aspects of the product assurance engineer's job is the ability to recognize the complexity of factor (variable) interaction within the complex system in test. Most development engineers search for functional errors created within the design (consider these as independent factors) and it is estimated that 70% of system faults fall within this category (Little, 2011). Linear (two factor interactions) account for 25% of systems faults and quadratic or 3 factor faults account for 5% of system faults. Whereas these are average estimates, some complex systems that are highly sensitive to environmental conditions of physical wear can contain a much higher amount of 2 or more factor interactive problems.

In the assurance process for a complex system, the design provided to the test team is already embedded with a large amount of system faults (although more faults can be added during the regression process) (Madachy et al., 2007). These faults can be in the form of defective components and isolated controllable variables, but more often, system defects are comprised of interaction issues between variables.

There is available research in heuristic algorithms and automatic test case generation, but it is mainly focused in the field of software development where Boolean logic reduces the feedback complexity (McMinn, 2004). Because of the large size of most complex software products, current research is focused on predicting software quality through an estimate of the defect potential of the code. Using a variety of testing techniques, the feedback for the assurance team's effectiveness is measured by their defect removal efficiency rate (Jones, 2008; Fenton et al., 2007).

In general, the majority of heuristic search algorithms are focused on finding one optimal point defined by a mathematical objective function. The challenge for this research is to develop an algorithm that allows for multiple target points (and presumably multiple searches occurring concurrently). In addition, most search algorithms are not dealing with a dynamic system, in that a reliability engineer has to be able to deal with how the physical system changes over time. Because of the possibility of latent and interactive defects in hardware systems, as well as the potential for multiple defects related to one sub component in a complex system, subsystems and interactions must be continually monitored in the verification process. To be effective, the algorithm must be scalable. Many search algorithms function with minimal variables. By definition, complex systems problems are intractable and, therefore, the scalability of the model will serve as a primary factor in its value.

During the product development process, engineers use verification feedback throughout the design life-cycle to track the reliability growth of the complex system over time. This verification feedback is typically a measure of system failure rates as measured by time between failures. Reliability growth analysis is an effective metric but may require high cost test methods and may be blind to potential risk elements not known to the design team.

In the fields of reliability engineering and system assurance, the science of test case (fault detection) development with problem resolution management vs. risk analysis and management are often managed independently with separate data and value

streams. This gap prevents the opportunity to focus verification resources on the test combination with the highest potential payback. In addition, time to market and limited testing resources can be a critical factor that affects verification strategies. With limited resources, the ability to modify the testing strategy may be an effective method to improve the fault detection process and system reliability growth. In order to compensate for these constraints, an adaptive search algorithm that feeds the current system metrics back into the test case generation algorithm would be useful. Test case choices can be dependent on many factors, including the level of safety desired, the amount of time and resources available, the complexity of the system, and the ability to describe the system at a module or sub-system level.

3.3 Research Question

With a primary goal to improve the tool set for the engineering community to increase sustainable lifetime value in new product development, the question that is central to this research is the following.

In the process of developing test verification strategies that aid in the design of sustainable products, what are the effects of test case diversity, resource consumption, and risk feedback on the effectiveness of the fault detection and system risk mitigation process?

This fundamental question is used to create the following general hypotheses that are focused on developing the model and potential search algorithm to achieve the desired goals.

- H01: Treating the cost of all potential test case variables as equal will ensures the best chance for the most diversified test case population and optimal fault detection and system reliability growth, given resource constraints.
- H02: After the detection of a system fault, the act of prioritizing the order of fault correction based on risk will improve the efficiency of the fault detection and system reliability growth process.

- H03: Taking advantage of knowledge from previously discovered system faults, by creating child test cases (cut and crossover) from successful parent test cases, will aid in the earlier detection of additional faults when given a fixed amount of time and test resources.
- H04: Testing all independent Sub-system variables before the use of combinatory test case generation, ensures the best chance for optimal fault detection and system reliability growth when given time and resource constraints.
- H05: Taking advantage of knowledge from previously discovered system faults, by modifying the probability of Sub-system variable selection, will aid in the earlier detection of additional faults when given a fixed amount of time and test resources.

3.4 Summary

Chapter three is dedicated to the problem definition and hypotheses used to research and design a solution that assists the development team in the verification of product designs. The first part of this research focused on the concept of Sustainable Lifetime Value Creation, in the pursuit of developing sustainable products. Tools were presented which are designed to aid the development engineer in the design and delivery of products that improve the sustainable value proposition.

The focus of this dissertation now shifts to the application of the concepts described in the integrated framework. The underlying premise of part two of this research is that a richer set of feedback during the development lifecycle will aid in improving the identified metrics in the Half-Life Return Model. A primary source of feedback in the product development process (PDP) is the testing and verification of the product throughout the development lifecycle.

As a result of improving the fault detection and mitigation process during the design lifecycle, the improvement of several key metrics in the Half-Life Return Model are enabled. These include shorter verification cycles and/or the ability to increase the utilization of your test resources. With increased verification throughput, product

quality and customer satisfaction increase. Finally, the net result is an increase in Sustainable Lifetime Value.

An Adaptive Genetic Search Algorithm is presented which is designed to improve fault detection and mitigation. The next two chapters develop the background and foundation for this model.

Part Two: Application of the Integrated Framework: Adaptive Genetic Search Algorithm

Chapter 4: Literature Review

4.1 The Role of Feedback and Verification in the Development Process

Research in the field of control theory and systems engineering is extensive, although the application in sustainability research is still limited. Donald E. Kirk describes optimal control theory as an increasingly important contributor to the design of modern systems. He describes the objectives as maximizing the return from or the minimization of the cost of the operation of physical, social, and economic processes (Kirk, 2012). In related work, authors Terry Bahill and Bruce Gissing describe systems engineering as an interdisciplinary process that ensures the customer's needs are satisfied throughout a system's entire life-cycle (Bahill and Gissing, 1998). These researchers describe the process in seven steps with the assigned acronym SIMILAR. The steps are: State the problem; Investigate the alternatives; Model the system; Integrate; Launch the system; Assess performance; and Re-evaluate.

From a scientific perspective, the fields of system engineering and control theory have the potential to address the need for creating tools and processes that take an interdisciplinary approach to developing solutions that address producer, consumer and socio-environmental needs. As stated in the introduction to the Half-Life Return Model (HLRM) and the Integrated Framework for Sustainable Product Development in Part one of this research, there can be competing goals and objectives due to the need to satisfy all parties involved in the sustainable value proposition. While research intended to model the development of sustainable products and processes could be focused on the optimization of the competing goals of the value proposition, in reality, the development of complex systems to be used complex environments creates too many variables to create a simple verification model. Because of this potentially intractable problem, this research stresses the important role that verification and feedback plays

in the interactive process of product design and delivery. In this dissertation, this process will be referred to as the system product assurance (PA) process.

Research in the field of quality assurance is quite extensive, but the majority of the emphasis is in the manufacturing phase. The assurance of the design phase in the product delivery cycle is sometimes referred to as the design assurance or product assurance process, depending on the organization (Carrubba and Gordon, 1988). Smaller organizations or startup companies may not have independent product assurance teams and typically integrate system verification within the development team. As the complexity of systems grows, along with competition and other pressures that affect the Half-Life return map, research is needed in the field of product assurance to support the development of more sustainable products and processes.

The traditional role of product assurance engineers on the development team is to provide feedback to the design engineers so they can detect systems problems and verify the designs meet specifications and customer expectations. Essentially, the product assurance process is a form of feedback in the overall product development process (Aström and Murray, 2010).

Research in this part of the dissertation is focused on the potential role the product assurance process has in improving the design of sustainable products. Therefore, a traditional systems engineering perspective is relevant. A product assurance model is described that feeds enriched data, as described in the Integrated Framework, back into the development process. Focus is placed on the process and speed of the fault detection and mitigation algorithm along with integrating risk and cost into the process.

4.2 Product Assurance Background

In order to design an effective model and test case development algorithm, a deeper understanding of product assurance processes is necessary. A number of topics will be

introduced in this section with their key aspects reflected in the workflow of the algorithm logic.

4.2.1 Product System and Solution Assurance Definition

Quality and assurance processes have evolved over the years and are typically tailored around specific product needs. The definition of Product and System Assurance varies across industries and engineering disciplines. In order to set a base line for this research, the following definitions are taken into consideration.

Carrubba and Gordon describe Product Assurance as “the integration of design assurance and quality assurance” (Carrubba and Gordon, 1988). Whereas there is certainly a focus on quality, especially from the perspective of meeting customer expectations, product assurance typically integrates the development team’s delivery process into the workflow to assist in the overall product delivery cycle.

To distinguish between the complexities of the assurance disciplines, the IEEE organization provides the following definitions (IEEE, 2002; Kersher, 2003).

> **Quality assurance** is defined as "a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements." Quality assurance (QA) can be broken down into two main areas: **product assurance and process assurance**.
>
> **Product assurance** is traditionally focused on the verification of product specifications. This verification is usually done via thorough testing. Ideally, it also includes verifying that the requirements are correct, the design meets the requirements, and the implementation reflects the design.

Design assurance is a highly specialized, narrowly focused, and strongly disciplined activity which is product focused, product/process engineering design oriented, technical in nature, based on the scientific method, and organized to promote development of high-reliability products and systems.

System assurance involves the application of design assurance principles on a system basis with the objective of delivering high-reliability products and systems into a market not yet oriented toward high reliability. The purpose, objectives, and implementation of design assurance are examined along with staffing.

For this research, Design Assurance and System Assurance are integrated into the focused definition of Product Assurance (PA). In addition, the goal is to broaden the scope of the traditional PA perspective and introduce the term Solution Assurance. One hypothesis is that by broadening the scope of the traditional product definition to include the overall sustainable value proposition and focusing on the broader solution presented to the customer, the assurance team helps deliver increased value to the producer, consumer, society and the environment.

4.2.3 Valuable vs. Value Add

One of the most difficult testing aspects for a quality engineer is having to report back to a development team with a problem that was discovered. The fear of disappointment in the process of discovering a problem must be overcome with the knowledge that the problem was already embedded in the design. The discovery of the fault was necessary in order to meet the long term expectations of the value proposition. Once an engineer overcomes this potential trap, a second trap must also be avoided. Once a problem is discovered, it is only the beginning for adding value in the verification process. In the assurance of the design and delivery of sustainable products, detecting a problem in a test is valuable, but value is not added until the problem and risk have been mitigated.

The product assurance engineer should avoid the assumption that their job is finished once a fault has been detected. Ultimately, the foundational role of the product assurance engineer is to aid in the mitigation of faults in an optimal manner.

A third trap for which a PA engineer should be aware is a phenomenon in product test that is nicknamed “problem discovery bait and switch.” In complex systems, there is the potential for two or more faults to be associated with the same sub-system variable. As a result, a particular test case may discover one particular fault but, during the fault isolation process, identify a different fault. A product assurance engineer should be careful not to assume that once a particular fault is corrected that there are no other potential faults with that variable.

4.2.4 The Cost of Poor Product Assurance

Although it may be easier to measure the cost of poor product assurance vs. good, the choice of proper metrics to define poor PA is difficult. A question one might ask is: “What should the report card of the product assurance team look like?” Product design faults that escaped the product assurance process (and reach the customer) are typically divided into three categories. These categories include the following: faults detected during the PA process but deemed (correctly or incorrectly) as acceptable risk; faults that escaped the development verification process but should have been detected; and faults that escaped which were caused by manufacturing variation or process defect (Figure 4.1). From a continuous improvement perspective, producers should track field escapes in all three categories. Of course, the further upstream a problem is detected and solved, the more valuable the activity is to the producer, consumers, and the socio-environment. In addition, from a sustainability perspective, another high level report card metric could be the product’s actual half-life in the market vs. designed expectations.

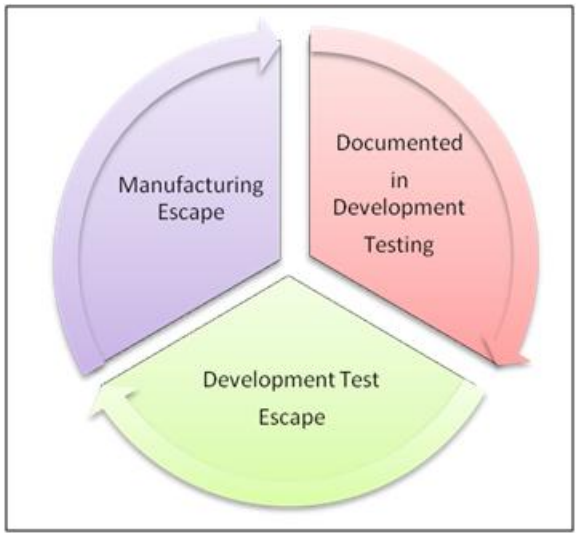


Figure 4.1: Breakout of customer level fault escape categories

Whereas the metrics described in Figure 4.1 can serve as an internal measuring stick for producers, perhaps the most damaging type of escape is when a field escape is so costly that it reduces the value of the producers brand equity. As was described in the introduction, the advancement of new technology drives a steady stream of new product introduction and marketing campaigns. Likely not as well known, there are a number of product recalls that are published daily, warning customers of particular product defects. For example, to provide better service in alerting the American people about unsafe, hazardous, or defective products, six federal agencies with vastly different jurisdictions have joined together to create a centralized website. This website is a "one stop shop" for U.S. Government recalls. These recalls include consumer products, motor vehicles, boats, food, medicine, and cosmetics.

Perhaps, the industry with the highest profile in product recalls is automobiles. Virtually all automakers (including exotic brands like Rolls-Royce, Lotus and Lamborghini) had at least one recall issued during 2012 (Gorzelany, 2012).

According to Gorzelany of FORBES, the auto industry sold around 14.5 million units in 2012. Meanwhile, according to his research, automakers recalled over 14.3 million

current and past models during the same time period. Toyota and Honda combined for more than half of all recalls issued during 2012.

One of the major reasons this trend seems to be growing with defects escapes to the field is because the complexity of system designs continues to grow. The traditional role of the product assurance engineer was to test the reliability of the systems, especially at end of life. Now, with the introduction of software and firmware, the potential number and type of system defects grows with the complexity of the system.

4.3 Product Assurance of Complex Systems

4.3.1 Embedded Defects

Defects on a smaller scale but just as important to individual consumers include those that are specific to the consumer's environment. The interaction of the software with hardware can create field defects quite often in complex systems.

Although major product recalls draw attention to the costs that are absorbed by society, producers and design teams still face the internal cost of verification and the need to manage the assurance process cost relative to projected risk. Depending on the type of product and the confidence requirements, the cost of product assurance verification can be up to 40% of the overall budget (Tassey, 2002). When fail safe systems are required, designers rely on redundant system to establish the factor of safety. When developing complex systems, especially integrating hardware with software, it is more common to discuss the projected defect rate of the product in the field vs. fail safe systems. Research conducted by SPR (Software Productivity Research) compiled data from studies of 600 companies and 13,000 projects, including IBM and ITT, and identified the following averages (Jones, 2008).

- The US average for software defect potential is about 5 defects per function point.

- The US average for defect removal efficiency is only about 85%.
- Therefore, the US average for delivered defects is about 0.75 defects per function point.

4.3.2 Reliability Growth Analysis

There is also research that draws attention to the growth of system reliability over the development cycle, which is referred to as reliability growth analysis RGA (Crow, 1977; Crow, 1982; Hall, 2008). Often new research in the field of system reliability analysis comes from military projects because of the need for high accuracy, dependability, and safety. Larry H. Crowe published some of the original work regarding the analysis of system reliability growth as tracked and measured during the development process. Crowe points out that during the early stages of the development of complex system, prototype models typically contain design and engineering deficiencies. During the product development process, engineering teams progress through phases of design, build, and testing of their respective concepts. As the system design faults are detected and mitigated, the overall system reliability in test grows until it is presumed to hit the intended targets before the start of production. The fundamental premise of this analysis (also referred to as a Reliability Growth Curve RGC) is the instantaneous system mean time between failures (MTBF or sometime mean time to fault MTTF) at that respective cumulative test time (Duane, 1964). Crowe's research noted that Duane's postulate could be stochastically represented as a Weibull process to allow a statistically based process to be injected into the reliability growth model.

During the development process, the act of discovering and correcting faults to drive toward verifying design specifications was traditionally viewed as increasing the reliability of the system. Early research in the field of modeling product assurance reliability growth was conducted by Dana Crowe and Alec Feinberg (Crowe and Feinberg, 1998).

As noted in product assurance definitions, the goal of the PA team is not only to detect and eliminate faults in the design but to do so in congruence with the team's process workflow. Crowe and Feinberg (1998) combine the two factors into the basic model as seen in Figure 4.2.

Crowe and Feinberg (1998) conducted related research in work centered on their stage gate reliability growth model. In this product assurance testing model, focus is placed on accelerating the discovery of embedded product problems through a variety of activities in each defined stage. As seen in Figure 4.2, the first stage calls for the development team to conduct FMEA studies. The next two stages call for aggressive problem discovery through highly accelerated life and stress testing. This model is excellent in detecting and eliminating system problems, but it is hardware oriented and focused on latent defect detection, not necessarily issues such as design for manufacturing, usability, and software issues.

4.3.3 Problem Discovery and Mitigation

During the stages of the development life-cycle, system testing and assurance is used to first detect faults, then analyze, mitigate, and conduct regression testing on the system to insure the effectiveness of the design correction. As the discovered faults are corrected and mitigated during the development process, the system reliability growth increases. The goal of the development team is to establish and execute with speed a product delivery process that includes a product assurance process to optimize reliability growth of the product over the development life-cycle. This same goal is amplified when placed in context of the main drivers of the sustainable Half-Life Return Model. In order to maximize the potential for the design of sustainable products, the goal of the development team is to deliver products that meet the sustainable value proposition and improve the HLRM metrics. Products that meet design specification, quality, and reliability targets are naturally going to be accepted and used longer in the market relative to products of poor quality and reliability. Likewise, development teams

that minimize the time and expense to deliver products that also meet the cost expectations of the consumer also improve the performance of the HLRM metrics.

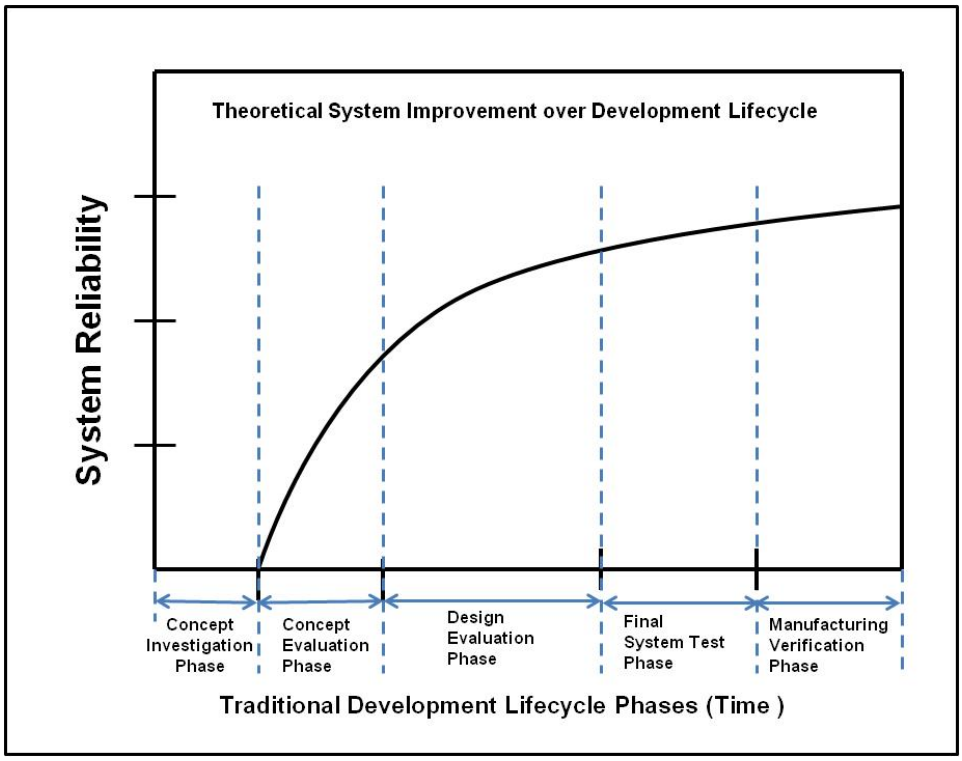


Figure 4.2: Stage gate reliability growth model (Crowe, 1998)

For complex systems, the goal of the product assurance process can be a difficult task due to potential number of subsystem interaction and latent defects that develop over the product life-cycle. Testing and reliability growth strategies are dependent on the number of system interactions, resource budgets, time, and risk management.

4.3.4 Complex System Definition

Today, the first four words of the original phrase by Alexander Pope, “to err is human; to forgive, divine,” are often used to signal an attempt to ask for forgiveness when a mistake occurs. In the past, the instinct was to look for the individual that caused the human error when an accident occurred. Now, as systems become more complex, the

appropriate question is not, “Who caused the failure?” but, “Why and how did the failure occur?” (Strauch, 2002). Because of the potential effects of a catastrophic failure, there is a wide variety of research in the field of complex system failure (Perrow, 1999; Amaral and Uzzi, 2007). The modeling of complex systems can take many forms depending on the desired utilization of the data.

Because the study of sustainable product development involves complex models, there is emerging research in the field of complex systems in sustainability (Fiksel, 2006). J. Fiksel points out there is an urgent need for a better understanding of the dynamic, adaptive behavior of complex systems and their resilience in the face of disruptions, recognizing that steady-state sustainability models are simplistic.

Research focused on complex systems and the effects on Sustainable Value Creation include works by Ueda et.al with the focus on value creation in a decision making society. (Ueda et al., 2009) By definition, modeling the effects of a stimulus on a complex system can be difficult, but often it is a necessity after a particular major failure. For example, as the supply chain becomes more complex in this global economy, a regional catastrophe such as an earthquake in Japan or a flood in Thailand can shut down production facilities around the world. (ElMaraghy et al., 2012) Because of these types of events, many businesses develop disaster recovery plans and use risk modeling to develop action plans deemed appropriate to the potential risks identified. To recognize the reality of product delivery processes within the business world, risk modeling of complex systems is essential to the development of sustainable products and processes.

In reality, complex systems are the aggregation of many Sub-systems. From an engineering perspective, the subsystems themselves are actually a form of smaller complex systems that must also be verified before being integrated into the major system. For the purposes of this research, a complex system (within product development) is one that integrates hardware, firmware, and software designs into one system. In an effort to model a complex design, including one used in the case study, a

complex model consisting of eight (8) Sub-systems with each sub-system containing ten (10) sub-system variables is introduced (Figure 4.3). During the product assurance process, it is possible, and sometimes common, to find independent sub-system faults, but the focus of system verification is to seek and understand faults (defective designs) created by sub-system interactions.

4.4 Risk Mitigation

To illustrate the essential goal of the product assurance team, the concept of risk mitigation is illustrated in Figure 4.4. Recall from the problem definition section, the ultimate goal of the product assurance team is to optimize the fault detection and elimination process and minimize system risk to drive maximal customer satisfaction levels. The constraints on these goals are typically limited time and material resources. Therefore, with a given set of resources, the PA team should create a plan that detects embedded faults in the design in the most efficient manner and, simultaneously, assure the detected faults are mitigated to drive program risk to acceptable levels.

Figure 4.4 (a) presents the ideal state of the product assurance process, focused on product test in order to mitigate system risk during the design lifecycle. In Figure 4.4 (a) and (b), the red lines represent the remaining problems in the form of risk that is still embedded in the final solution. Figure 4.4 (b) presents a risk mitigation curve that represents a more typical development process. The black curve represents the summation of problems discovered by the testing team minus the problems that have been properly resolved. Whereas industry specific producers and consumers establish acceptable product risks levels, the goal of the team is to drive the net risk to a level established by the value proposition.

There is a direct correlation between the reduction of product risk and the growth of the reliability curve. This illustration is simple from a theoretical perspective, but the team must overcome several challenges to accomplish the stated goal.

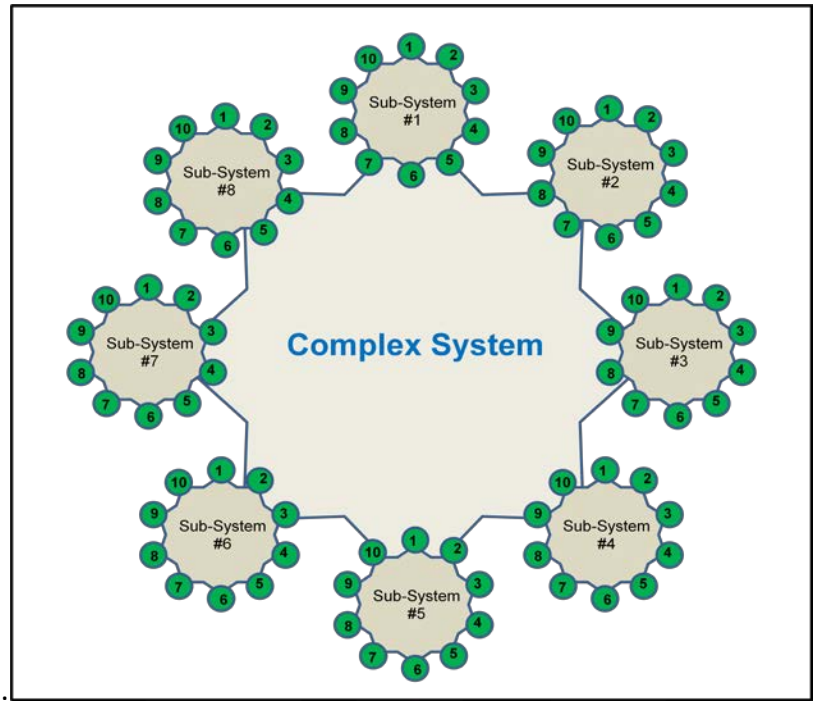


Figure 4.3: Graphical representation of the complex system used in the case study

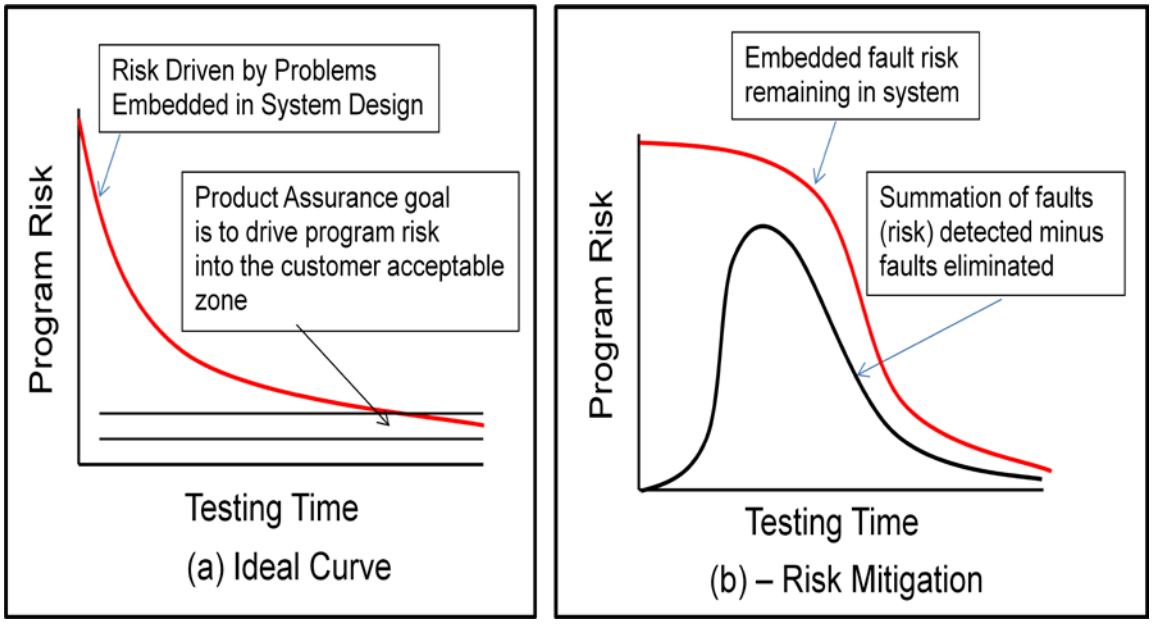


Figure 4.4: The goal of the product development team is to optimize the fault detection and elimination process in order to drive the program risk to customer acceptable levels

4.4.1 Reliability Growth and Fault Detection Problem Statements

In the verification of complex systems, especially with limited testing time and resources, choices have to be made with regard to the goals of the product delivery team. Finding problems in the product verification process is valuable, regardless of the product development phase. Unfortunately, poor behavior is often the result of an assurance process that focuses on the timing of the production start over independent data. The reward for early problem detection includes extra time for problem correction, but a problem detected late in the development cycle can still drive value into the final product.

4.4.2 Reliability Growth Analysis Model Weaknesses

When emphasis is placed on the integration of the reliability growth curve in combination with the product delivery design phases, the benefits of problem discovery earlier in the design process becomes visually evident. Some supporting tools, such as FMEA and accelerated life testing, have been identified to achieve the goal of accelerated problem detection, but problems still exist with the current model. The following statements summarize the drawbacks:

1. Traditional reliability growth analysis is focused on the reliability of complex hardware systems with the failure mode typically detected on a reparable latent failure such as fatigue.
2. In calculating the MTBF or MTTF, all detected faults are treated with the same risk (risk prioritization number -RPN) value. In reality the risk and severity of all faults are not relatively equal.
3. The modeling of the reliability growth is typically represented by a continuous function, but, in reality, many different types of faults are embedded in the design that are discovered at different rates. For example, latent defects and

multivariable faults are usually detected later in the testing process. The time lag between problem discovery and design correction is typically missed in test case development strategies.

4. Because of the time lag between problem detection and problem mitigation, poor decisions can be made in the assumption of the system reliability. Due to deadlines and limited verification resources, human error can be made with assumptions based on a partial set of data. A problem that is discovered at any phase of the development cycle is valuable. A verification strategy that is too greedy may miss important faults in complex systems.
5. Current reliability growth models do not integrate resource consumption or sub-system risk (in the form of feedback) back into the model.

Perhaps one of the most important aspects of the system reliability growth analysis process is highlighted by the saying "you don't know what you don't know." Reliability growth analysis is based on data captured in the past but may not reflect a pocket of embedded faults in a system design that simply has not been detected yet.

4.4.3 Verification Process Weaknesses

The following list summarizes potential problems that product assurance engineers face under the expanded definition of Product Assurance roles.

1. When comparing products relative to previous generation product designs or the competition, scripted or pre-determined tests plans are typically followed. These scripted plans can leave many untested variable combinations on the table for complex systems.
2. Because complex system testing can be an intractable problem, the majority of testing combinations are conducted at ambient (nominal) conditions with standard inputs.

3. Typically, the role of rating the severity of a detected fault and the overall management of product risks is conducted independently of the testing process and is not integrated into the assurance testing feedback loop.
4. In only focusing on conducting scripted tests, the Product Assurance Engineer can become solely focused on test execution and not necessarily focused on the goal of driving risk out of the program and, ultimately, delivering an optimized solution with a finite set of resources. Discovering and driving out system faults toward the highest levels of confidence is the goal of the product assurance engineer, but it comes at a price. Product verification budgets have limits on time and materials. The optimal use of these resources is the primary focus of this research.
5. In addition to a finite amount of resources, another is the problem discovery process. Slow problem resolution and risk management increases the potential for product development delays. By tracking sub-system performance and design delivery, an adaptive test algorithm could possibly increase the risk mitigation of the system.
6. Hardware faults are typically quite different than software faults; therefore, detection testing is often conducted by separate organizations. Faults can be functional, interactive, and latent, including end of life reliability.
7. Complex problems can be masked or hidden from the tester's search capability. This dependent multi-variable problem is undetectable until an overriding independent problem has been detected and corrected. A phenomenon nicknamed "Bait and Switch" can occur when the initial test case finds a problem but, in the isolation and regression process, a different problem is eliminated. An engineer should not assume the possibility of further defects when a particular variable does not exist. (Isolation Testing Returns Alternate Fault)
8. Prototype variation can add to the complexity of system verification. While it is good to represent the range of possible dimensions with tolerance, some

aspects of the parameter designs are not defined early in the verification process. Undefined dimensions and tolerance ranges can confound test results.

9. In the fields of reliability engineering and system assurance, the science of test case (fault detection) development with problem resolution management vs. risk analysis and management are typically managed independently with separate data and value streams. This gap prevents the opportunity to focus verification resources on the test combination with the highest potential payback.

4.4 Test Case Combinations

In the business and technical world, the phrase “analysis paralysis” is often used to describe situations where more time is spent thinking about a problem (and therefore money spent) than actually solving the problem. This may happen when people are simply avoiding the problem but, quite often, it occurs because the decisions makers are overcome by the sheer quantity of information and choices (Schwartz, 2009). It can also be used as a term to vent frustration over traditional product assurance testing methods requiring an amount of testing resources that could cost more than the product’s projected profit. With this in mind, new testing strategies and product development theories are desired by businesses seeking to improve their path to market and quality of product. (i.e., one of the primary goals of the Half-Life Return Model).

Recently the Agile Software Development Methodology has become very popular. It focuses on quick learning cycle sprints and incremental field improvements vs. long development life-cycles (Martin, 2003). In some regards, this methodology is a more natural process for software development over hardware development because it is much easier to send software bug fixes directly to customers as opposed to fixing hardware devices in the field. In fact, it has become common to get software updates on a constant basis and, many times, without the customer even knowing about it. The underlying problem with complex systems is that it is impossible to test every

combination of subsystem variables and, as a result, there is a need for research in test case development strategies.

There are a number of research papers indicating that developing test cases that cover all variable combinations is an intractable problem (Kuhn et al, 2004; Cohen et al., 2003). For example, in the case study designed for this disssertation, the compex system is defined as containing 8 Sub-systems (x = 8) , each with 10 sub-system variables (y =10). The number of potential test cases that covers every sub-system variable combination would be the following:

$$y^x = 10^8 \text{ (Test Case Combinations)} \quad (4.1)$$

Obviously, this is not a practical solution; therefore, the next step is to use combinatory testing and designed experiments that take advantage of multiple pairwise combinations in full system test cases (Taguchi, 1987). The majority of this type of testing, including orthogonal array testing (OATS), “Robust Testing” and covering array testing, has been developed for the software industry (Brownlie et al., 1992; Krishnan et al., 2007; Cohen et al., 2003). In the software industry, Kuhn and Wallace point out that studies show that the majority of design faults were either single variable independent or two variable dependent faults (Kuhn and Wallace, 2004). In addition, the use of historical knowledge could be used to identify sub-sets of the code that have been more prone to failure. The researchers propose a technique that does not test every combination, but, with the use of inteligience in their test case choices, can be considered equivelent to exhaustive or “pseudoexhaustive” They use a formula to create the smallest amount of test cases to cover all pairwise combination and they prioritize the test cases using an assigned value to modify the algorithm (called failure-triggering fault interaction – FTFI). Their research is proactive in seeking more efficient test case generation strategies but uses historical data to alter the search focus. The focus of this dissertation is to use real time feedback to alter the test case generation process, referred to as an adaptive search model.

To simplify the mathmetics, the combinations of test cases are described by a *n-tuples,* which is simply an ordered set of *n* elements (this can be interpreted as a vector) (Weisstein, 2014). From combinatory theory, the system test case size and desired test variable combinations within each system test case is used to calculate the number of test case runs required to hit every combination at least once (covering arrays).

4.4.1 Product Assurance Testing Strategies

There are a variety of strategies and tools to achieve the ultimate goal of fault detection, elimination, and final system risk assessment. As products become more complex, verification costs rise and assurance confidence levels diminish. Because it is impossible to test every combination in a complex system design, many techniques to aid product assurance engineers have evolved over the years.

In order to draw attention to the need for improved test development strategies, the two extremes of traditional methods are described.

The most logical method to test any system is based on a predetermined test plan that is established that covers (a.k.a. covering array strategy) the historical usage and environmental conditions (Krishnan et al., 2007). It can also be referred to as balanced or grid testing because a predetermined test plan is in place, regardless of the quality or maturity of the product design. This type of testing also covers comparative methods that are used to establish the metrics of the products relative to previous products or the competition. Whereas the results of pre-established testing are useful, in complex systems where only a percentage of system combination can be evaluated, test gaps are a reality and it is possible that faults can go undetected.

On the other extreme of testing methods (from 100% pre-determined) is a method that is based on 100% reactive testing (Figure 4.5). Knowing that all combinations of a complex system cannot be tested with limited amount of time and resources, product assurance engineers often react to a particular problem discovered in the test. Another

description of this action is “smell the blood testing.” When a fault is detected, the engineer will zero in on the system problem to try to flush out related problems. The issue with this type of testing is that, without any logical tools to guide the test engineer, test case selection tends to become highly reactive and can lead to overcompensation of searches in local areas. This overcompensation leads to a larger percentage of the complex system not exposed to testing combinations and results in potential fault escapes.

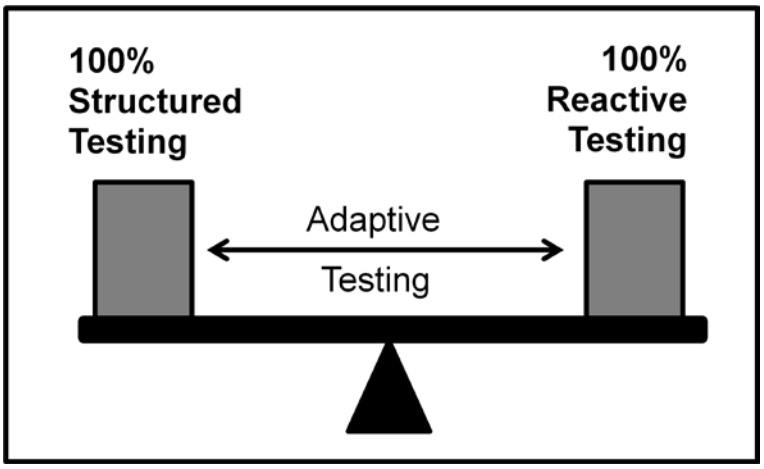


Figure 4.5: Test case generation strategies vary from 100% reactive to 100% predetermined

A third and emerging method is referred to as discovery testing. In some fields, test engineers have abandoned traditional specification testing practices and a new field of verification methods are being developed under the umbrella of “Discovery or Persona” testing (Kaner, 2008). With complex systems, there can be hundreds of primary design variables, and it is physically impossible to test every combination. Therefore, the strategy is to focus on the primary “real world” scenarios within which the product will be used. Test engineers are encouraged to take on the persona of the target customer and use the product in the target environment. The theory is that the focus is placed on discovering the most mainstream and relatively important problems in the most efficient manner.

With the third category described above, the majority of new testing methods, including research in applying techniques such as intelligent algorithms, has been in the field of software development (Pauik et al., 1991; Sharma et al., 2010; Bach, 2003; Blanco et al., 2009). Because software and some aspects of firmware are digital in nature, heuristic test algorithm can be developed that take advantage of high speed computing power. The use of these types of intelligent test system strategies has been less prevalent in hardware reliability engineering research.

Hardware engineers often focus on factors of safety for no-fault systems with redundancy or predicting reliability rates for repairable systems. Tests results are used to statistically predict component and system reliability. Component variation and manufacturing tolerances play a primary role in the documentation of the design. On the other hand, software engineers work in a digital environment and, therefore, Boolean logic drives the majority of verification processes. Software verification methods typically focus on the use of historic models and relative problem burn down rates to predict the current level of code quality. These results could then be used to estimate the relative risk remaining in the system (Jones, 2008).

Because hardware and software engineers traditionally treat these processes differently, the availability of standardized tools across disciplines is diluted. Since complex systems combine various analytical disciplines and metrics, cross functional tools for product verification and reliability assurance must be developed. New testing methods and algorithms are needed to provide the assurance and reliability engineering community adequate tools to perform their job with a measurable level of confidence.

Referring back to the two testing extremes (100% standardized vs. 100% reactive), the reality is that testing all combinations in a complex system is an intractable problem, so an adaptive technique is required to optimize the risk mitigation of the product shipped to the field with a fixed amount of testing time and resources.

4.4.2 Case Study Test Combination - Calculation

For complex systems, such as the one identified to be used in the case study (containing 8 Sub-systems, each with 10 variables), the use of a covering array strategy can be implemented. In system testing, product assurance engineers are able to take advantage of many n-tuple combinations with each test run. The focus of this fault search algorithm is on developing test cases for system product assurance. For the case study, a test case will be comprised of one variable from each sub-system. A new test case could be as simple as changing just one sub-system variable (from a previous test) to see if there are any new dependent two variable faults between the new variable and any of the seven unchanged subsystem variables. With this single test case, seven new (untested) two variable test combinations are created and executed with one system test.

In the case study, there are 10^8 unique system test combinations for the system comprised of 8 Sub-systems (each with 10 variables). Research also shows that the majority of faults will be independent single variable or two and three variable dependent faults (Kuhn et al., 2004). Therefore, taking advantage of n-tuple (2-tuple and 3-tuple) combinations with each 8 variable system test) will greatly reduce the number of test cases necessary to run to cover every 2-variable and 3-variable combination in the complex system. If the product assurance engineer designed a test strategy to cover every n-tuple combination at least once (in the case study), the following would be required:

- 80 - test cases to cover every independent sub-system variable
- 2800 – test cases to cover every two variable sub-system combination
- 81200 - test cases to cover every three variable sub-system combination

The general formula to calculate the number of test case combinations (where order does not matter) required to cover every "*n*-tuples" combination (designated by *r*) with a given total Pool of variables (designated by *n*) is:

$$\frac{\boldsymbol{n}!}{\boldsymbol{r}!(\boldsymbol{n}-\boldsymbol{r})!} \qquad (4.2)$$

Because the focus is on sub-system interaction in system testing, each sub-system is represented by one variable in the system test case. Therefore, the total amount of *n-way* combinations in each sub-system should be subtracted from the total amount of minimal test cases.

The general formula to calculate the number of test case combinations (where order does not matter) required to cover every "*n*-tuples" combination (designated by *r*) with a given total Pool of variables (designated by n) where each sub-system is represented by only one variable is:

$$\left\{\left(\frac{\boldsymbol{n}!}{\boldsymbol{r}!(\boldsymbol{n}-\boldsymbol{r})!}\right) - \left\{\left(\frac{\boldsymbol{x}!}{\boldsymbol{r}!(\boldsymbol{x}!-\boldsymbol{r})!}\right) * \boldsymbol{y}\right\}\right\} \qquad (4.3)$$

where:

r = the number of desired variables in the combination to be tested (two variable combinations = 2)

n = the total number of subsystem variables in the complex system

x = the total number of Sub-systems

y = number of variables in each sub-system

Note: the assumption for this example is that every sub-system has the same amount of variables. This assumption can be adjusted in individual examples.

4.5 Background of Heuristic Search Algorithms

Heuristic algorithms have been developed in many forms over the years, but, with the advancement of desktop computing in the 1980's, a new source of analytical power

increased the development of algorithms used to model and optimize a wide variety of issues. Taking inspiration from nature, several heuristic techniques, such as genetic algorithms and simulated annealing, have been developed in an attempt to model natural evolution or travel patterns. Other popular heuristic algorithms include tabu search, genetic programming, and more exotic methods such as bacterial growth simulation. In most cases, the modeling of a specific problem requires a unique set of logic and decision-making criteria to create useful and efficient tools. The primary reason for developing these types of search algorithms is because developing a model to optimize a complex system with multiple variables can become computationally impossible. This phenomena is referred to as an NP-hard problem. Essentially, the optimization of the product assurance verification process can also be an NP-hard problem. As a result, there is research dedicated to the use of heuristic techniques to develop test cases with the goal of system fault detection. (see next section for literature review examples) The foundation for some of this research, as well as part of the algorithm developed in this dissertation, is the use of a genetic algorithm for optimization.

4.5.1 Related Research in the Field of Heuristic Search Techniques in Reliability Optimization

The following section high-lights relative research in the use of heuristic search techniques in reliability optimization. A brief summary of focused research is presented to give a sense of the current literature. Additional references are listed in the bibliography.

Search-based Software Test Data Generation: A Survey: (McMinn, 2004) This article provides a survey of various techniques used in the field and trends. This source provides good background material and a broad overview.

Test-Data Generation Using Genetic Algorithms: (Pargas et al., 1999) Used for software verification focusing on code branches. It is not as scalable as this research and it is only focused on programming (software) code analysis.

***A Uniform Test Generation Technique for Hardware/Software Systems*:** (Jervan et al., 1999) This research introduces the concept of generating testing techniques for hardware alongside software. The algorithm focuses on describing software and hardware on the same schematic and then using logic to test sub systems. This algorithm is more conceptual and not as scalable.

Breeding Software Test Cases with Genetic Algorithms: (Watkins et al., 2004) This paper focuses on breeding automated software test cases using genetic algorithms. Their research is similar to this dissertation research because the algorithm uses broader search techniques early and local focus later. It is still a traditional GA due to being focused only on software. The fitness function is measured relative to the previous test case vs. an absolute value. Error injection, a popular technique in testing, is used. Errors are injected into the system and the ability of the algorithm to find the problem is measured.

Exploring Very Large State Spaces Using Genetic Algorithms: (Godefroid, 2002) This work introduces the concept of combining two modeling tools into one algorithm. This model uses genetic algorithms for large space search and then combines the feedback of model checking for additional logic. Focused on software, this is another branching search algorithm. It is relevant to this research because a genetic search algorithm is combined with a tabu search in the adaptive genetic search algorithm.

DART: Directed Automated Random Testing: (Godefroid, 2005) A paper with over 700 citations, this dissertation shares similarities to their research because the objectives are a large scale tool for testing by combining three different techniques. This algorithm is only focused on software. DART detects standard errors in the code such as program crashes, logical violations and lock-ups.

From genetic to bacteriological algorithms for mutation-based testing: (Baudry et al., 2005) This paper is an interesting adaptation of heuristic algorithms. The research focuses on imitating the growth of bacteria in software testing. It is similar to this proposal because it is more focused on mutation than crossover in the genetic algorithm search process.

The following table lists additional references in the field of heuristics in test case generation.

Table 4.1: Additional literature review references

The Automated Generation of Software Test Data Using Genetic Algorithms	Sthamer H. H., 1995
Automatic Software Generation and Improvement through Search Based Techniques	Arcuri, 2009
Black-Box System Testing of Real-Time Embedded Systems Using Random and Search-Based Testing	Arcuri et al., 2010
Automated Continuous Testing of Multi-Agent Systems	Nguyen, 2007
Functional Search-based Testing from State Machines	Lefticaru and Ipate, 2008
The Reactive Tabu Search	Batitti and Tecchiolli, 1994
Tabu Search-Part II	Glover , 1990
Specification-based Regression Test Selection with Risk Analysis	Chen et al., 2002
The Capability Maturity Model for Software	Pauik et al., 1991
Exploratory Testing Explained	Bach , 2003
Towards the Prioritization of System Test Cases	Srikanth et al., 2013
Automated test data generation using a Scatter approach	Blanco et al., 2009
Test Cost Optimization Using Tabu Search	Sharma et al., 2010
Sequential Testing of Product Designs: Implications for Learning	Erat and Kavadias, 2008
Value –Based Design of Software and V&V Processes for NASA Flight Projects	Madachy et al., 2007
Human Based Genetic Algorithm	Kosorukoff, 2001

4.5.2 Outline of the Basic Genetic Algorithm

In order to accelerate the growth of the reliability curve during the development process via fault detection and mitigation with fixed resources, this research introduces an adaptive genetic search algorithm that integrates the search for functional defects, interactive defects, and latent defects embedded in a complex system. This fault diagnosis process is focused on the integration of hardware, firmware, and software into one system for test. By introducing test case cost and detected fault risk value into the algorithm, the ability to increase the lifetime value of the product and shareholder value of the producer will improve. This algorithm will take advantage of genetic algorithm techniques to improve test case development design to accelerate fault detection in complex systems.

The following pseudo code serves as the basis for imitating the evolutionary process of nature in order to search for the optimal solution for an NP-hard problem (also see Figure 4.6 for code flowchart)

- *Initial Population: Start by randomly creating a population of potential solutions to the objective function. This solution is often represented by a string or array and is referred to as a chromosome (aka CZ).*
- *Fitness: Evaluate the fitness of each chromosome in the population*
- *New Population Cycle: Create a series of new (evolutionary) populations with the following actions*
- *Parents: Select two parent chromosomes from the population based on their fitness (there are a variety of ways to increase the probability of a chromosome being selected based on their fitness)*
- *Crossover: Cut the original parents and crossover the genes to form a new chromosome (the offspring of the parents)*
- *Mutation: Using a probability algorithm, mutate a determined number of chromosomes at a particular gene*

- *Test: Use the created population to further test for fitness*
- *Logic: If the testing end conditions are met – stop, otherwise repeat the cycle*

For this research, genetic algorithm vocabulary will be integrated into product assurance system test vocabulary in an effort to develop the adaptive search model. For example, the complex system defined in the case study consists of 8 Sub-systems where each sub-system contains 10 variables. In the search algorithm, a system test case will be presented in the form of a chromosome consisting of genes (one for each sub-system). The first digit or space in the chromosome will represent the chosen variable for sub-system 1; the second place represents the variable chosen for sub-system 2 in the test case and so on. These variables are referred to as (sub-system) gene-variables (see Chapter 5 for detailed description).

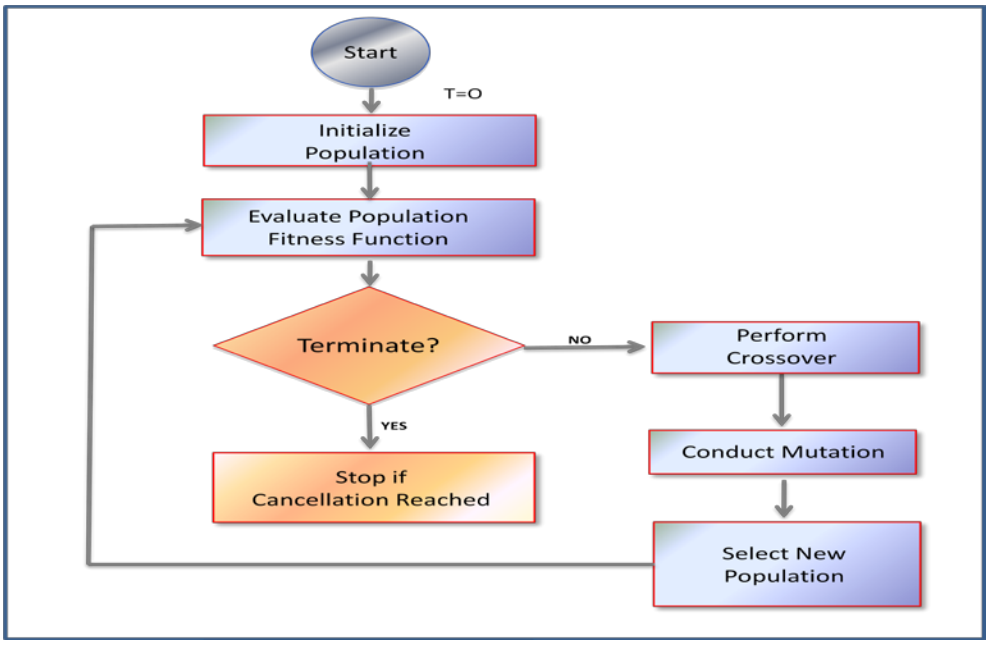


Figure 4.6: Basic logic for a genetic optimization algorithm

4.6 Summary

Chapter four describes the critical role the assurance process plays in the development of products that meet the expected value proposition. A critical, yet often overlooked

aspect of product development is testing, verification and product assurance activities. Unfortunately some products, including consumer electronics, have become so complex that traditional product assurance and reliability engineering processes cannot adequately predict the system reliability, or average life of a product. With the integration of hardware with firmware and software, the number of system combinations requiring traditional product verification testing is impossible. Because of this, undetected system design faults are often embedded in products when they are introduced, and can create unplanned expense to consumers and producers. Product recalls and program updates are becoming a common process in many industries. The societal costs of these escapes, along with the current expense rates of product verification in the design process, create the need for advancements in process and tool development.

The literature review in this chapter identified advancements in research focused on fault detection and test case generation using heuristic techniques. These new fault detection algorithms are primarily in software development which does not present the same difficulty as verifying the combination of hardware, firmware, and software. Because of the possibility of latent and interactive defects in hardware systems, as well as the potential for multiple defects related to one sub component in a complex system, subsystems and interactions must be continually monitored in the verification process.

In the fields of reliability engineering and system assurance, the science of test case (for fault detection) development, with problem resolution management vs. risk analysis and management, is typically managed independently with separate data and value streams. This gap prevents the opportunity to focus verification resources on the test combination with the highest potential payback. In addition, time to market and limited testing resources can be a critical factor that affects verification strategies.

The second part of this research is the development of a broader adaptive algorithm that can integrate the search for functional defects, interactive defects, and latent defects embedded in a complex system. In addition, this fault diagnosis process is

focused on the characteristics of a complex system that integrates hardware, firmware, and software into one system to test. By introducing test case cost, a verification budget, and detected fault risk value into the algorithm, the ability to increase the lifetime value of the product and shareholder value of the producer will improve. By focusing on the primary drivers of the Half-Life Return Model, the ability to create sustainable lifetime value is also enabled.

Whereas the long term benefits of improving the sustainable value proposition will include the integration of total cost as well as social and environmental factors, research focused on the extension of product half-life, material utilization and development resource optimization will play a major role in sustainable product development.

One of the key metrics used in this process is the reliability growth of the system throughout the design lifecycle. This research focuses on breaking this process down in order to improve the feedback model, especially the fault detection and mitigation process. The goal of the next chapter is to integrate the aspects of the reliability growth model, as well as the defined fault types in complex system development, and present an adaptive genetic search algorithm designed to improve the fault detection and mitigation process.

Chapter 5: Complex System Fault Detection: Modeling Through Application of Integrated Framework

In developing tools that aid the engineering community in the design of sustainable products and processes, this research points out the role that feedback plays in the development of complex systems. The creation of sustainable lifetime value involves delivering a product in the most efficient manner that meets or exceeds the targeted value proposition. Beyond product testing, the product assurance engineer focuses on risk management and development resource optimization in an effort to improve the driving metrics of the Half-Life Return Model. During the design life-cycle, system reliability growth is one of the primary forms of feedback to the development community.

The ongoing goal of the product assurance process during system verification is to detect as many faults as early as possible in the development process. In addition, the goal is to show growth in system reliability over the same development period. The engineer is challenged to create a testing strategy and a value system that aggressively grows the reliability curve through strategic test case generation that is not so aggressive that faults are left undetected before the testing resources are fully consumed. In complex systems, the number of test cases required to cover every sub-system variable combination is so large that the ability to run all of them is cost prohibitive and impractical. On the other extreme, if the engineer did not have to worry about latent or multiple faults involving a single test case variable, the use of an orthogonal array test case strategy would cover every independent, two variable, and three variable combination in a very effective manner (Kuhn and Reily, 2002; (Lazic and Mastorakis, 2008; Kuhn et al., 2004). The use of combinatory testing that optimizes the amount of two and three variable combinations within each full system test improves

the efficiency of fault detection. Unfortunately, with complex systems that include the possibility of latent and interactive defects, there is a need to test sub-system variables more than once during the development life-cycle. Essentially, a combination is sought from test case diversity still sensitive to test case cost and potential payback.

In the model simulation for this research, a designed experiment is used to evaluate the independent effect and interdependence of five controlled variables that focus on cost, detected fault assigned risk, test case evolution, test case selection probability, and fault type search priority. The unique contribution of this part of the research is the development of a broader adaptive genetic search algorithm that integrates the search for functional defects, interactive defects, and latent defects embedded in a complex system. In addition, this fault diagnosis process is focused on the integration of hardware, firmware, and software into one system for test.

This chapter introduces an adaptive genetic search algorithm that integrates the search for functional defects, interactive defects, and latent defects embedded in a complex system. In addition, this fault diagnosis process is focused on simulating the characteristics of a complex system that integrates hardware, firmware and software into one system for test.

By introducing test case cost and detected fault risk value into the algorithm, the ability to increase the lifetime value of the product and shareholder value of the producer will improve. By focusing on the primary drivers of the Half-Life Return Model, the ability to create sustainable lifetime value is also enabled. In the model introduced in this chapter, five independent variables are measured in a designed experiment in order to compare the relative affects on the test case and fault detection process for complex systems.

5.1 Model Development - Foundation

In order to create a search algorithm that achieves the desired goals in the research question, a foundation focused on the product assurance process is presented which will lead toward the development of the independent model variables.

5.1.1 Multiple Goals of the Product Assurance Team

The actual testing process is the foundation of the product assurance team's role, but there are multiple goals that make up the entire scope of the PA team's focus on the protection of the customer, business, and the development team (and now society and the environment). Figure 5.1 shows how the variety of deliverables build on each other toward the ultimate target of assurance of the overall solution designed to meet the value proposition. These deliverables are described below.

Product Testing: As the foundation of the product assurance team's role, the accuracy and credibility of physical testing is critical to the long term success of the product. As with any foundation, the other product assurance deliverables are in question if any data or process is compromised.

Test Case Development and Specification Analysis: Beyond the execution of the product verification and certification tests, the design of the test strategies is critical to the overall success of the product assurance team. Because product assurance teams are provided with a finite set of resources and verification time, a strategy must be developed that maximizes fault discovery (as early in the delivery process as possible) and supports the mitigation of the faults with constraints.

Problem Tracking and Management: Once a fault in the system is discovered in test, the problem tracking tool serves as the central repository and risk management tool. The quality of tool management can serve as a direct link to overall return on development investment.

Product Claims and Regulatory Certification: In order to deliver products to customers, especially when solutions are designed for multi-national customers, the adherence to local and federal regulatory requirements is required. In addition, the majority of industries also use certification programs to distinguish product offerings within a competitive family. The product assurance team is responsible for the accuracy and, sometimes, the delivery of the certifications.

Product Delivery Process: The product delivery process is designed to provide a standard process with the goal of meeting the proposed value proposition and expected ROI. It provides the delivery teams an infrastructure, timeline, and criteria expected to be met to ensure success.

Risk Identification: The first five goals of the product assurance pyramid are typical for most businesses, but the next two goals separate testing organizations from product assurance teams. The identification of risk with each problem discovered, or failure to meet a product delivery criteria, must be measured in the form of risk to the business and, ultimately, the value proposition.

Risk Management: Along with risk identification, there is the collection and management of individual risk items identified during the product delivery process. Risk management is not only focused on the identification of risk issues but also the mitigation of the risk.

System Delivery Metrics Integration: The integration of the problem tracking system with the risk management system into the system delivery metrics is critical to the overall success of the product delivery process. In the study of sustainable product development, most of the focus of resource consumption is in the manufacturing and product use portions of the product life-cycle. In some industries, the material and resource consumption during the development process can be a large portion of the overall consumption totals.

Solutions Assurance (customer, business, development team): The product assurance team creates and executes a strategy that seeks to meet or exceed the expectations of the customer, investors, and development team. It is important to look at the entire solution from order entry to delivery to end of life to judge the success of a product.

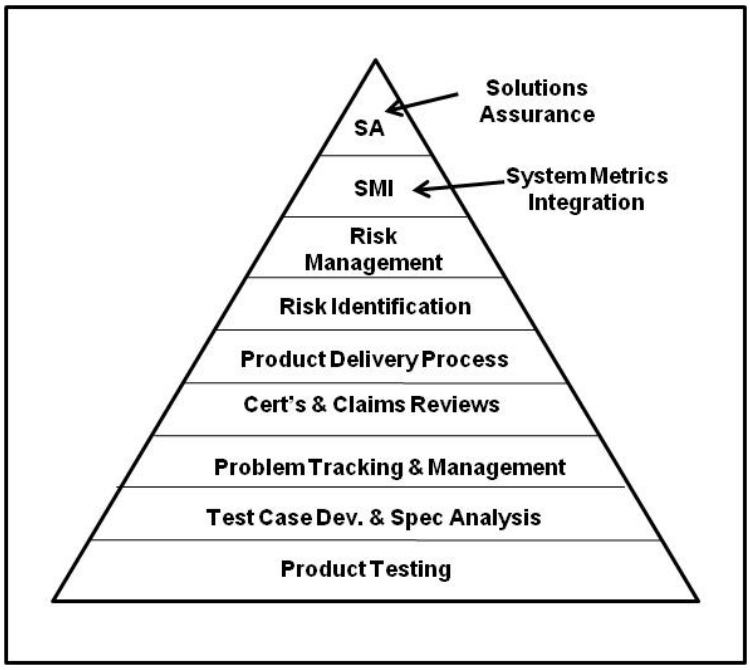


Figure 5.1 Multiple aspects of the product assurance process

The following section presents addition background on the fundamental goals that will be used as the foundation for building the adaptive genetic search algorithm.

An underlying premise in the call for research in heuristic test case development is the need to improve the efficiency of the verification engineer who seeks to find embedded faults in a system with limited resources. The process of fault detection may be a difficult concept for an engineer who does not intend to create a failure point in their original work.

In fact, in order to improve the driving metrics of the Half-Life Return Model, the design team should not just face the reality of the potential for embedded faults in the current design. They should embrace the value of detecting faults as soon as possible. In order

for a design engineer to drive toward the sustainable value proposition and improve the Half-Life Return Model metrics, they must learn how to fail faster. In order to learn how to fail faster, one must first learn how to fail. The same principle applies to the development of the adaptive genetic search algorithm. In order to improve the speed of the fault detection, problem mitigation, and reliability growth of the system, the analysis and understanding of the problem detection process must aid in the improvement of the model.

Complex systems typically contain a high number of interactive design variables, and, therefore, traditional methods of product assurance and reliability verification become an intractable problem. It is especially true with exponential growth in technology and the integration of hardware, firmware, and software in the same (complex) system. Many products are released to the market with the knowledge that defects are still embedded in the system (Jones, 2008). Although the goal of this research is to provide tools to improve the fault detection and risk management process, the tool presented in the form of a search algorithm differs greatly from the typical heuristic problem. The following section introduces the primary aspects of the product assurance process that will aid in the model development.

System Analysis

During the development process, the injection of faults and design defects can and, typically, will happen due to a number of reasons. Defects that are independent from any system interaction are generally discovered during a verification test designed to test the intended function. This type of defect will be referred to as an independent fault. Faults that are due to the interaction of subsystems are more difficult to detect and usually require system verification testing that exercises the various combinations of system interactions during the life-cycle of the product. This class of defects will be referred to as dependent faults. Defects that are due to the specific interaction of two subsystem variables are defined as two variable faults. Defects that are due to the

specific interaction of three subsystem variables are defined as three variable faults and so on. Faults that do not show up until later in the product life will be referred to as latent defects. The role of system verification is a critical aspect of the product design and delivery process. Essentially, it is a form of feedback to the design team. Traditional metrics include the reliability growth of the system during the design phases. Sustainable value is increased as the system reliability increases. By expanding the aspects of the feedback loop beyond reliability, the growth of sustainable value during the Product Delivery Process can be enhanced.

5.1.2 Reliability Growth Model: Dependent vs. Independent Faults in the System Design

Referring back to the primary drivers of the Half-Life Return Model, the length and cost of the development process have a direct effect on the relationship between product half-life and the product's financial success. With that being said, it is not unusual that product delivery dates for complex systems designs are often missed or delayed. Typically, the complication of the verification process and reliability growth during the development phases is underestimated. The integrated framework identified six critical drivers (value creation, robust design, verification feedback, risk management, velocity of workflow, and resource optimization) that, when applied in concert, drive the reliability growth curve toward the ideal state. In essence, the more typical reliability growth curve is experienced because of the break-down of the product delivery process, particularly the coordination between the development and verification processes. In order to improve the reliability growth model for complex systems, it is necessary to break the reliability growth curve down into the various phases.

The Reliability Growth (Analysis) model (RGA model) presented by Crowe (1982), is a simplified curve and assumes a continuous fault detection and mitigation process. In reality, there are different types of failures that affect and or block the fault detection process. In order to improve the modeling of test resource consumption in the fault

detection and mitigation process, there is a need to break the problem down into the building blocks that add to the reliability growth curve.

Taking a closer look at the more typical reliability growth curve (Figure 5.2), there are three distinct zones with two transitions, which is similar to another reliability based model called the “bathtub curve” shown in Figure 5.3 (NIST/SEMATECH, 2013). If you plot the integration of the area under the bathtub curve, it would resemble the typical risk mitigation curve.

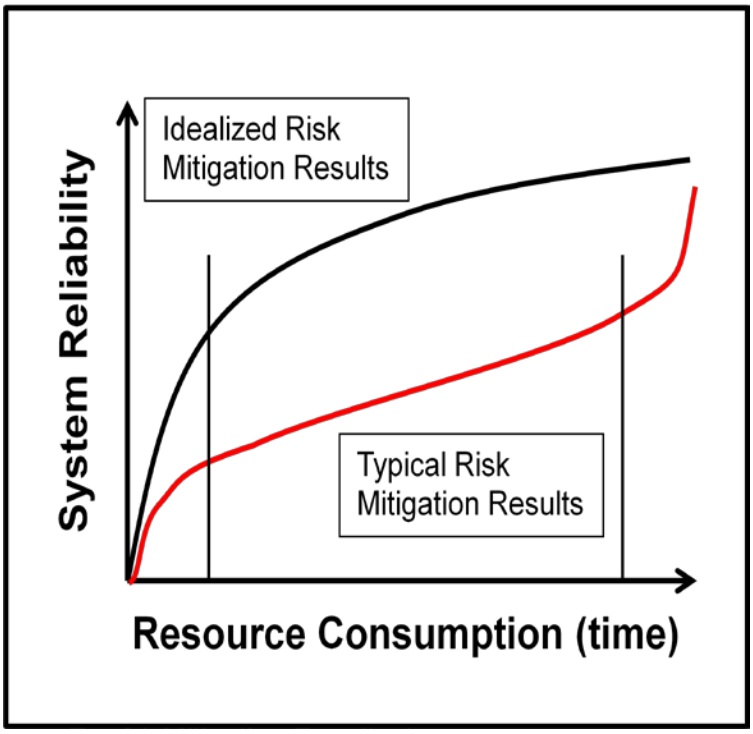


Figure 5.2: Ideal system reliability growth vs. typical curve during development life-cycle

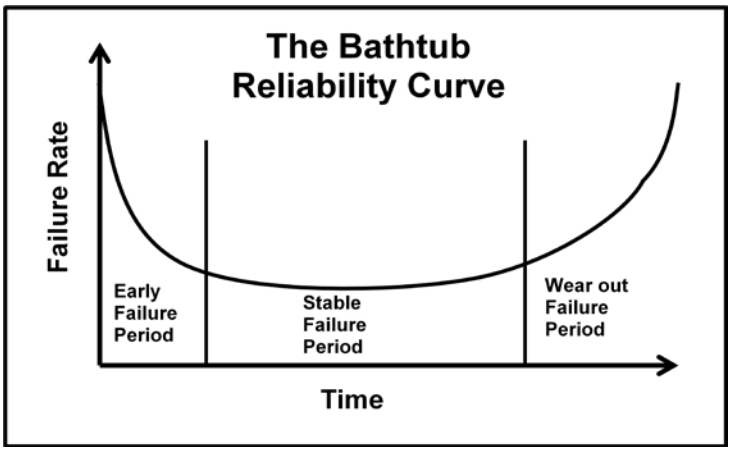


Figure 5.3: Typical bathtub reliability curve over product lifetime

The first section of the curve begins upon delivery and product setup for the customer. This section is represented by high but rapidly decreasing failure rate after the initial set-up. It can be caused by transportation, manufacturing, or installation issues. The origin of the bathtub curve is from actuarial curves and the first section is sometimes referred to as the infant mortality rate. The second section is referred to as the stable failure period and is typically represented by a low failure rate until the product starts to reach its intended end of life. This third section is represented by a rapid growth in failures because of expected latent defects due to material degradation. The three phases of the traditional bathtub reliability curve do provide some support for breaking down the typical reliability growth model in the product delivery process, but it is typically representing the hardware reliability aspects of the design. In reality, there are a variety of failure types that affect the final shape of the curve. By understanding the various types of faults, an improved model can be created in order to drive toward improved product assurance verification processes.

Beyond recognizing the difference between valuable and value add in the product assurance process, taking the time lag between problem discovery and problem mitigation into account while executing the verification strategy is critical to accelerated reliability growth. The concept of taking action early in the development cycle to

accelerate the reliability growth curve is simpler when evaluating systems with limited interdependent factors. Teams developing complex systems often face program delays due to product verifications issues that are typically unexpected. The red curve in Figure 5.2 represents a typical curve that has a mix of both independent factor and dependent factor faults embedded in the system. In reality, the product assurance verification team can't detect all dependent interactive faults until any related independent faults have been discovered and corrected. When verification teams are constrained with limited resources, many standard tests sweep over non-functional variables that note the discovered independent problem, but fail to identify the need to return to the fault area for full system regression.

To simplify this concept, the first graph in Figure 5.4 (a) identifies three important but separate aspects of which the test designer should be aware. They are as follows: faults due to poor functional design (these are typically independent); faults due to variability and system interactions; and faults due to latent/end-of-life defects (typically referenced as reliability errors).

The three types of potential system faults and the typical progression of the particular detection timelines aid in the development of an objective of this research. Focus is placed on the development of an adaptive search algorithm that is more efficient than traditional test strategy methods, maximizes faults detected (given constrained resources), and minimizes the overall embedded risk of the system.

Development engineers should understand the amount of two and three factor interactions that are present in a system when they are focused on robust optimization and tolerance design. One of the primary tools used to isolate these effects is Design of Experiments (DOE) computer software. Little (2011) estimates that on average, 10-20% of effects in system response are due to system interactions. In addition, 5-10% of all effects are due to curvature referred to as (multi-variable) quadratics.

The second chart in Figure 5.4 (b) graphically depicts the effects of the delay in discovering and correcting multi-factor dependent faults until relative independent faults have been discovered and corrected (see Point B in Figure 5.4 b). The same cycle holds for three factor dependent faults and so on (see Point C in Figure 5.4 b). Another reason reliability growth curves are often late in maturity can be explained by the difficulty of testing for latent defects (see Point D in Figure 5.4 b). It is important to remember that the third aspect of product assurance testing is latent or reliability testing. These defects by definition do not typically show up until the end of the product life (classic bath-tub curve). If the product verification test is delayed due to functional or interactive faults, the required testing is delayed for extended life failure points. Development engineers may be caught off guard when an unexpected (and independent factor) fault is detected at the end of the verification test. This independent factor has to be corrected and placed into regression testing. Again, the engineering team often conducts regression tests on the independent design factor but fail to search for complex interdependent faults that may have been present all along but were undetected or infected into the system with the new design.

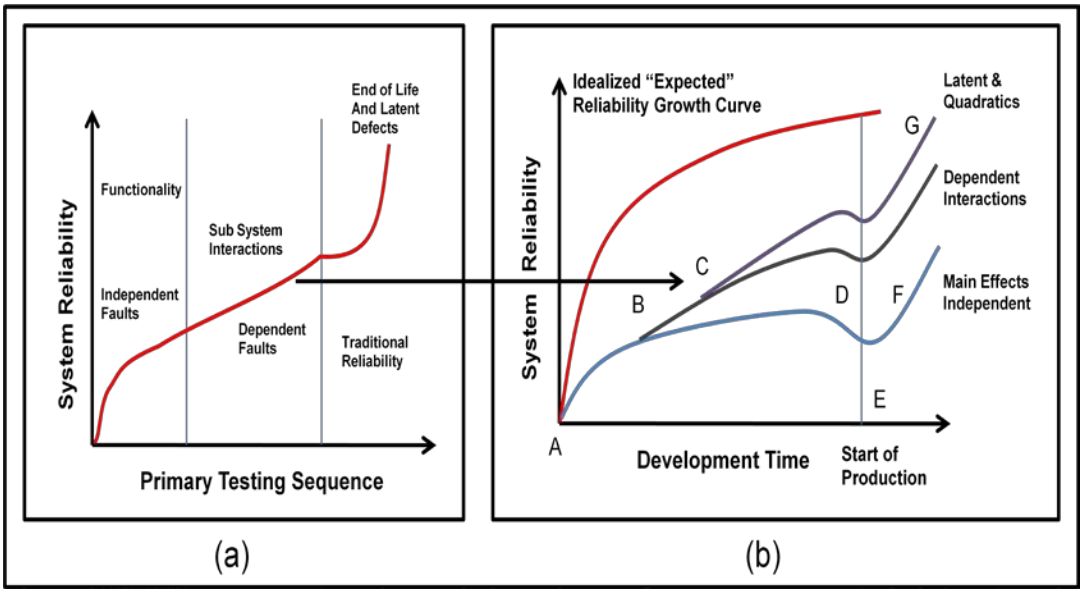


Figure 5.4: Graphical representation of the effects of dependent factors in the reliability growth curve.

5.1.3 Defect (Fault) Type Definitions

By combining the three (types of) reliability growth curves in Figure 5.4, a graphical representation of the more typical growth curve is created. Breaking down the reliability growth curve into driving aspects actually identifies seven different types of faults that must be dealt with when the engineer is in the process of problem discovery and mitigation (Table 5.1). The four primary types of faults are: single sub-system variable – independent faults; dependent faults that involve one variable from two separate Sub-systems, dependent faults that involve three or more sub-system variables, and latent defects that are triggered later in the product life-cycle (latent defects can be independent or multivariable dependent type faults). In addition to the primary fault types, three additional fault states must be recognized and managed in the problem detection and mitigation process. These fault states include the following: a discovered and isolated fault which has not yet been corrected (these faults block the ability to mitigate associated multivariable faults), any discovered multivariable dependent faults that have not yet been mitigated, and two or more faults associated with the same sub-system variable. The last fault type is a special case that has the potential to be the most costly for an engineering team. These faults types are graphical depicted in Figure 5.5. In some circumstances, there may be more than one fault associated with the same sub-system variable. Human nature drives action that may mask the ability to understand and detect another fault associated with a particular subsystem variable involved with another defect. The greatest risk is when one fault is detected but another fault is corrected which, in turn, leads the engineer to assume the original fault has been properly mitigated. This risk is referred to as the bait and switch phenomenon.

Table 5.1: Complex system embedded fault types

1	Independent Fault
2	Two variable dependent fault
3	Three (or more) variable dependent fault
4	Latent defects (fault)
5	Discovered and isolated independent fault – blocking sub-system variables for further verification
6	Discovered and isolated two (or more) variable fault
7	Two (or more) faults associated with one sub-system variable

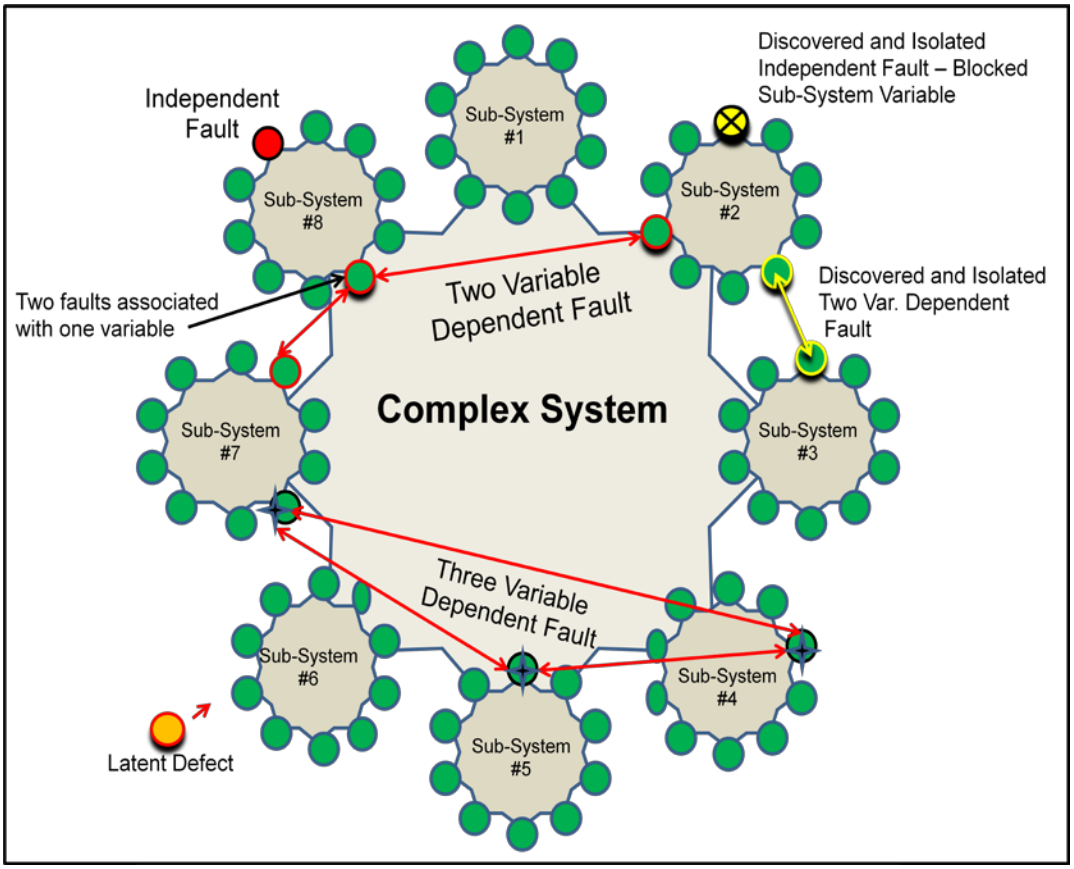


Figure 5.5: Graphical presentation of the multiple defect types in complex systems

5.2 The Integration of Risk and Fault Detection Management

In the fields of reliability engineering and system assurance, the science of test case (fault detection) development and problem resolution management vs. risk analysis and management, are typically managed independently with separate data and value streams (Figure 5.6 (a)). This gap prevents the opportunity to focus verification resources on the test combination with the highest potential payback. The first step to improve the process is the integration of these two major aspects into one model (Figure 5.6 (b)).

At the core of the verification process is the testing of the product in an attempt to validate the design against the specifications and customer expectations. The discovery of design faults during this process is valuable to the engineering team. Value is added to the design once the discovered fault is isolated, the design is corrected, and the system is verified in a regression test. Together, these aspects make up the problem detection and mitigation process in product assurance and are measured with a specific set of problem tracking and resolution metrics. The efficiency and productivity of the verification team is one measure of the maturity of the team. The ability to seek and find faults in an optimal manner not only improves the effectiveness of the testing budgets but improves the efficiency of the overall development team.

The risk assessment and management sub-group may not be associated with the product assurance process as much as the fault detection and elimination group. In order to perform problem resolution management in the most efficient manner, each problem discovered must be examined with some reference to design and customer expectations. In the simplest form, the severity of the problem discovered is noted in order to rank the problems for resolution priority. For this model, a familiar problem risk metric is assigned to each problem discovered. The risk method employed is referred to as a risk prioritization number (RPN), the fundamental metric used to describe potential

faults in a tool called Failure Mode and Effects Analysis (FMEA) (Department of Defense, 1949). By tracking and ranking faults detected in the product assurance process, the engineering team has the ability to create a value system around the performance of their particualr test plans and methods. This value system is used in the adaptation process of the search algorithm.

By integrating the RPN values for the individual faults into an overall system risk management method, the product assurance team can not only present sub-system and overall system risk, but but can take advantage of the risk information and feed it back into the test case generation algorithm. Feedback is used as a driving factor in the adaptive fault search algorithm.

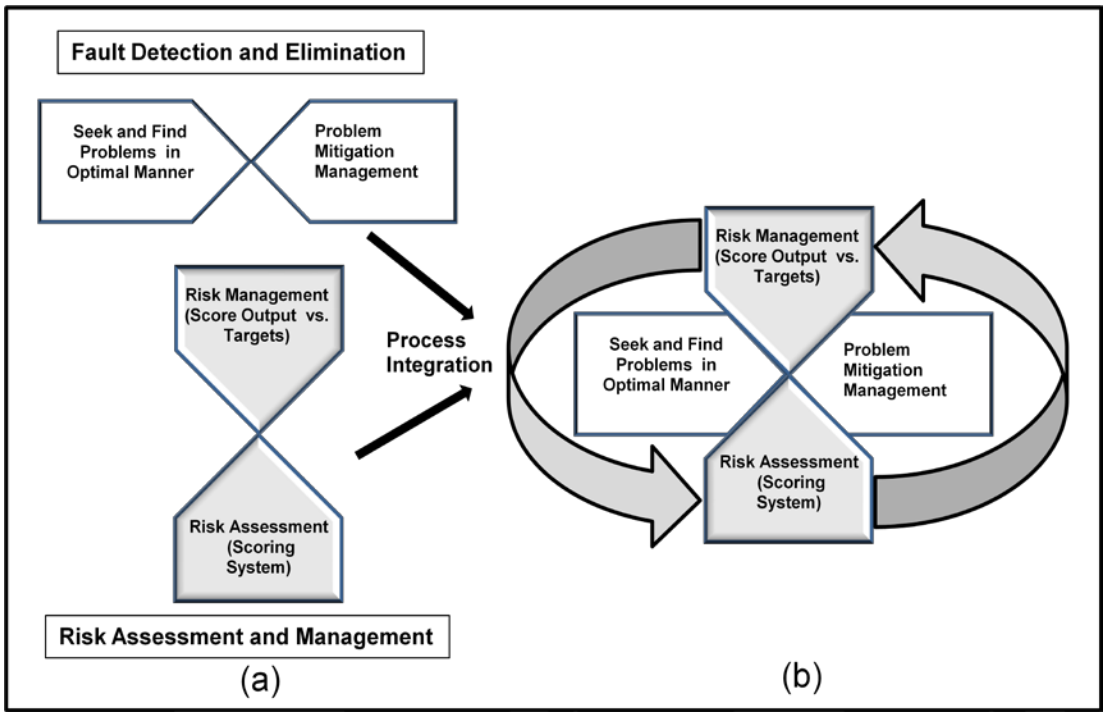


Figure 5.6: The integration of the product assurance deliverables

5.2.1 Integrated Product Assurance Maturity Model

The goals of maximum fault detection and reliability growth during the development life-cycle, given limited time and resources, are enhanced when the two major aspects of the product assurance deliverables are integrated into one system (Figure 5.6 (b)). Just as the development and verification processes are more efficient when integrated into one product delivery process, the velocity of information flow and fault mitigation increases with the symbiotic fault detection and risk management system.

The product assurance deliverables are presented as an integrated maturity model in prioritized order (Figure 5.7). Each sub-group serves as a foundation for the next deliverable. The maturity map can be used to assess the product assurance capability of a given team.

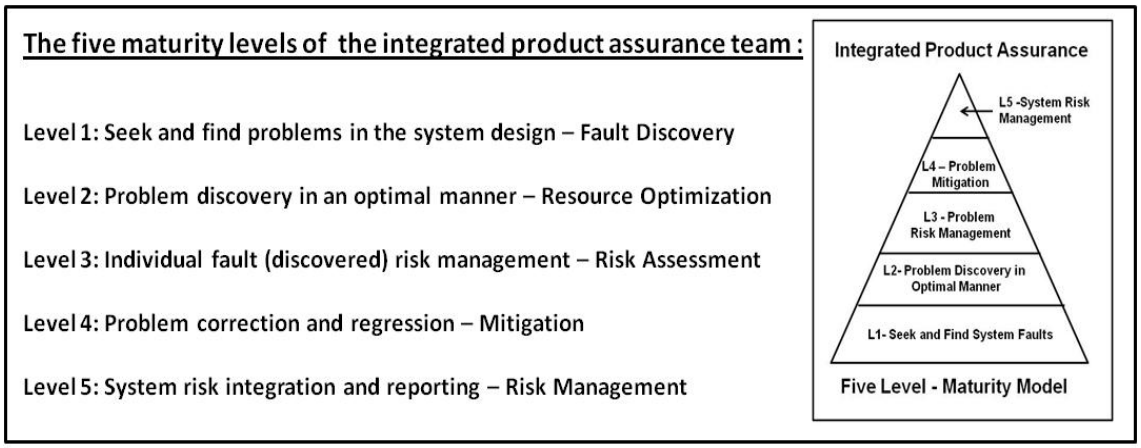


Figure 5.7: The five levels of the integrated PA maturity map

5.2.2 Verification Feedback in the Development Process

In Chapter two, the integrated framework for sustainable product development was described (Figure 2.17). Part of the foundation of the framework was the integration of verification and the product development process such that their symbiotic relationship improves the velocity of the process workflow. The final step in building the schematic

model of the product assurance process reflected in the search algorithm is the integration of the four focus items in the product assurance process into the product design feedback loop (Figure 5.8).

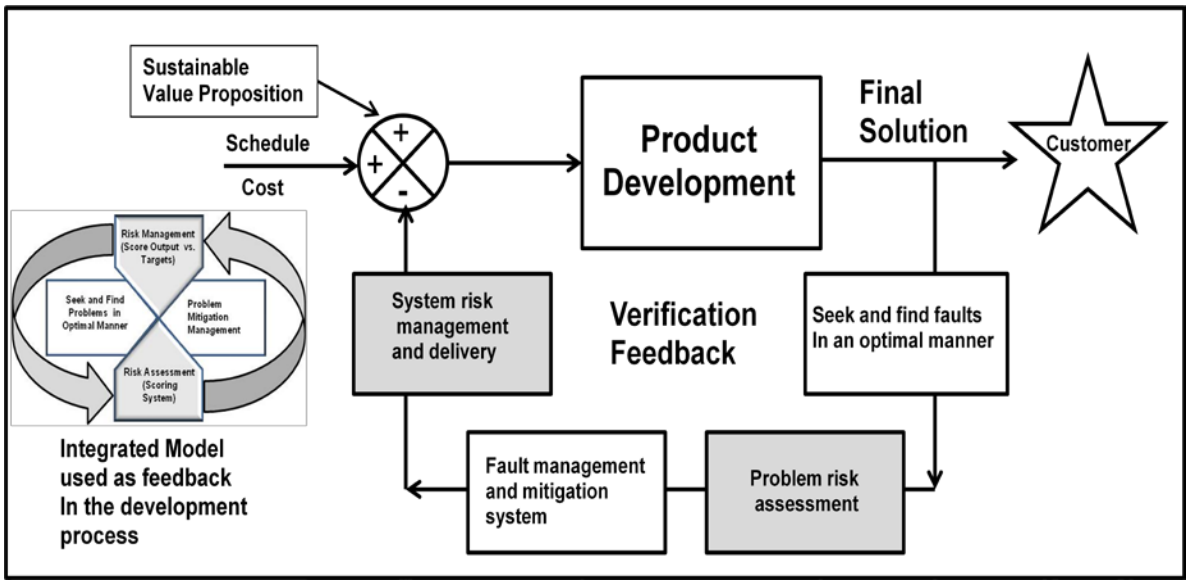


Figure 5.8: The integration of product assurance deliverables into the product design feedback loop

The integrated feedback loop is the first set of foundational building blocks needed to create the heuristic search model. The challenge in the design of an effective search algorithm is the ability to create logic that focuses on the following key aspects:

1. Fault detection, time lag to develop resolution, and then mitigation
2. Risk management and reliability growth tracking
3. Tracking the cost and resource consumption of the system verification process
4. The velocity of the workflow or the relative (time) efficiency of the process

5.2.3 The Effects of Cost and Resource Consumption on the Search Process

The second set of foundational building blocks needed to develop the search model is the understanding of the effects of test case generation and the need for variable diversity, play in resource consumption, and the adaptation process. One of the goals of the search algorithm is to be aggressive enough to accelerate the amount of faults detected early in the process but not be so greedy that critical faults are left in the system undetected before all test resources are consumed. Most research in heuristic test case generation is in the field of software development verification. The reason behind this is that the cost of relative test case is essentially equal (resource consumption) and the only limit to a magnitude of test case executions is computing power and bandwidth. The majority of software verification can be automated. With complex systems, as defined by the integration of hardware with firmware and software, the cost of each defined test includes physical expenses such as models, physical testing facilities (including environmental chambers), and lab technicians. In addition, testing budgets can be inflated if there is required verification of long-life Sub-systems or destructive testing necessary to improve the sustainable product design value proposition. An important aspect of hardware testing is the potential for a large range of resources required to test the variety of sub-system variable combinations. The following section will describe the nomenclature that describes potential test cases for a complex system test. The section will also be used to draw attention the wide range of cost that individual test cases can have relative to each other.

5.3 Fault Detection and Mitigation Model Development

In the case study in this research, the complex system is defined as a system with eight (8) major Sub-systems. Each of the Sub-systems contains 10 variables. The primary

purpose of system tests is to seek and find interdependent faults between the Sub-systems. The search algorithm presented in this research is focused on that goal and also has the ability to detect sub-system (independent faults). It is not focused on the detection of intra-sub-system dependent faults. In reality, many Sub-systems can be defined as complex systems and similar logic is used to detect and mitigate faults before the sub-system is integrated into the final complex system. A fundamental assumption for the development of the search model is the test cases generated are designed to search for interactive faults. Therefore, a test case is defined as a system test that focuses on the interaction of one variable from each sub-system. Figure 5.9 presents a generalized description of the complex system and variable interactions to be considered in the case study. In addition, it is assumed that there is at least minimal system function with at least one variable for each sub-system.

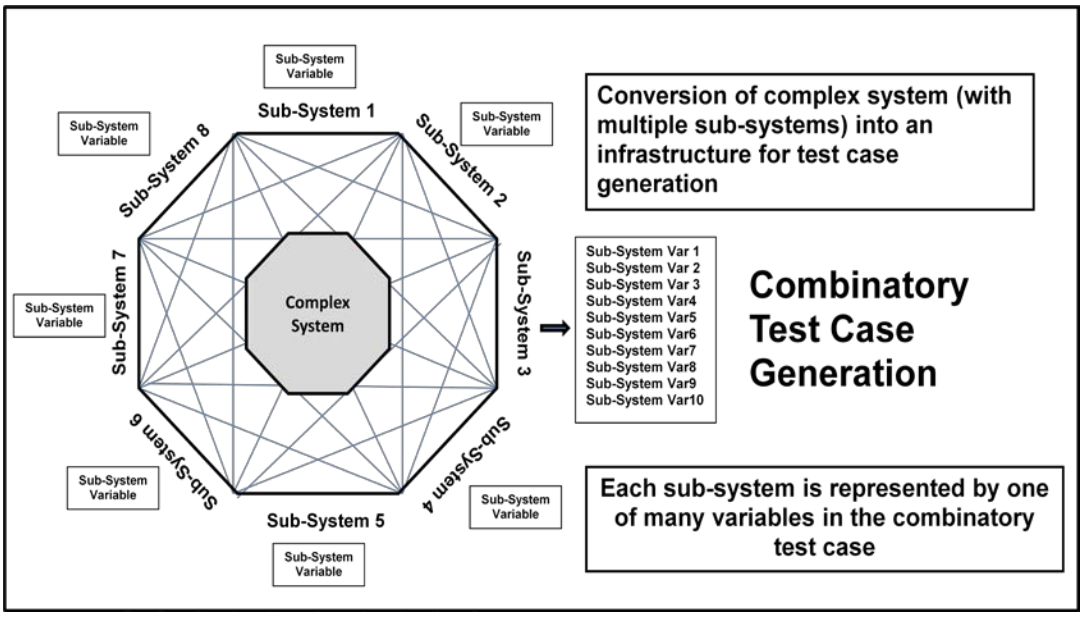


Figure 5.9: Each test case is represented by one identified variable per sub-system

In order to improve the database structure of the test case generation system, the complex system described in Figure 5.9 is converted into a two-dimensional grid with

Sub-systems represented by the columns and the subsequent sub-system variables in the rows below each column (Table 5.2). A test case is, therefore, presented as the combination of variables (also an order set of numbers) representing their respective Sub-systems. For example, the minimal function test case is presented as (1,1,1,1,1,1,1,1), which describes a test case represented by the first variable of each sub-system (see the string of red boxes in Figure 5.10). A system test can be conducted with this particular configuration in order to see if it performs as expected or a fault is detected. In this (single) particular system test case, a number of potential fault types are covered. For example, eight independent sub-system variable are tested (the first variable of each Sub-system) and a large number of two variable and three variable combinations.

A second sequential test case could be represented by the order set of (1,1,1,1,1,1,1,2), which keeps all sub-system variables the same in the system test except for the second variable used in the last sub-system. With this change, a new independent sub-system variable is tested and 7 new two variable dependent combinations and so on.

Table 5.2: Complex system converted to 2D grid

Gene Variable	Test Case Chromosome Set							
	Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Gene 6	Gene 7	Gene 8
	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2
	3	3	3	3	3	3	3	3
	4	4	4	4	4	4	4	4
	5	5	5	5	5	5	5	5
	6	6	6	6	6	6	6	6
	7	7	7	7	7	7	7	7
	8	8	8	8	8	8	8	8
	9	9	9	9	9	9	9	9
	10	10	10	10	10	10	10	10

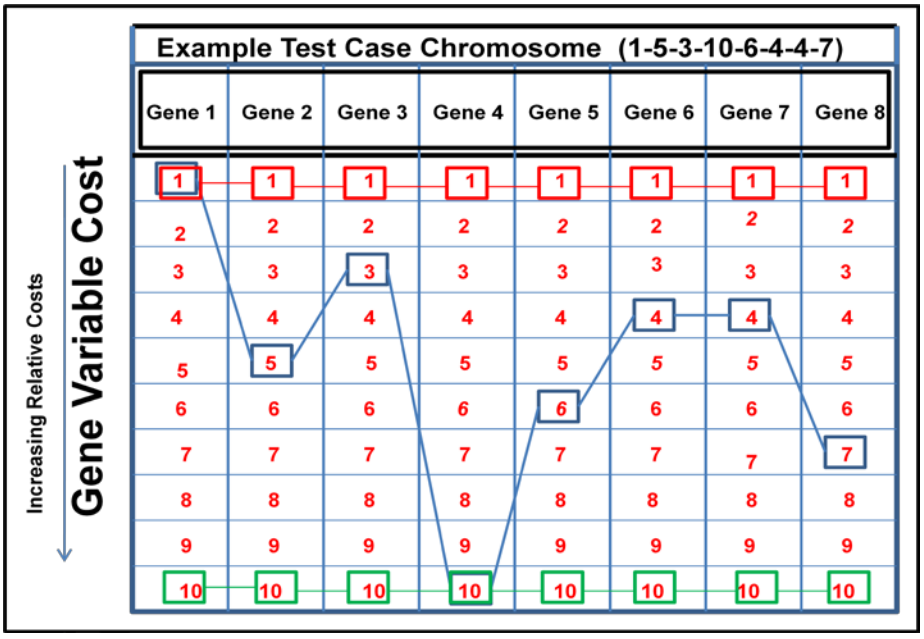


Figure 5.10: Chromosome test case examples

Test case number three is represented by the ordered set (1,1,1,1,1,1,1,3) and so on until test case number 8^{10}, which is represented by (10,10,10,10,10,10,10,10). See the string of green boxes in Figure number 5.10. A typical random test case is represented by the string of blue boxes in Figure number 5.10, which is represented by the ordered set (1,5,3,10,6,4,4,7). For adaptation into a genetic type search algorithm, each ordered set representing a test case will be referred to as a test chromosome. Therefore, for this case study, a test case chromosome will contain 8 variables in order with each digit representing their particular sub-system. These eight ordered values are referred to as genes within the chromosome. The variable representing their specific sub-system genes are referred to as the specific Gene variables.

In actual complex systems, each of the chromosome genes can have variable states and the search algorithm can be modified to reflect this occurrence. For this research, the amount of variable states for each chromosome Gene is set to ten to exercise a difficult degree of potential fault combination for an eight Gene chromosome. As a result, the design requirements for the sub-algorithms, data tracking tables, and probability

algorithms is more difficult than a traditional genetic algorithm. Another reason the genetic makeup of the chromosome is so complicated is that the goal of this research is to create an algorithm that integrates feedback on hardware, firmware, and software. This integration is a critical aspect of the search algorithm because, with the potential for latent defects, there is a need to repeat the testing of the independent and test case combinations. This factor accentuates the need to be conscious of the costs involved with test case creation.

In complex systems, there can be a wide variety of expenses associated with each test case and system configuration. Some tests that exercise a complex system in basic configurations and nominal conditions might cost significantly less than a particular test designed to test the most extreme variable combinations in the most extreme environments. In order to illustrate the effects of cost on the verification process, a generic cost model is applied to the case study complex system. For the purposes of this research, the cost of test cases is described in a generic term referred to as test resources. Test resources can include physical material and labor expense. As an example, consider one unit of test resource to be equivalent to $1 (or any currency). In order to distinguish between the costs of test case variables (a.k.a. chromosome Gene variables), this model assigns the cost of the first Gene variable to be 1 test resource unit ($1), the second Gene variable to be 2 resource units ($2), and so on, until the last variable in each Gene is assigned the cost of 10 resource units ($10) (see Table 5.3). Because the variable cost within each Gene ranges from 1 to 10 resource units, the average cost of any randomly selected variable within each Gene would be 5.5 units ($5.5). This will become relevant in the analysis of the search algorithm when comparing the effects of cost on search effectiveness vs. treating all test cases as equal expense.

To illustrate the effects of cost on relative test cases, the following examples should be considered: The minimal function test case of (1,1,1,1,1,1,1,1) is assigned the cost of ($1,$1,$1,$1,$1,$1,$1,$1) = $8, or eight test resource units. The test case (1,1,1,1,1,7,4) is assigned the cost of ($1,$1,$1,$1,$1,$1,$7,$4) = $17, or 17 resource units to exercise.

In a standard test case sweep intended to cover every two variable combinations between the chromosome Gene variables, it would require a minimum of 2800 unique test cases. A standard test suite is a strategy to check every unique test case combination in order. Given the case study resource cost assignments, the average cost of each gene-variable is $5.5 test units and the average test case expense would be (8*$5.5) $44 test resource units.

Table 5.3: Chromosome Gene variable cost table

Gene Variable Expense ($ or test units)	Test Case Chromosome Set							
	Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Gene 6	Gene 7	Gene 8
	1/5.5	1/5.5	1/5.5	1/5.5	1/5.5	1/5.5	1/5.5	1/5.5
	2/5.5	2/5.5	2/5.5	2/5.5	2/5.5	2/5.5	2/5.5	2/5.5
	3/5.5	3/5.5	3/5.5	3/5.5	3/5.5	3/5.5	3/5.5	3/5.5
	4/5.5	4/5.5	4/5.5	4/5.5	4/5.5	4/5.5	4/5.5	4/5.5
	5 /5.5	5 /5.5	5 /5.5	5 /5.5	5 /5.5	5 /5.5	5 /5.5	5 /5.5
	6/5.5	6/5.5	6/5.5	6/5.5	6/5.5	6/5.5	6/5.5	6/5.5
	7/5.5	7/5.5	7/5.5	7/5.5	7/5.5	7/5.5	7/5.5	7/5.5
	8/5.5	8/5.5	8/5.5	8/5.5	8/5.5	8/5.5	8/5.5	8/5.5
	9/5.5	9/5.5	9/5.5	9/5.5	9/5.5	9/5.5	9/5.5	9/5.5
	10/5.5	10/5.5	10/5.5	10/5.5	10/5.5	10/5.5	10/5.5	10/5.5

Therefore, it would require 2800 test cases at an average cost of $44 to cover every two-variable combination one time. Overall, it would equate to a total cost of ($44*2800) $123,000 test resource units. Unfortunately, with the combination of hardware with software and the potential for latent defects, it requires multiple sweeps of these combinations to monitor for system faults. With the goal of creating a model in

pursuit of the idealized reliability growth curve, taking multiple sweeps to detect the majority of faults in the system will normally consume all of the test resources before all faults are detected (see graph in Figure 5.11).

The problem becomes exponentially worse if the intent is to use a standard test sweep to detect every three variable combinations in the complex system. In a standard test case sweep intended to cover every three variable combination between the chromosome Gene variables, it would require a minimum of 81,200 unique test cases. Therefore, it would require 81,200 test cases at an average cost of $44 to cover every three-variable combination one time. This would equate to a total cost of ($44*81,200) $3,587,200 test resource units. Again, this would cover only one sweep of all three variable combinations which, in turn, runs the risk of not detecting any similar latent defects (Figure 5.12).

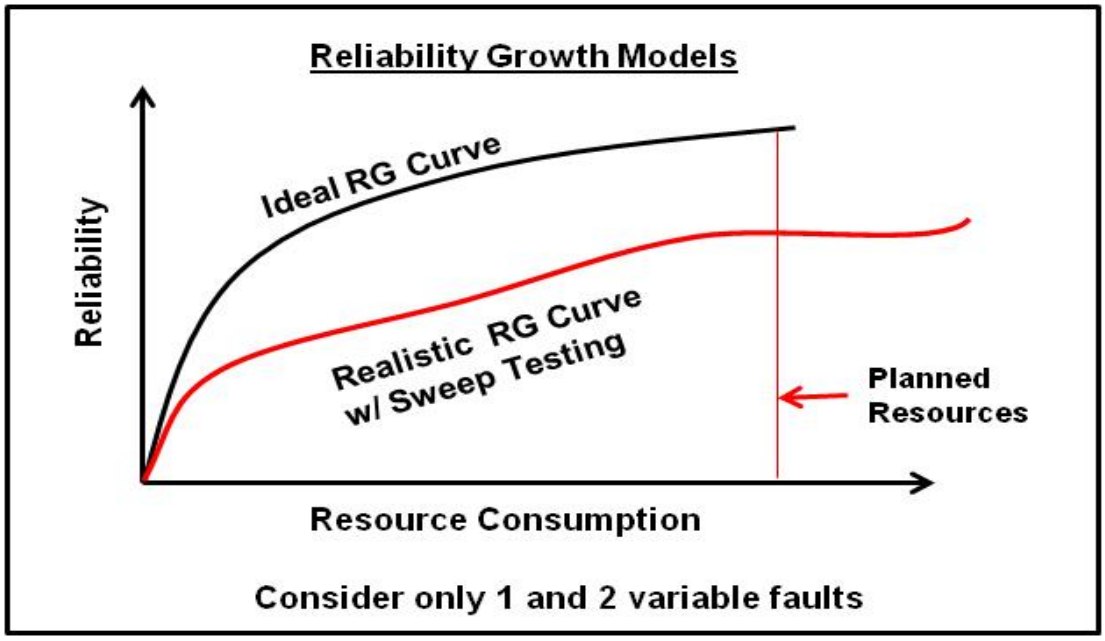


Figure 5.11: Example of 2-variable combination standard test sweep

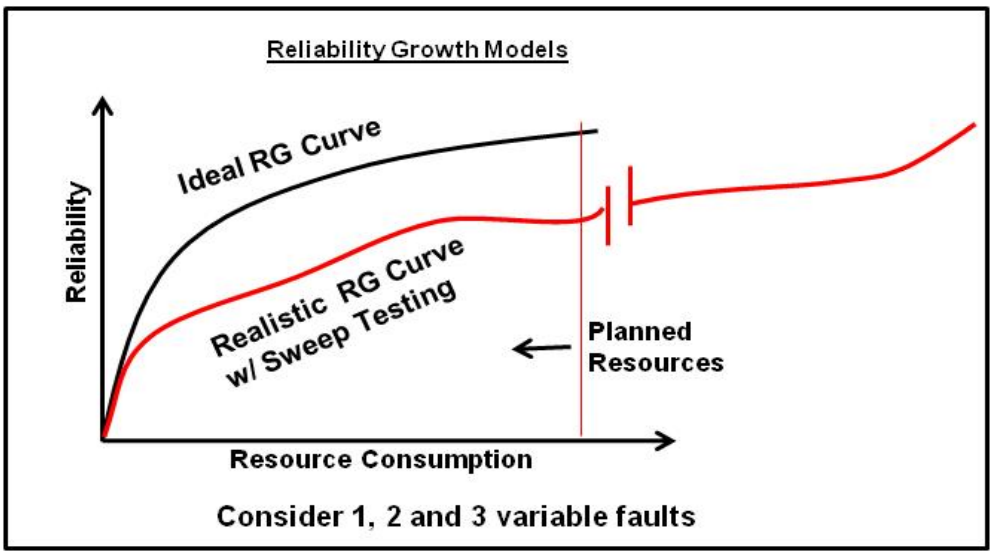


Figure 5.12: Example of three variable standard test sweep

Most complex system will not have the same size and shape as the example used in this case study, particularly the broad range of test resource expenses for each chromosome gene. The case study is set up this way to exercise potential search algorithms in difficult conditions. It does represent the reality of some test conditions being relatively less expensive to run than others. Many engineers choose to run a system test with the least expensive set-up minus their focused set of variables. In a similar manner, one of the goals of the search algorithm is to be aggressive enough to find faults as early in the process as possible without being greedy enough to consume all the test resources before the faults are all discovered. A greedy algorithm could search for faults with test cases that combine the most expensive variables in each gene. This algorithm has the potential to quickly consume the test resources when, in fact, many faults might be embedded throughout the complex system. The goal is to take advantage of the most cost effective test cases that still succeed in maximum fault detection. One of the interesting choices that product assurance engineers often face is choosing between ignoring the individual sub-system variable costs to assure the broadest coverage or taking advantage of the cost variance to optimize the available resources. This choice will be one of the primary independent variables considered in the search algorithm.

5.3.1 Fault Mitigation Process (Three Stage Process)

The final set of foundational building blocks needed to develop the search model is the understanding of the true cost of fault detection and the mitigation process in test case generation. This research points out that in the product assurance process, detecting a problem in test is valuable, but value is not added until the problem and risk have been mitigated. In the product assurance maturity model, this research points out the five levels of adding value in the verification process. The act of discovering faults is the first maturity level of a test organization. To drive faults out of the system and improve the reliability growth curve during the development process is to improve the return on investment in the sustainable products Half-Life Return Model. Therefore, it is important to consider these steps in the development of the search model. Many test engineers create a long term testing strategy with the assumption that test resources are consumed only for discovery of system faults. In reality, system fault discovery is just one of three major area of the product assurance process where testing resources are consumed. This research will refer to the three areas of consumption as resource Pools.

In Chapter four, seven types of faults were identified in the development of the search algorithm. The discovery of faults in a complex system is only the first step in the risk mitigation process. In addition to a continued search for additional faults, resources are consumed in the process to isolate and correct discovered faults. In the product assurance process, resource consumption is categorized into three Pools (Figure 5.13), which include the following:

1. General large scale fault search – the idea with this resource Pool is to cast a wide net in the test case variable combination in order to detect a potential fault.
2. Fault Isolation – the use of additional test resource to isolate the specific faulty Gene variables within the system test case.

3. Regression and Release – the use of additional resources to test the design improvement and verify the potential fault correction.

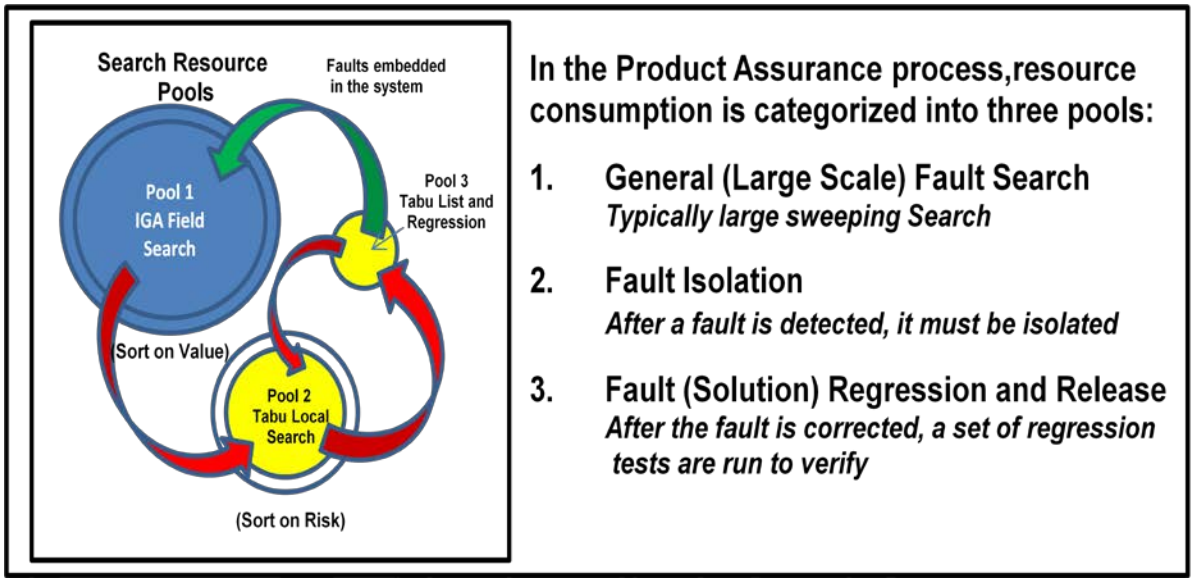


Figure 5.13: Three stages of fault discovery and resource consumption

In the adaptive genetic search algorithm (and related computer program), there will be three distinct processes that perform the unique requirements of each Pool. In the spirit of heuristic designs, real world analogies will be used to explain the multiple activities taking place in the search and risk analysis algorithm used in this research.

The difference between the first Pool algorithm and the second Pool algorithm is similar to a strategy used in the sales and marketing industry referred to as the "hunters and farmers sales process" (Brown and Miller, 2008; Shapiro, 2002). Just as sales people develop strategies to find new clients, this research refers to the goal of seeking and dealing with faults according to the "Hunter, Farmer, Warehouse Manager" method. The hunter/farmer terminology is used in sales and marketing literature to emphasize a method that optimizes the revenue generating process with limited sales resources. In the adaptive genetic search algorithm presented in this research, the hunters refers to the process used in Pool one, where a portion of the total available resources is used to find new faults just as a portion of a sales team is used to discover potential clients (or

sales leads). Once a sales lead is identified (or a new fault is discovered), the potential customer is handed over to a different sales person who is more adept in closing a sale, inferring a different skill or process. In the search algorithm, once a new fault is discovered, the test case that discovered the fault is taken through another (different) type of search algorithm that is focused on the isolation of the specific fault within the system test case. It is important to note that, when a fault is discovered with a particular test case, a trained engineer will usually have a good idea of the exact sub-system or set of sub-system variables that caused the fault, but additional testing is required to verify and isolate the fault. It is identified as the suspected fault (sick) Gene variable(s). In the genetic search algorithm nomenclature, a test case that identifies a fault in Pool one is referred to as a "sick chromosome" and the isolated fault is referred to as the "sick gene-variable" (or combination of variables). The third Pool is referred to as "the warehouse manager."

5.3.2 Detailed Description of the Three Resource Consuming Processes

Pool 1: The "Hunting" search algorithm - This group of resources is dedicated to seeking and discovering any possible fault in the system (with limited intelligence). Often this group seeks to cover broad swaths of territory to flush out system problems and faults. In the algorithm, an adaptive genetic search will be used to minimize the amount of test combinations that could potentially pay off the highest rewards (i.e., detecting the highest risk faults with the minimal amount of resources). As in real life when a "sick" chromosome is discovered (test case detects a problem), further testing is necessary to isolate the problem. Once a problem is detected, it is moved to Pool 2.

Pool 2: The "Farming" search algorithm – This group of resources is dedicated to isolating the sick Gene or combination of genes within the chromosome once it has been handed over from the hunters. This process is necessary to properly correct the fault, but it is a different search strategy than the general search process. For that

reason, a different search algorithm will be used and modeled after tabu search techniques. Once the sick gene(s) are identified, the identified fault and original test case are placed on a tabu waiting list for correction regression in Pool 3. Once a fault is isolated and is determined to be an independent (1 variable) fault, it is removed from the available relative subsystem Gene Pool until it has been corrected and verified. This real world example illustrates the most efficient method to conduct system verification. A defective sub-system design which is independent from any dependent variable fault(s), should not be available for system test because it only consumes test resources for an invalid system that will be redesigned. In doing so, the Gene probability table must be properly updated to spread the probability of Gene variable selection in Pool 1 test case generation. In the process of transferring the test case ("sick chromosome") to a second search algorithm, a suspected Gene variable is identified along with the rated RPN value for the fault. These two data points are typically provided by the test engineer and will be used in the isolation process.

Pool 3: The "Warehouse Manager" Regression Algorithm – This group of resources is dedicated to holding the detected faults and then conducting regression tests on problems in the form of "sick/isolated" chromosome test cases that have been released by the engineering development team. One of the most important aspects of the product assurance engineer's role is conducting a full set of regression tests once a previously detected problem has been corrected. Beyond confirmation that the original problem has been corrected, the test strategy should seek interactive (dependent fault) problems that may have been masked by the original problem. If an independent fault was corrected and released, the Gene probability table in the search algorithm needs to be updated to assign the appropriate probability for future selection in the Pool 1 test case generation algorithm. In addition to the regression function in Pool 3 algorithm, another critical function is modeled.

After a fault has been discovered and isolated, it is held in a problem tracking system. In this model, the tabu list indicates that the fault has already been detected and that it

has not yet been corrected and mitigated through regression testing. In reality, additional resources and time are required to correct the problem. In the algorithm designed for this research, resource consumption is identified with a generic term, test resource units. The same measure is used to indicate the passage of time required (in the form of resource units) to identify a potential solution for the discovered fault. In the model, a separate bank account of test resource units is allocated to Pool 3, to track the amount of appropriate consumption before the potential release of the fault for regression testing.

5.3.3 Test Case Resource Consumption Summary

With the aim of providing a richer set of feedback during the development process, the product engineer should focus on risk management and resource consumption as a means to improve the value proposition. By integrating risk management and resource consumption into the feedback loop used to adapt the test case generation process, the goal is to improve the effects of maximum fault detection with limited resources and to improve reliability curve growth once a detected fault is mitigated.

In reality, there are limited resources available for the fault detection and mitigation process that must be divided between the three search Pools (Figure 5.14). The working model should be able to track the overall resource budget, Pool allocation, and Pool consumption. In addition, the adaptive search algorithm should be scalable in order to accommodate the degree of resource consumption required to achieve desired reliability growth results for any particular complex system.

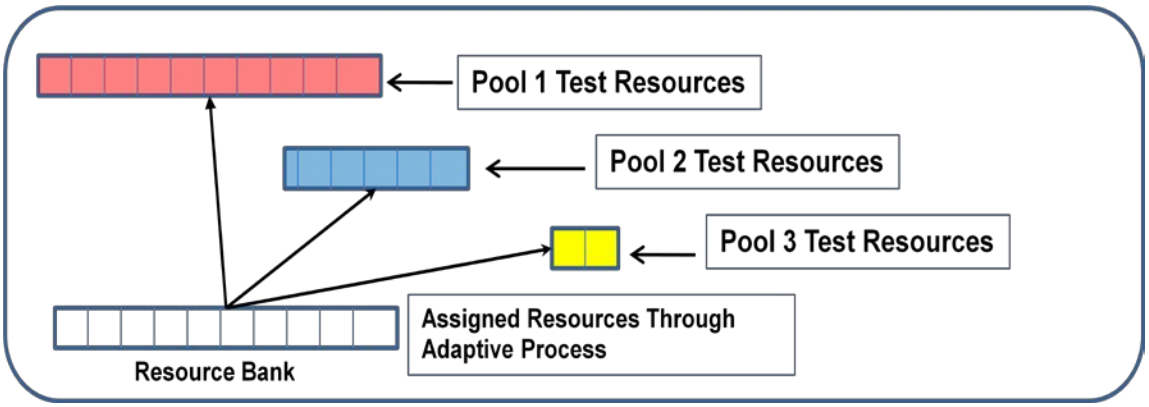


Figure 5.14: Resource allocation bank and consumption Pools

5.4 Adaptive Genetic Search Algorithm Model Objectives

This section's primary focus is to integrate the defined aspects of the product assurance verification and risk model. It addresses the development of a heuristic search algorithm that provides the ability to analyze the effects of primary independent variables against the competing goals of maximum fault detection and minimal system risk with limited resources. A detailed list of product assurance fault detection and mitigation models was presented in the previous chapter. The primary issues or requirements for a successful model of complex systems include the following:

- Current heuristic search models assume all things are equal with an unlimited test budget. In reality, the costs of test variables can vary greatly. Test budgets are finite.
- Current search models do not consider feeding risk back into the model. The degree of relative risk of detected faults is not equal.
- Current fault detection models assume a binary (or pass/fail) result. They do not adequately account for latent defects. In complex systems, multiple faults can be associated with the same test variable. Test results can create a phenomenon called "Bait and Switch," where one fault is detected but another fault is isolated and corrected.

- Faults can be independent (functional), interactive, and latent which include end of life reliability.
- Complex problems can be masked or hidden from the testers search capability. These types of problems are typically dependent multi-variables undetectable until an overriding independent problem has been detected and corrected.

5.4.1 Search Model Goals

With the advancement of computing power and inexpensive memory, the use of technology to advance the art of system assurance and product delivery has continued to grow. In developing advanced tools to assist in the creation and study of test case development, many aspects should be considered. For the adaptive genetic search model and algorithm that is focused on the improvement of sustainable product development, the following goals are identified:

- The model should integrate the four aspects of the Product Assurance Management Model into a search algorithm. It requires a set of metrics and interface points that allows the value of test resources and risk mitigation to be integrated into the search algorithm.
- Develop a fault detection search algorithm for a given complex system and identify by Sub-systems and sub-system variables. The general hypothesis is that an adaptive search algorithm, with multiples search groups (test resource Pools), will be more efficient in the fault detection and risk mitigation process vs. a more traditional grid or even random search testing methods, which are known to be an NP-Hard problem.
- Develop a series of interactive sub-algorithms that are necessary to conduct multiple prioritized concurrent searches. This development requires splitting the resources into sub-Pools to conduct several separate and unique fault detection or isolation actions. Because this search algorithm differs from the traditional optimization problem of searching for one optimal point, the

tracking of resources and discovered faults is necessary. There should be three primary search Pools: a general search modeled by an adaptive genetic algorithm; local Gene isolation, modeled after modified tabu search; and fault correction regression testing.

- Develop a suitable risk management system that will aid in feedback and reliability growth in the solution assurance process. The objective is to integrate assigned risk priority values for each detected fault and develop a risk mitigation tracking system.
- Use the assigned risk values for the detected faults to feed back into the adaptive fault detection algorithm and steer the verification process toward the test variables that would potentially create the most value. The assumption is faults are embedded in the system throughout the design process. By utilizing the fixed assurance resources in an adaptive manner, the fault detection and reliability growth of the product in the field will increase.

For this research, instead of verifying the effectiveness of potential search algorithms against multiple physical systems, a virtual complex system is created with embedded faults. The same set and location of the faults will be used in the designed experiment, although the search algorithms will not know the location of the faults. As the search algorithms are exercised, the test case is presented to the complex system (fault) simulator. If the test case detects the embedded fault, the simulator returns the appropriate information. The case study will be used to study the effectiveness of this model.

In order to complete the system search algorithm, additional databases need to be developed and integrated into the source code. These databases should include resource tracking, test case cost menu, fault detection history, and the Gene probability table.

The following section summarizes the specific objectives for the adaptive genetic search algorithm.

5.4.2 Adaptive Genetic Search Algorithm – Model Objectives Summary

1. Recognize the goal of fault mitigation and risk reduction with fixed resources.
2. Split the resources over three distinct search goals: broad search capabilities, local defect isolation, and fault correction regression
3. Once a fault is discovered through the isolation process, the discovered fault information and the original discovering test case should be transferred and held in a database. This information could be utilized as a potential tabu list.
4. In order to improve search efficiency, the algorithm should have the ability to block specific chromosome (sub-system) Gene variables that have been isolated as faulty. This blocking will prevent the specific Gene variable form being selected in a potential test case until the fault is mitigated.
5. Prioritize the discovered and isolated faults by assigned RPN risk levels.
6. The use of an adaptive genetic algorithm will allow the risk value attached to a detected fault (in the form of a risk prioritization number (RPN)) to serve as the primary driver in the mitigation of resource allocation function.
7. The system should be rewarded for discovering embedded problems as quickly as possible.
8. The system should not be so greedy that the search algorithm completely misses pockets of potential faults.
9. The system should be flexible enough to carry on multiple searches (with local interrogation).
10. The algorithm must be adaptive. Probability of Gene selection should be modified based on previous fault detection history.
11. A test case value system must be integrated into the algorithm to maximize resource utilization. Design a resource allocation and consumption tracking algorithm.

12. As part of the value system, the algorithm should have the flexibility to create any number of potential test cases with a calculated cost. The user should have the ability to select the amount of test cases to consume per Pool generation to control the resource consumption and greediness of the search.
13. In order to be adaptive, the algorithm should be able to modify or adapt the search based on feedback that includes risk metrics. For example, changing the allocation of resources to the Pools (or multiple search engines) depending on the sub-system feedback.
14. The overall model must be scalable. Complex systems contain a large amount of design variables and, therefore, the algorithm must be scalable, yet still remain efficient and manageable.
15. A "glass box" system fault simulator is required to test and verify the algorithm. This system includes a case study with all four types of identified faults embedded in the fault simulator. The simulator serves as a surrogate representative of an actual test where the faults are locations and related data are pre-determined (in order to analyze the efficiency in any relative search algorithm) but not given to the algorithm.

5.5 Model Description

5.5.1 Analysis Focus Areas

In creating an algorithm designed to study the major drivers and interactions that affect the efficiency of fault detection in a development process, focus will be placed on five key aspects of the model. Complex system verification is a complex problem and, therefore, there are many potential variables that can be adjusted to study their particular effects on the search algorithm efficiency. A designed experiment will be used to study the significance of the individual values and interactions between the five input signals. The focus areas include the following:

1. **Test case expense**

   When comparing the relative cost of two potential test cases, it is important to note that more test cases can be executed if the average cost per case is lower. The issue may be in the lack of ability to cover all combinations and reflecting a test case generation process that may be too greedy or too passive. For example, if too many cost corners are cut in the verification process, the effectiveness to find all faults may be eliminated. This study will analyze the effectiveness of the designed algorithm focused on taking advantage of relative costs vs. treating all potential test case combinations (chromosomes) the same.

2. **Sorting discovered faults based on risk**

   Since the goal is to accelerate the growth of the system reliability curve, it could be beneficial to first place priority on correcting the problems with the highest RPN number. This study will analyze the effectiveness of the search algorithm by comparing the process of correcting discovered faults in the order they were discovered or in the order of the highest to lowest RPN ranking.

3. **Imitating nature I – Crossover and Mutation**

   In the development of complex systems, some Sub-systems can be affected by defects more than others for a variety of reasons. As a result, one strategy is to focus a larger percentage of test resources on areas where previous defects have been discovered ("smell the blood method"). This study will analyze the effectiveness of the algorithm by comparing the use of genetic algorithm techniques (crossover and mutation) in the creation of new test cases vs. not taking advantage of information regarding previous fault detection.

4. **Imitating Nature II – Genetic Algorithm Probability Modification**

   In the process of creating potential test cases in the genetic algorithm process, a random number generator is used to choose the representative variables for each sub-system. Initially, there is an equal chance of all variables within each sub-system being chosen. As previously mentioned, one strategy designed to improve the effectiveness of the fault search is to focus a larger percentage of

test resources on areas where previous defects were discovered. Another method that may accomplish this objective is to change the probability of a particular sub-system (chromosome gene) variable being chosen if a fault has been previously associated with that variable. It should be noted that, if the fault is designated as an independent fault type, the sub-system variable is not available to be chosen for a new test case until the fault has been corrected and mitigated.

5. **Prioritize sweep testing of all independent sub-system variables first**
   The goal of the search algorithm is to create the most efficient test case development by using feedback during the fault search process and adapting the test case generating strategy. The majority of search efficiency is gained by taking advantage of two and three sub-system variable combinations in the same test case. In the process, most independent sub-system variables are covered in a short amount of time but not in a systematic process. This study will analyze the effectiveness of the algorithm by comparing the strategy to check all independent sub-system variables before the use of adaptive genetic algorithm vs. jumping directly into the adaptive combinatory testing.

5.5.2 Overview of the Integrated Adaptive Search Algorithm

There are six primary sub-algorithms, several data tracking tables, and a complex system test case simulator required in the adaptive genetic search algorithm (Figure 5.15). The major algorithms are as follows:

1. Command Center (tracks resource bank/consumption, risk management data, scorecard, etc.)
2. Program Parameter Initialization and DOE Switches
3. Resource Pool 1 – Genetic Test Case Generator
4. Test Case Queue

5. Resource Pool 2 – Fault Isolation Via Tabu Search
6. Resource Pool 3 – Fault Management and Test Regression
7. Complex System Test Case (Fault) Simulator

A description of each section follows.

Command Center

The command center serves as the data bus and graphical user interface (GUI) for the user. In addition to the GUI, the command center has two primary sections. The first section is the central repository or test resource units. The amount of resource units is delivered to each of the consumptions Pools as determined by the tool user. Second, consumption metrics are tracked and recorded in the database and used to potentially adapt the resource allocation process.

In order to achieve the goal of optimizing the available resources in the three Pools, the modification of available resources over the course of the testing process is useful. In the early test phases, a broad spectrum of testing (Pool 1 search) may be more valuable than the other Pools.

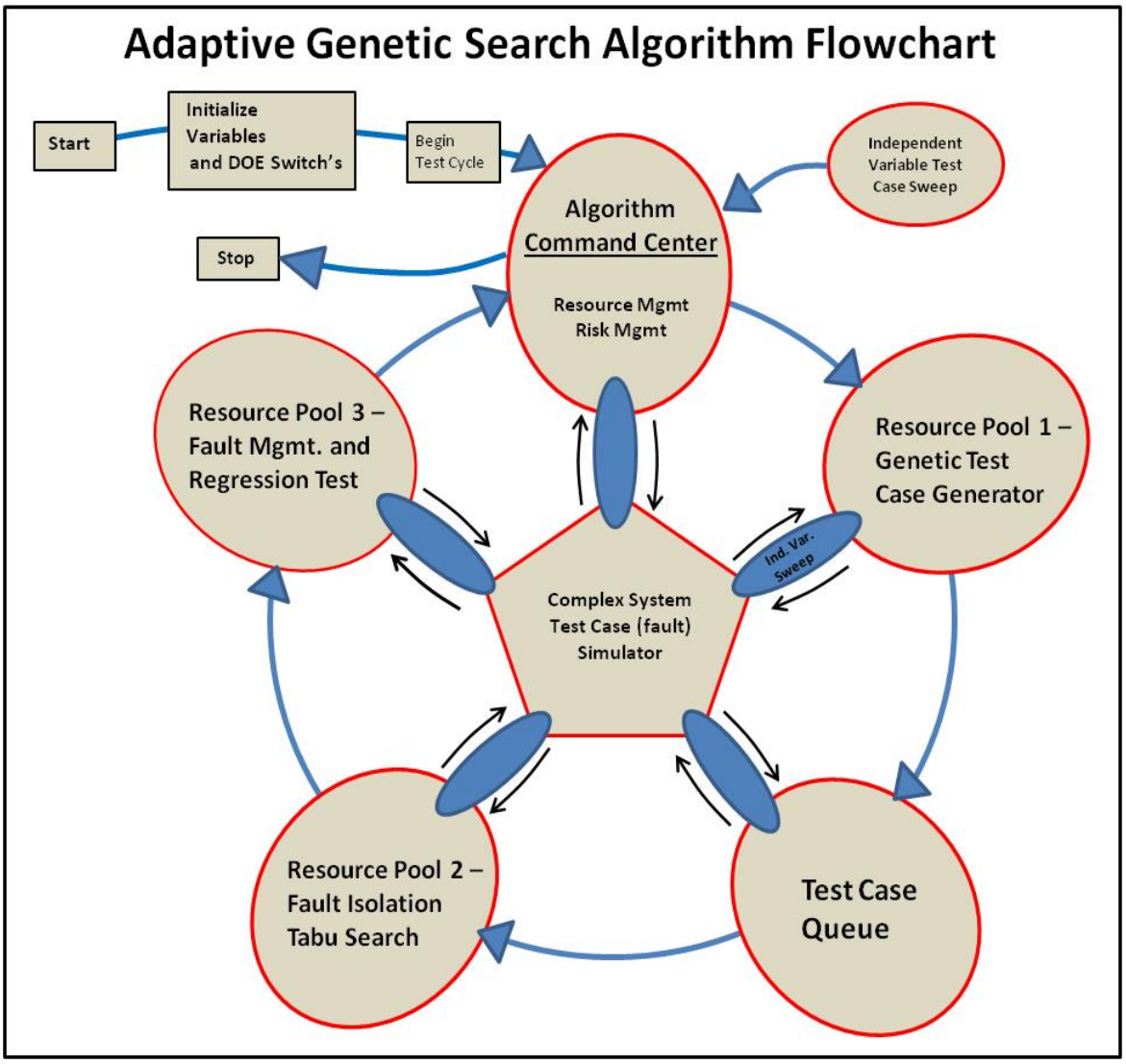


Figure 5.15: Adaptive Genetic Search Algorithm Concept Map

Once a number of high values (relative RPN risk scores) have been discovered, the reallocation of test resources between Pools could improve the optimization model. The algorithm could be based on relative cumulative-RPN scores in each Pool or an advanced method could utilize Bayesian networks to judge the relative risk between Pools. The computer program written for the case study allows the user to modify the distribution of resources per program cycle and initial Pool allocations at the start of the algorithm.

The second section is focused on data collection and analytics which are reported via the tool dashboard. There are a large number of metrics that are tracked and presented in response to the desired output signals as well as system debug information and fault detection timing patterns. The following list of the primary feedback metrics is presented in the dashboard:

- Isolated Faults (number) in order of discovery
- Cumulative resource consumption at the time of each fault discovery
- Resource count for the release of faults from the tabu list
- Breakout of resource consumption for the separated Pools
- The test case cycle number when each fault was discovered
- The test case cycle number when each fault was released (mitigated)
- A potential system reliability growth number (accumulation of RPN values associated with each discovered fault) over time
- Estimated system reliability growth number (accumulation of RPN values associated with each fault discovered and mitigated over time)
- A large variety of detailed feedback sources for program debug and instantaneous relative results

Program Parameter and DOE Alternative Initialization

The specific search algorithm and subsequent case study designed in this research is focused on developing insight on the identified variables to apply to more realistic verification test plans. The primary method to draw summation results against the experimental hypothesis is through a five variable designed experiment.

Therefore, it is necessary to design a user interface in the tool that allows experiment switches in the code, where certain sections of the code are to be turned off or modified depending on the experiment. In addition, the GUI and variable declaration section of

the model allows the user to modify values of the parameters and the controlled variables.

Resource Pool 1 – Genetic Test Case Generator

The genetic test case generator serves as the heart of the search algorithm and the largest potential consumer of test resource units. A typical genetic algorithm will produce a population of potential solutions to a given optimization problem. A sample of the population is chosen and evaluated against the objective function in an effort to eventually evolve to the best solution. In a similar manner, the first resource consumption Pool (algorithm) of the adaptive genetic test case generator creates a given number of potential test cases (the number is determined by the user). The expense to execute the potential test cases is then calculated and assigned to each test case chromosome. The user has the choice of how large of a sample to take from the population to conduct the testing for each cycle. For example, if the population size is only one test case and it will be chosen no matter how expensive it is to run, this strategy will insure a very diverse set of tests. On the other hand, if a large population of potential test cases is created (along with the cost) and the user always chooses the (one) least expensive test case to run, there is a chance that many sub-system variables affected by a fault will not be tested. In the spirit of test case diversity, yet with the most cost effective path, the goal is to take advantage of both drivers. Creating a larger population of potential test cases insures a greater overall diversity of testing. By choosing a sample of each generated test case population, there is an opportunity to include several combinatory tests per cycle but at a more cost effective rate. In the algorithm, the program is designed to continuously cycle through the three resource consumption Pools until all resources are consumed or the algorithm is terminated by the user. Theoretically, a user can never be certain if all faults are discovered. In the case study, a glass box concept is used with the same set and location of faults to analyze the various aspects of the models.

Depending on the size of each population sample, each adaptive genetic search algorithm program cycle may consist of several internal cycles. For example, if the sample size used in the Pool1 algorithm is four, the algorithm will cycle through the test case queue, Pool 2 algorithm and Pool 3 algorithm, and then go back to the next test case in the queue four times before going back to the Pool 1 algorithm to create a new set of test cases to be placed in the queue.

Description of three primary data tables

There are several data tracking tables required for this system algorithm, but two tables are critical to the objective of maximized fault detection and reliability growth with fixed resources. In the first part of this research, one of the primary objectives of the Integrated Sustainable Product Development Framework was the introduction of risk management and resource optimization into the robust design and verification process. Therefore, a) the test case variable cost, b) assigned fault risks (RPN), and c) the genetic algorithm Gene probability are tracked in the model.

The cost table is static (i.e. metrics do not change during the course of the adaptive search) once the desired experimental set-up is chosen, and contains the relative value of the Gene in reference to the generation of a test case. Because resources and time are limited, the lowest cost test cases that still achieve the goal (in the aggregate form of time and material) would be preferred. While it is not possible to catch all faulty genes by only running the least costly test cases, it is still valuable to optimize the detection of the most valuable genes with minimal resources. For the case study used in the verification of the algorithm, the relative cost for each variable normalized between 0 and 10 are entered into the table (see Table 5.4). An engineer could use actual cost as well. In an effort to study the effects of cost on the adaptive search model, experiments will be conducted where each sub-system (gene) variable will be treated as the same cost, regardless of the actual cost. This action would help insure the most diverse test case suite but could exhaust the total resources before all faults are discovered. Because the standard cost in the case study runs from 1 resource unit to ten for each Gene

(which contain 10 variables), the average cost of each test variable is 5.5 resource units. This average cost is also reflected in Table 5.4. For example, the expense to execute the third variable in the first sub-system (gene) would be 3 resource units, if actual assigned costs were used, or 5.5 resource units, if average costs were used.

The second input that is critical to the adaptive feature of the search algorithm is the Gene probability table. In the spirit of designing a heuristic genetic algorithm, the probability of individual values within each Gene is modified during the course of the testing and fault discovery process. Table 5.5 contains the initial relative probability of a sub-system (gene) variable being chosen given a random number generator between 0 and 1 being used to pick the variable that represents the sub-system in the test case. As the algorithm starts to execute, all probability values are equal. There are two ways to change the probability values in the algorithm. The first way is having an independent fault detected and isolated. In this case, it does not make sense (or create new value) to continue testing a Gene variable design that will be modified to test the detected fault. Therefore, the Gene variable is blocked from being chosen in the test case generator. This block is accomplished by assigning a probability value of 0 to that specific Gene and then redistributing the range of numbers equally between the remaining variables in the sub-system.

The second modification method integrates the hypothesis that faults may be in clusters or hidden from view due to another fault. In reality, it can happen if the complexity of a particular design is higher than other modules or the relative experience of the module design team is less than others.

Table 5.4: Test case variable cost: actual vs. average

Gene Variable Expense ($ or test units)

Test Case Chromosome Set							
Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Gene 6	Gene 7	Gene 8
1/5.5	1/5.5	1/5.5	1/5.5	1/5.5	1/5.5	1/5.5	1/5.5
2/5.5	2/5.5	2/5.5	2/5.5	2/5.5	2/5.5	2/5.5	2/5.5
3/5.5	3/5.5	3/5.5	3/5.5	3/5.5	3/5.5	3/5.5	3/5.5
4/5.5	4/5.5	4/5.5	4/5.5	4/5.5	4/5.5	4/5.5	4/5.5
5 /5.5	5 /5.5	5 /5.5	5 /5.5	5 /5.5	5 /5.5	5 /5.5	5 /5.5
6/5.5	6/5.5	6/5.5	6/5.5	6/5.5	6/5.5	6/5.5	6/5.5
7/5.5	7/5.5	7/5.5	7/5.5	7/5.5	7/5.5	7/5.5	7/5.5
8/5.5	8/5.5	8/5.5	8/5.5	8/5.5	8/5.5	8/5.5	8/5.5
9/5.5	9/5.5	9/5.5	9/5.5	9/5.5	9/5.5	9/5.5	9/5.5
10/5.5	10/5.5	10/5.5	10/5.5	10/5.5	10/5.5	10/5.5	10/5.5

For the algorithm, the user has the ability to choose if they desire to increase the probability that a Gene variable will be chosen relative to the other variables on the sub-system during the random selection process. In this case, after a fault is detected and mitigated, the probability can be slightly increased on the Gene involved with the corrected fault, but it has to be done at the expense (and lowering the probability) of the other variables in the sub-system (i.e. chromosome gene). It is a very sensitive variable and cannot be so greedy that others faults, especially the expensive ones, are never chosen. This independent variable is analyzed in this research.

The third table contains the pre-assigned risk-RPN value and primary suspect gene-variable for each fault embedded in the complex system test case simulator.

Table 5.5: Example of normalized Gene probability

Gene Variable – Initial Probability (0-1 scale)

Test Case Chromosome Set							
Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Gene 6	Gene 7	Gene 8
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Because computer program is too large for this dissertation, the pseudo code and logic flow chart are presented for each major algorithm.

The pseudo code for the Resource Consumption Pool 1 algorithm is presented below and the flowchart is shown in Figure 5.16.

Pseudo Code of General (Resource Consumption Pool 1) Fault Search Algorithm

Start

-If new test resource units are available from bank, then receive

-If test resources are not available, then return to command center

-Else - conduct the following algorithm

-From 1 to X population size – generate a round of test case population candidates

- For I = 1 to 8

Generate random number, compare to probability table and assign Gene variable (reference Gene probability table)

-next

-Evaluate variable expense and assign total cost of each test case

Reference cost table

-Sort the test case population by cost (lowest to highest)

-For I =1 to x (x is the desired sample size from population)

Transfer the x - lowest cost test cases to the test case queue

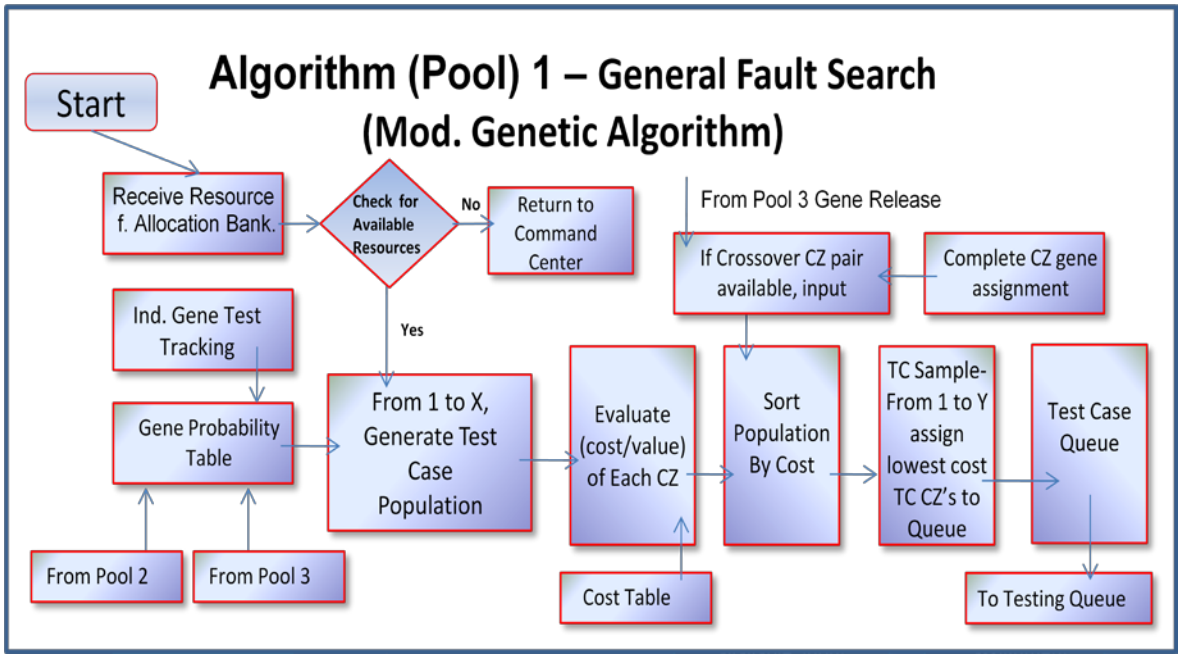


Figure 5.16: Logic for the general fault search algorithm

For illustrative purposes, assume the test engineer would like to create a population of eight (8) potential test cases in the complex system with 8 Sub-systems (genes). In addition, the engineer elected to take a sample of the four (4) lowest cost test cases for each cycle. In Figure 5.17, an example shows the generation of eight potential test cases. Each individual Gene in the chromosome is randomly picked based on the probability table. After the chromosome population is generated, the cost of each test

case candidate is calculated and then ranked from lowest to highest. The top four lowest cost test cases are chosen to be executed because they present the best utilization of resources. In this example, the lowest cost test case chromosome was #6 which would cost 12 resource units to execute; then test case number three (21 resource units); then test case number 1 (24 resource units); and then, finally, test case number 2, which would cost 25 resource units. These four test cases (the population sample for this algorithm cycle) would then be sent to the test case simulator queue.

Pool 1 Example

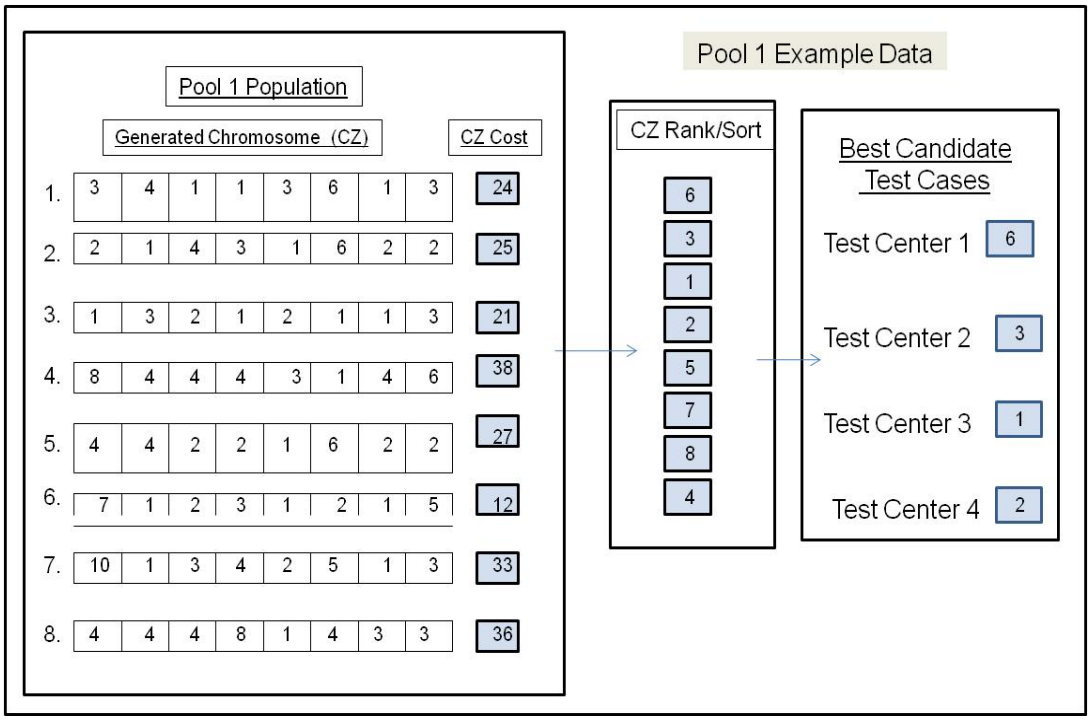


Figure 5.17: Diagram of data management in search Pool-1

In this example, test cases 5, 7, 8, and 4, were more expensive than the first four test cases and were not chosen to be used in the product verification test plan. There are an unlimited amount of possible configurations to choose from in consideration of how big the population should be and how many test cases are chosen each round. The focus of this research is to take advantage of the adaptive aspects of this approach and,

therefore, a smaller population and sample size is used per round to take advantage of the feedback. For the case study, these two values are held constant to focus attention on the five identified independent variables.

Test Case Queue

The test case queue is an algorithm and database designed to hold the set of test cases generated in the adaptive genetic search algorithm, queued up to be sent to the complex system test (and fault) simulator. The test cases are in the form of ordered number arrays and are also referred to as the test case chromosomes. The majority of the test cases are created by the Pool 1 algorithm, but some are also created by the Pool 3 process that takes any released test cases that previously detected a fault and creates new test cases (through crossover and mutation). The new test cases are placed directly in the front of the test case queue. The designed experiment used in the case study assessment has the ability to turn this feature on and off. Each time a test case is sent to the simulator, the appropriate amount of resources is deducted from the respective consumption Pool's bank account to simulate the expense of executing an actual test. The appropriate results are sent to the command center for data collection and analytics.

Resource Pool 2 - Faulty Gene Isolation Search Algorithm

The primary purpose of the Pool 2- resource consumption algorithm is to isolate the fault (Gene variables) within the test case that identified a system fault. It is important to recall that finding a fault is valuable, but value is not added until it is isolated and then mitigated.

The following pseudo code describes the Pool 2 isolation algorithm which uses a form of a tabu search to eliminate the various variable test case combinations in the process of isolating the actual fault (Figure 5.19 for logic flow). This search can be a critical process

if there is actually more than one fault that can be discovered in a particular test case. It can also create a phenomenon where the test fails for one particular reason, but the engineer isolates and corrects a different fault.

A fundamental requirement for system testing and an assumption for the assurance verification process is that the product can function (including at least one variable from each sub-system) at the most basic level. This base chromosome, referred to as the basic function test case (Test case = (1,1,1,1,1,1,1,1)), will be used in the tabu mutation Gene isolation algorithm within resource Pool 2. The base chromosome (all 1 values) will also be referred to as the primary "healthy" test case. Note: It is logical that, if the product does not function at the system level, further system testing is an inefficient use of resources.

The pseudo code for the Resource Pool 2 algorithm is presented below and the flowchart is shown in Figure 5.18.

Pseudo Code

-Receive test cases and supporting data that have identified a system fault

Update array counters and Pool 2 database

-Load test case into Pool 2 algorithm parameters

Look up the suspected Gene variable associated with the fault (in the associated database)

-Step 1: Search for any independent faults

-For i= 1 to the chromosome length (8)

-Freeze the value of the suspected "sick' Gene of the test case chromosome

-Replace the value of all remaining test case chromosome genes with the minimum function variable #1. (example (1,4,1,1,1,1,1,1)

-Send the new test case to the complex system test case simulator

If fault not detected – unlock suspected Gene variable and return to try next value for i in the isolation process

Else – if fault detected Goto FaultDetected Logic below

-If test cases 1 through i do not detect an isolated fault then go to step 2

-Step 2 : Search for any two variable – dependent faults

-For i= 1 to the chromosome length (8)

-Freeze the value of the suspected "sick Gene variable and the ith position of the test case chromosome

-Replace the value of all remaining test case chromosome genes with the minimum function variable #1. (example (3,4,1,1,1,1,1,1)

-Send the new test case to the complex system test case simulator

-if fault detected Goto FaultDetected Logic below else

-If fault not detected – return to try next value for i in the isolation process

– if fault detected Goto FaultDetected Logic below

-If no faults are detected on the possible two variable combinations with the suspect gene, then try other two variable test case combinations in order

If the test cases above do not detect any fault then go to step 3

-Step 3 : Search for any three variable – dependent faults

-For i= 1 to the chromosome length (8)

-Freeze the value of the suspected "sick Gene variable and the ith and jth position of the test case chromosome

-Replace the value of all remaining test case chromosome genes with the minimum function variable #1. (example (3,4,8,1,1,1,1,1)

-Send the new test case to the complex system test case simulator

-if fault detected Goto FaultDetected Logic below else

-If fault not detected – return to try next value for i in the isolation process

-repeat for the j'th position

– if fault detected Goto FaultDetected Logic below

-If no faults are detected on the possible three variable combinations with the suspect gene, then try other three variable test case combinations in order until fault is discovered or transfer test case to holding area and return to command

Goto: FaultDetected

If a fault has been detected and isolated then

-If the fault is an independent type, block the Gene variable and modify the probability table

Else modify the Gene probability table of the Gene variables identified in the fault isolation process

-Send isolated fault to the Pool3 problem tracking system (tabu list)

-End

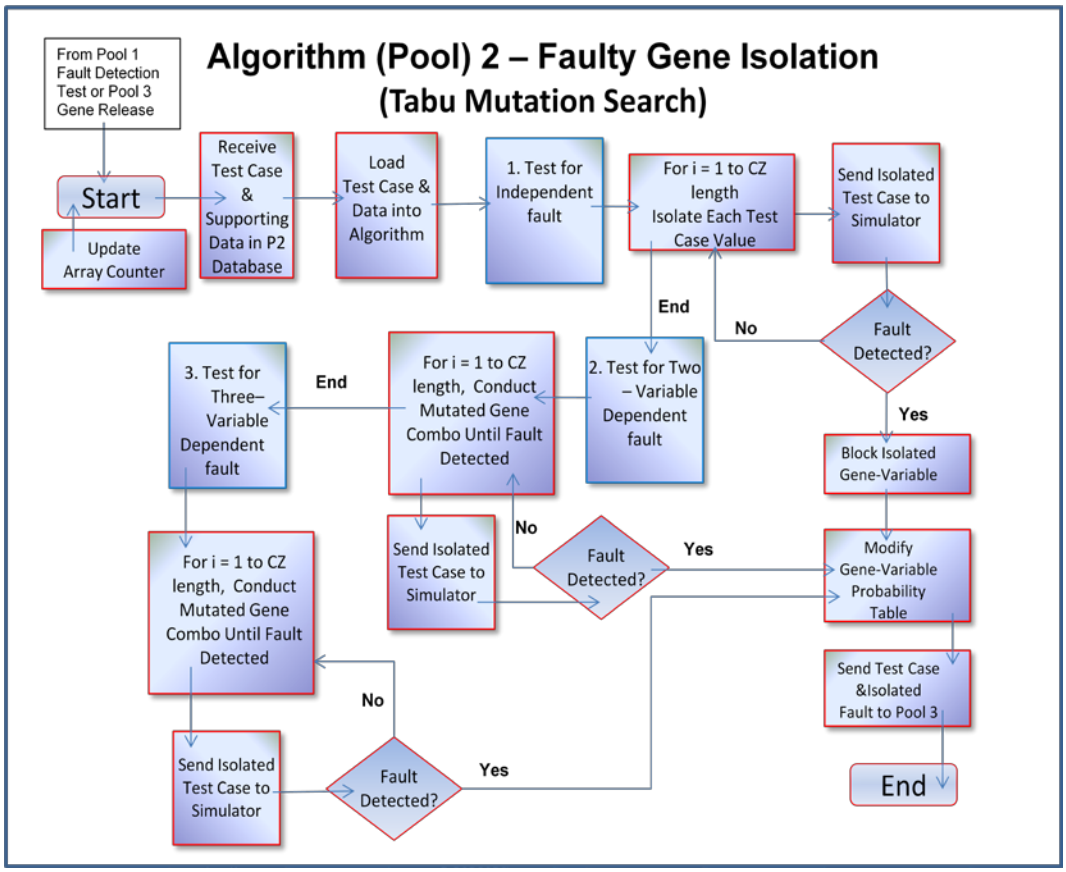


Figure 5.18: Logic for the Gene isolation algorithm

Example 1: Independent Fault

The first example presented is the isolation of an independent fault within a test case that indicated a fault. For example, this could be a function feature of the product that is defective regardless of the environment or other test case variables. Because there is no dependence with other test case variables, it is the simplest Gene to isolate.

Assume the general fault search in Pool 1 identifies a fault within the test case chromosome (CZ) and the following information is sent to the Pool-2 holding area - [(1,3,2,1,2,1,1,3), 132, 2] (Figure 5.19). The following information is parsed from the delivered information.

- Faulty Test Case CZ = (1, 3, 2, 1, 2, 1, 4, 5)
- RPN = 132

- Primary Suspect Gene = #2 (For the case study, this implies the second Gene out of eight in the chromosome is the primary suspect that caused the product failure).

After a test case with a system fault is received by Pool 2 algorithm, the first action in the isolation process is to determine if the fault is independent. The process is to isolate the suspected sick Gene with the known (healthy) base test case. All genes variables in the chromosome except the target Gene value are changed to the (base) value of "1" and this chromosome is tested in the simulator. If the response back is a faulty CZ, the suspected faulty Gene is confirmed and sent to the tabu holding list in Pool 3. If the result of the test is no fault detected, each of the other individual sub-system variables are isolated with the others values changed to the base value of 1 and sent to the simulator. If after all Gene variables are tested independently and no faults are detected, further isolation testing is required.

Example 2: Two Variable Dependent Fault

If an independent fault was not isolated in the first process, the next step is to search for any two variable combinations within the test case that triggered the fault. For example, this could be a functional feature of the product that is defective when used in a specific design of another module. Because there can be many combinations within the chromosome, two and three variable faulty Gene combinations can be resource intensive to isolate. In the below example, the fault detected in the simulator is caused by the combination values of the of second and sixth Gene in the test case.

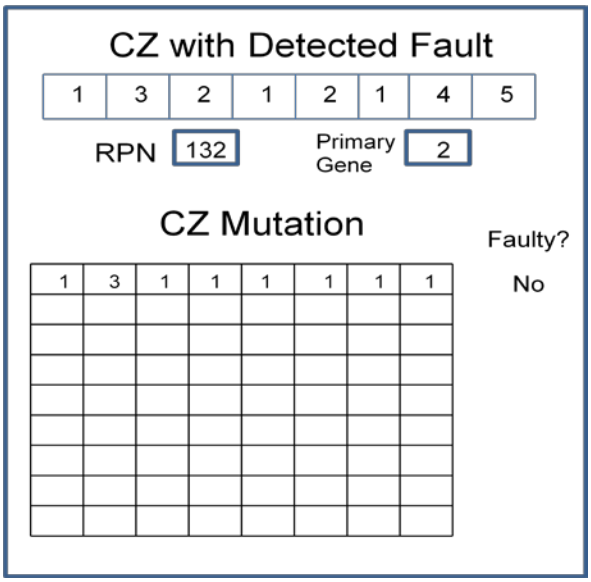


Figure 5.19: Diagram of Pool-2 mutation for independent fault

Assume the general search in Pool 1 identifies faulty chromosome (CZ) and the following information is sent to the Pool-2 holding area - [(6,3,2,4,2,3,4,5), 132, 2], (Figure 5.20). The following information is parsed from the delivered information.

- Faulty Test Case CZ = (1, 3, 2, 1, 2, 1, 1, 3)
- RPN = 132
- Primary Suspect Gene = #2 (For the case study, this implies the second Gene out of eight in the chromosome is the primary suspect that caused the product failure).

Assuming an independent fault was not detected in the first step of the isolation process, the algorithm takes five more test cycles to isolate the faulty Gene combination of Gene 2 and Gene 6. The test resources used for all five tests are recorded. If the second Gene in the combination was after the fifth or there was a three variable fault combination, isolated combinatory testing will continue until the fault is detected.

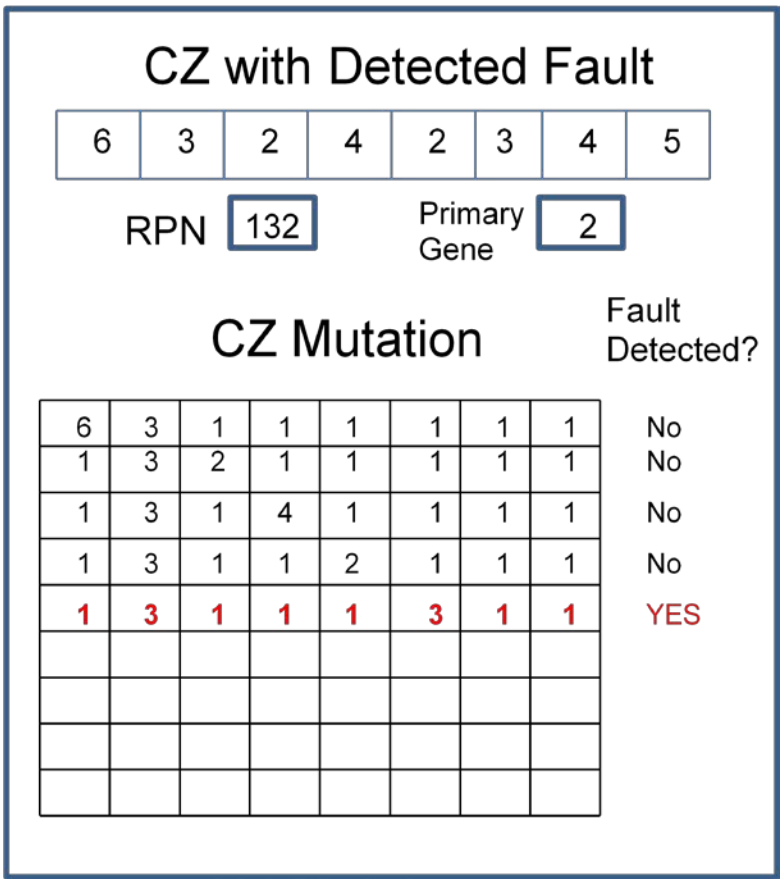


Figure 5.20: Diagram of Pool-2 test case mutation for 2 variable search

Resource Pool 3 – Fault Management and Test Regression

The Pool 3 algorithm is designed to carry out several tasks during the execution of the model. It includes a problem tracking and monitoring system, mitigation resource tracking, fault regression testing, and release and test case evolution via cut and crossover. The logic involved in this algorithm is described below.

Not only are resources required to discover and isolate faults in a complex system, they are also necessary to develop correction to the faulty design. As stated in the previous chapter regarding product assurance reality, taking the time lag between problem discovery and fault mitigation into account while executing the verification strategy is critical to accelerated reliability growth. In the case study, a theoretical complex system is simulated with various types of faults embedded in the system. In addition to the

type, location, and assigned RPN number, the fault also has an assigned amount of resources necessary to correct the problem. The Pool 3 algorithm receives and tracks the progress of faults that are isolated in the discovery process. The model also regulates the allocation of resources toward the faults and, when enough have been applied to a particular fault, it is released for regression testing.

One of the primary objectives of the experiment designed to analyze the effectiveness of the search algorithm is the hypothesis that prioritizing the order in which the discovered faults are corrected and released based on the highest RPN number first, will improve the overall effectiveness of the model. The user has the ability (via a switch) to command the algorithm to apply resources in the fault correction process to the faults in the order they were discovered or by the order of their risk rating level. In the second case, the fault with the highest RPN number always gets first priority in the allocation of available resources.

After a fault is released, the original test case is sent back to the complex system test case simulator for regression testing. If the test case comes back with a detected fault, it is sent back to the Pool 2 fault isolation process. If the test case comes back clean, the fault has been corrected and the proper data is sent to the command center. Genetic algorithm probability tables are updated and the test case is modified to create additional test cases in the assumption there may be other faults associated with the sub-system variables in the original test case. The model takes the original test case, cuts it in half, and re-populates the open Gene variables in the new test case chromosome with randomly generated values. The two new test cases are sent to the test case queue for analysis. Often problems can be clustered because the particular area under stress could be higher risk than average. The hypothesis is that the probability of detecting faults with half-parents of new chromosomes in the genetic algorithm is better than a random search. This is part of the adaptive genetic search process. The following pseudo code describes the Pool 3 algorithm and the logic flow chart is shown in Figure 5.21.

Pseudo Code for Regression Testing, Tabu Gene Release and CZ crossover (Pool3) Search Algorithm

-Receive test cases and supporting data that have identified a system fault

-Update array counters and Pool 3 database

-Sort Isolated Faults by RPN Value (from High to Low)

-If new test resource units available from bank, then receive

-If test resources are not available, then return to command center

-Else ; conduct the following algorithm

-Apply available resources to the highest priority faults in the problem tracking system

-Check if any faults have accumulated enough resources for release

-If no then Return to Command Center

-If yes, load original test case and send to complex system test case simulator

-If fault detected, then transfer test case and data to Pool 2 fault isolation algorithm

-Else (fault not detected) release fault from tabu list

-Update Gene variable probability tables and update dashboard metrics

-Cut original test case chromosome in half, re-populate parent test cases, and send to test case queue

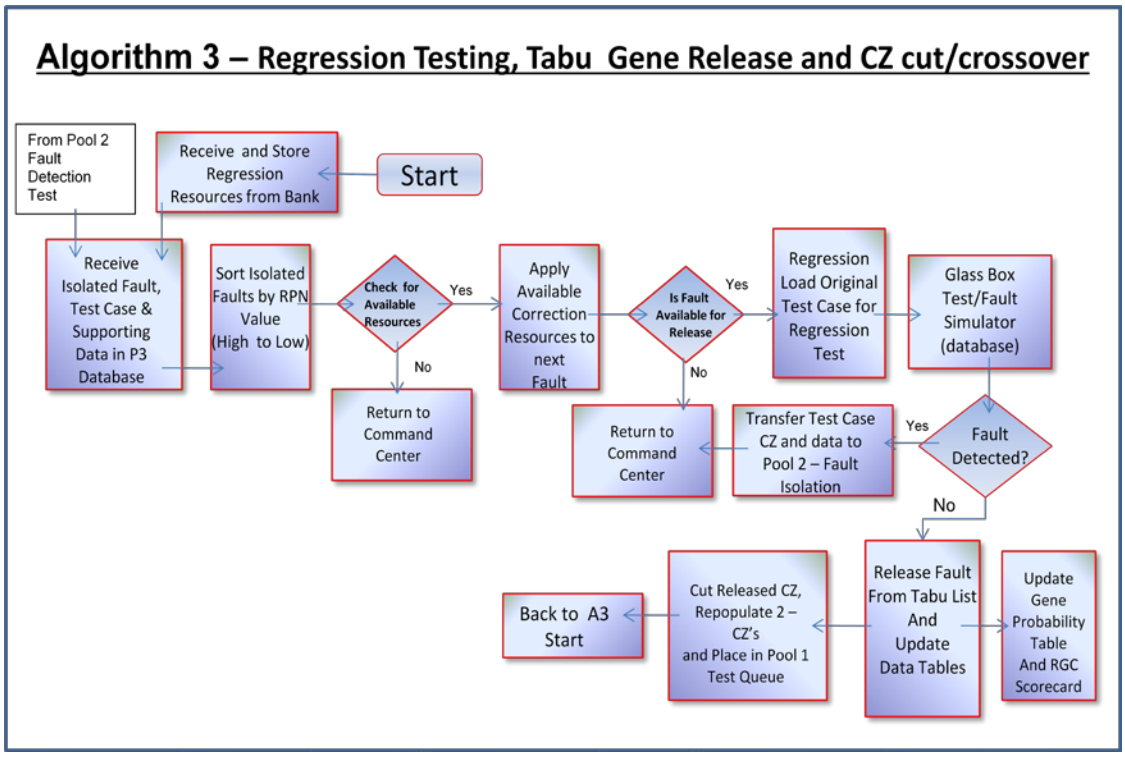


Figure 5.21: Logic for the regression testing and tabu Gene release algorithm

Complex System Test Case (Fault) Simulator

The complex system test case execution simulator is designed to imitate the experience a product engineer will face in the verification process when seeking out embedded faults. The fault simulator contains a "glass box" database containing the location of each embedded fault and related risk. These faults contain the identified test case variable ("sick gene") that is suspected to be involved with the problem but must be verified and assigned a RPN score. If a fault is embedded within a test case, the simulator indicates a defected test chromosome but not the particular or combination of genes. This realistic feedback is similar to the data a product assurance engineer would receive. Although a product system fault is detected, further testing is necessary in order to isolate the gene(s) within the test chromosome that caused the defect. The fault location database is considered a glass box where the location and assigned risk of the faults is defined ahead of time and held constant for all trials of the search

algorithm. This is done to assess the effectiveness of the algorithm designs relative to the five independent variables. The search algorithms themselves do not know the location of the faults.

In addition to the pre-assigned location of the independent and the two and three variable dependent faults, an internal clock based off of test unit resource consumption is used to release latent defects during the execution of the search algorithm. It is also important to remember that this simulator is recreating faults in the system in the same manner and sequence that the test engineer would see in the actual lab. For example, a particular Gene may be involved with 2 or more faults. As was described in the background section, independent faults must be detected and corrected before interactive (dependent) faults can be corrected. Therefore, only one problem at a time will be identified by the simulator.

5.6 Analysis of Model Effectiveness

5.6.1 Identification of Adaptive Genetic Search Algorithm Variables

The adaptive genetic search algorithm simulator is a complex computer program written to model the effects of a variety of influencing variables. The model is applied to a case study which contains 32 faults and whose locations are held constant in the simulator. The faults are a variety of single variable-independent; two-variable dependent, three-variable dependent, and latent defects that are released at a fixed time after the algorithm has started. The type, location, and assigned risk of the faults are based on real world experience. In addition to the independent variables which were described in Section 5.5.1, the dependent and controlled model variables are defined below:

5.6.2 Dependent Variables (Measured with Each Test Run)

- Cumulative Resource Consumption during system test runs
- Total Discovered System Risk (cumulative RPN) over Total (cumulative) Resource Consumption
- Resource consumption per fault discovery, including total consumption to find last fault
- Test case count per fault discovery and total test case count to discover all faults
- MTBF during the fault detection and mitigation process can be indirectly calculated

5.6.3 Controlled Variables (Values in the Model Held Constant)

- Number of potential test case candidates created per pass
- Number of test case candidates chosen to be used per pass
- Total amount of Test Resources available
- Initial amount of resources available for each test Pool (fault discovery, fault isolation and fault regression)
- Amount of test resources added to fault regression Pool over the system run (test case generation cycles)
- Total and type of faults injected into the fault simulator (case study)
- Timing of Latent Defect releases (based on resource consumption)

(Note: The controlled values can be modified in the model if desired)

5.6.4 Adaptive Genetic Search Algorithm - Simulation Hypotheses

In analyzing the effectiveness of the proposed adaptive genetic search algorithm, a five variable, two level designed experiment is used to present the statistical significance of five identified independent model variables. Therefore, the following null hypotheses will be investigated:

H01: Treating the cost of all potential test case Sub-systems variables (gene-variables) as equal, ensures the best chance for maximum fault detection and reliability growth, given resource constraints.

H02: Prioritize the order of fault correction: Driving resources to the correction, regression, and release of the detected fault with the highest assigned risk value rank first (based on customer satisfaction), will ensure the maximum fault detection and reliability growth, given resource constraints.

H03. Creating offspring "child" test cases through crossover and mutation of parent test cases that previously discovered a fault ensures maximum fault detection and reliability growth RGC, given resource constraints.

H04. Testing all independent sub-system test variables before the adaptive genetic search algorithm is enabled ensures the best chance for maximum fault detection and reliability growth, given resource constraints.

H05. Focusing on sub-system history: By modifying (increasing) the probability of a sub-system variable to be chosen within a test case sub-system, based on previous success, will ensure maximum fault detection and reliability growth, given resource constraints**.**

5.6.5 Expected Shape of the Reliability Growth Curve

One of the desired effects of the search algorithm is to create a tool for the engineering community that increases the understanding of the dynamics involved in the reliability growth curve for a complex system. This research identified seven types of fault or fault states that can affect the progress of the reliability growth curve. In reality, the detection and mitigation of faults during the development life-cycle may not be as quick or efficient as expected and cause delays in the product delivery. Traditionally, the reliability growth curve is presented as a continuous curve, plotting the mean time between failures over time. In this research, the reliability growth in the form of discovered and mitigated risk will be measured and plotted for each factor of the designed experiment. The chart in Figure 5.22 identifies the expected shape of the reliability growth curve if the search and mitigation process is the most efficient. The rate of discovered faults per consumed resources can also be plotted.

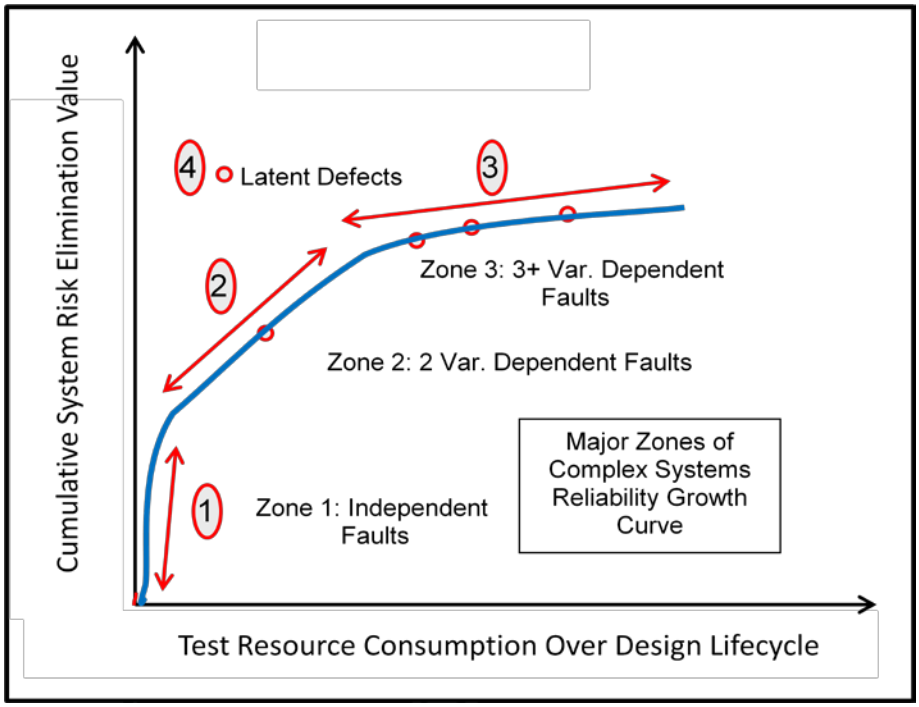


Figure 5.22: Expected Reliability Growth Curve Shape

5.7 Summary

Chapter five presented the design of the adaptive genetic search algorithm, along with the logic behind the fault detection process. Emphasis was placed on the reality that there is a time lag between fault detection, and fault mitigation, which can create unexpected product development delays and fault escapes to the field. The goal of the search algorithm is to increase the efficiency of the fault search and mitigation process. Chapter six is focused on designing a validation experiment in the form of a case study and a complex system simulator. The simulator will be used to exercise the adaptive genetic search algorithm with a designed experiment that will be used to analyze the effectiveness of the model and hypotheses.

Chapter 6: Case Study: Model Execution, Data Collection and Data Analysis

This chapter presents two case studies that illustrate models designed to aid the engineering team in sustainable product development. The first study compares generation to generation product designs relative to the sustainable products' value proposition metrics. The second study applies the adaptive genetic search algorithm to simulated complex systems with embedded faults. An experiment is conducted to evaluate the impact that five independent variables have on the model's effectiveness.

6.1 Sustainable Products Value Proposition - Case Study

In the pursuit of producing sustainable products, there is not a simple prescription for designing products that perfectly meet the needs of producers, consumers, and the socio-environment in a single package. A producer may make a tradeoff for one of the sustainable value proposition driving aspects in order to improve in several other metrics. It is still important to look at the aggregate score of a particular next generation design as compared to the previous generation as well as the best of breed offering in the market for each category. Figure 6.1 presents the collected driving aspects of the value proposition that the engineering community is encouraged to measure new potential new product design concepts relative to previous designs and vs. the best of breed in the industry.

In a competitive market with worldwide competition, the value of any one particular sustainable value proposition metric is relative to the competitive offerings and societal impacts. For the value proposition comparison tool, a scale of 1 to 10 is used to rate each driving aspect as compared to the product in the field that is the best of breed for that particular value. In order to promote continuous improvement, the best of breed is

given a set value of 8 across the board. Therefore, when considering potential designs for next generation product offerings, surpassing the current best of breed value proposition would be rated a relative score of 9 or 10. The hypothetical best score of the best of breed product is 144 points.

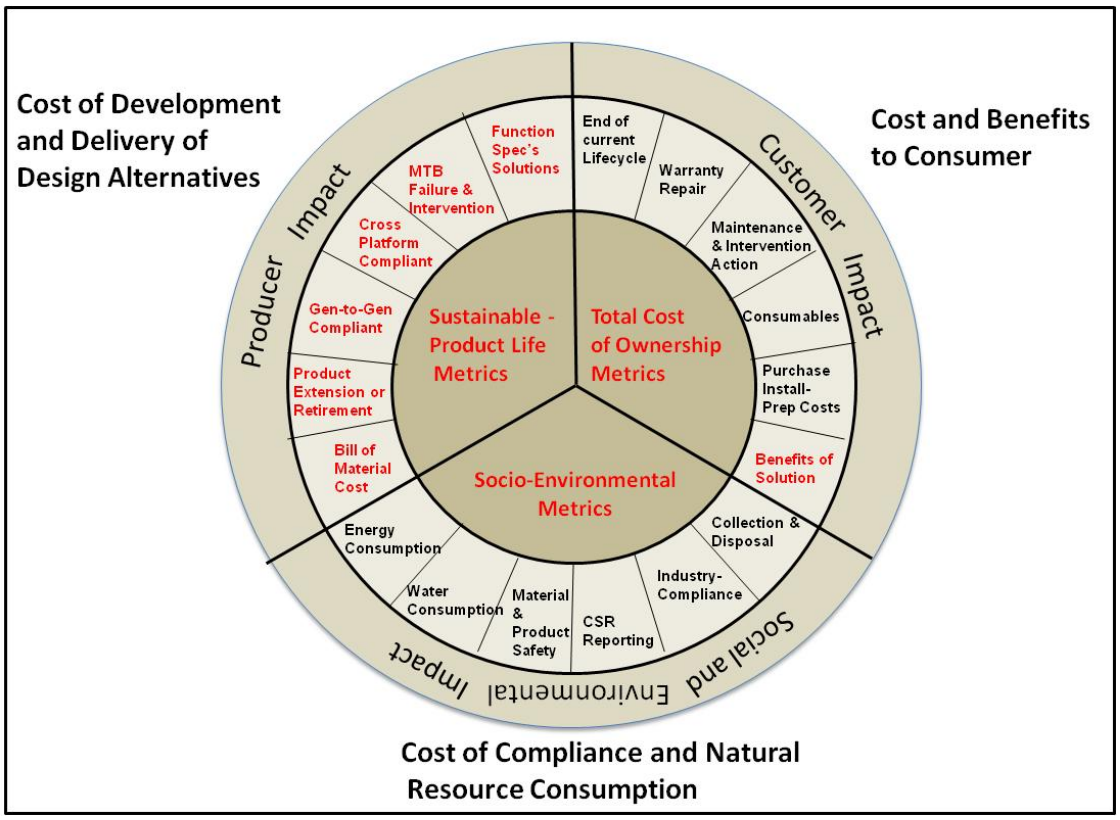


Figure 6.1: The complete set of sustainable value proposition driving aspects

A case study is presented comparing the relative rating of a potential next generation product design (in each of the sustainable value proposition driving aspects) to the current producer's product offering and the best of breed. The results of this analysis are presented in Table 6.1 and graphically presented in Figure 6.2. In the graph, the light blue color represents a relative scale of 7-8 points where 8 is the value of the current best of breed offering to a consumer at that time. The green segment on the graph represents a next generation design that exceeds the current value proposition to the consumer.

Table 6.1: Comparison of case study results

Sustainable Product Design Metrics: Driving Aspects			
Producer Impact: Cost of Product Life Development Metrics	**Current Industry Category Best of Breed**	**Generation 1 Design**	**Generation 2 Design**
Bill of Material Expense	8	7	8
Relative Design concepts	8	8	10
MTBF and MTBI	8	5	8
Cross Platform Compliance	8	2	6
Generation-to-Generation Compliance	8	6	6
Product Life Extension or Retirement	8	1	8
Total Producer Impact Score	48	29	46
Consumer Impact: Total Cost of Ownership Metrics			
Benefit of New Innovation	8	5	8
Cost to Purchase, Install	8	8	10
Cost of Consumables	8	7	9
Cost of Maintenance	8	2	8
Cost of Warranty Repairs	8	6	8
Cost of the End of Current Life Cycle	8	3	6
Total Consumer Impact Score	48	31	49
Socio-Environmental Impact:			
Total Energy Consumption	8	6	9
Total water consumption	8	7	8
Product and Material Safety Compliances	8	8	8
CSR and Environmental Activities	8	6	9
Industry specific certifications	8	7	8
Collection and Product Disposal	8	5	8
Total Socio-Env. Impact Score	48	39	50
Overall Product Design Score	144	99	145

In this particular case study, there is a 46% improvement in the next generation design and a rating of 15 of the 18 driving aspects considered equal to or better than the current best of breed. With the next generation design, the table shows that three of the driving aspects did not meet the current best of breed offering. These results point out potential opportunities but indicate that an improvement in those three categories may come with a need to lower a rating in another category.

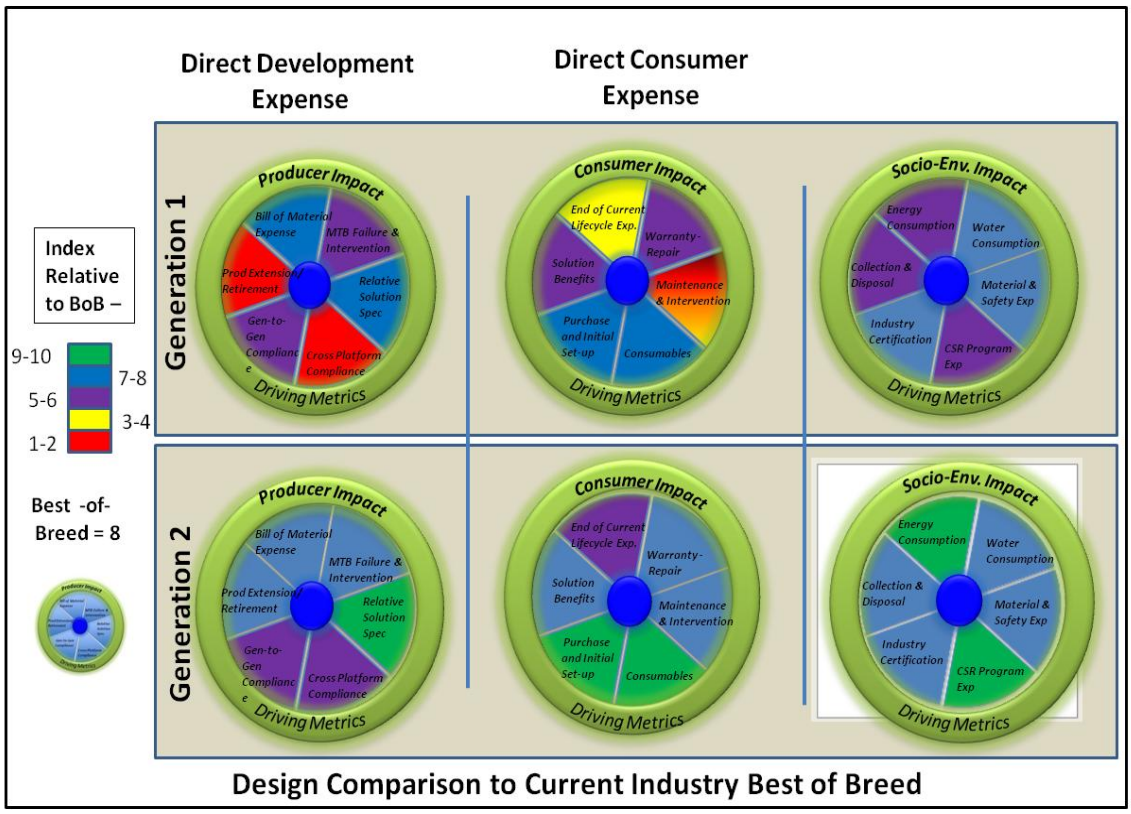


Figure 6.2: Graphical presentation of relative sustainable value proposition case study results

6.2 Adaptive Genetic Search Algorithm – Case Study

The premise of the research described in the last chapter centered on the value that timely feedback contributes to the development process. The opportunity for improvement in the development of robust designs that meet the sustainable value proposition can be seen graphically in the reliability growth curve of a product during the development life-cycle. In complex systems, early detection of embedded faults increases the likelihood of meeting target risk levels at the start of production and improves the utilization of verification resources. Both of these aspects contribute directly to increased value outlined in the sustainable products Half-Life Return Model. During the development process, one of the primary drawbacks of the analysis of reliability growth curves is that they are only based off of discovered faults. In reality, an embedded fault may exist in a system design that has not been detected and, therefore,

the engineering team does not even know of the potential risk. The adage, “you don’t know what you don’t know,” is relevant in the context that research is necessary in the study of verification strategy and test case development to increase ability to discover complex system faults that may escape the development process with traditional verification efforts.

In the area of complex system verification, this study contributes to the field of test case development strategies by focusing on the effects that five targeted variables have on fault detection and risk elimination. This chapter focuses on evaluating the adaptive genetic search algorithm by conducting a five variable, 2 level, full factorial designed experiment. In order to improve the accuracy of the results, each of the test run (treatment) combinations is repeated eight times and the average of the dependent variables is used in the analysis package. Graphical and statistical analysis is conducted to draw conclusions regarding hypotheses and general search effectiveness. The statistical package JMP by SAS is used to provide mathematical confidence in the results.

6.2.1 Experimental Set-up

A computer program was written to convert the adaptive genetic search tool into a working model to analyze its effectiveness. The program was written in Microsoft Visual Studio with an interface designed to capture the results of each run and transfer into a Microsoft Excel database. In addition, a complex system simulator was created to include all seven fault types embedded in the system.

6.2.2 Independent Variables

The independent variables, which are modified with each treatment and placed in ordered arrays, defined by the full factorial designed experiment (detailed in Table 6.2) and a description of values assigned is provided below.

1. Cost of Each Test Case Gene Variables (i.e. Cost of Test Case Chromosome)

The high signal (signified by 1) for this independent variable ("cost") assumes the actual cost of each Gene variable in the chromosome will be used. As defined by the experiment, the values range from 1 test resource unit for the first variable in each sub-system and incrementally climb to 10 resource units for the tenth sub-system variable. The low signal (signified by 0) for the first independent variable assumes the average cost of the Gene variables will be used. For this case study, the average cost of the sub-system variables is 5.5 test resource units. The short name for this variable is "cost."

2. RPN – Risk value of each fault detected (Prioritizes fault mitigation resources)

The high signal (signified by 1) for this independent variable (RPN utilization) assumes the algorithm will utilize the assigned relative risk prioritization number (RPN) for each discovered fault to prioritize the allocation of resources in Pool 3. The low signal (signified by 0) for the second independent variable assumes all discovered faults have equal risk level ratings and, therefore, resources are applied to the correction (time release) of the faults in the order they were discovered. The short name for this variable is "RPN". Traditionally, a risk prioritization number is based on the multiplication of three factors (severity, occurrence and detection), each on a scale of 1 to 10, with 10 signifying the highest risk for each category. Therefore, the worst case scenario for a problem (which is highly idealistic) would be 10 x 10 x 10 = 1000 (Cohen et al., 2013). A different scale was used in the case study in order to match the relative scale of the case study in the computer model. The assigned RPN values in the case study range from 750 to 3600, which is higher than the traditional number set. The significance of the assigned values is only relative to the embedded faults in the system.

Table 6.2: Case study full factorial designed experiment

Algorithm - Independent Variable Analysis - Experiment Design						
Treatment No.	Pattern	Cost	PRN	Pool 3B	1D Sweep	Probability
1	00000	Low	Low	Low	Low	Low
2	10000	High	Low	Low	Low	Low
3	01000	Low	High	Low	Low	Low
4	11000	High	High	Low	Low	Low
5	00100	Low	Low	High	Low	Low
6	10100	High	Low	High	Low	Low
7	01100	Low	High	High	Low	Low
8	11100	High	High	High	Low	Low
9	00010	Low	Low	Low	High	Low
10	10010	High	Low	Low	High	Low
11	01010	Low	High	Low	High	Low
12	11010	High	High	Low	High	Low
13	00110	Low	Low	High	High	Low
14	10110	High	Low	High	High	Low
15	01110	Low	High	High	High	Low
16	11110	High	High	High	High	Low
17	00001	Low	Low	Low	Low	High
18	10001	High	Low	Low	Low	High
19	01001	Low	High	Low	Low	High
20	11001	High	High	Low	Low	High
21	00101	Low	Low	High	Low	High
22	10101	High	Low	High	Low	High
23	01101	Low	High	High	Low	High
24	11101	High	High	High	Low	High
25	00011	Low	Low	Low	High	High
26	10011	High	Low	Low	High	High
27	01011	Low	High	Low	High	High
28	11011	High	High	Low	High	High
29	00111	Low	Low	High	High	High
30	10111	High	Low	High	High	High
31	01111	Low	High	High	High	High
32	11111	High	High	High	High	High

3. Cut and crossover of test cases (CZ's) from high probability parents

The high signal (signified by 1) for this independent variable (genetic algorithm crossover) assumes the algorithm will utilize the genetic algorithm process of cutting a

released test case that previously discovered a fault and forming two new test cases. The low signal (signified by 0) for the third independent variable assumes this process will not be utilized and all search based test cases will be based on the probability tables for each chromosome gene. The short name for this variable is “Pool 3”.

4. Sweep of all Independent variables first in test case chromosome creation

The high signal (signified by 1) for this independent variable (initial independent variable test sweep) assumes the algorithm will create test cases that sweep through all independent sub-system variables before the process is converted to the genetic algorithm test case generation method. The low signal (signified by 0) for the fourth independent variable assumes this process will not be utilized and genetic algorithm probability tables will be used to choose the test case sub-system variable with the first test case. The short name for this variable is “1D sweep”.

5. Gene probability modification given fault detection

The high signal (signified by 1) for this independent variable (Gene variable probability modification) assumes the algorithm will utilize the genetic algorithm process of modifying the probability of a particular sub-system (gene) variable. The variable will be chosen based on its involvement in previously successful test cases. The low signal (signified by 0) for the fifth independent variable assumes this process will not be utilized and the use of genetic algorithm probability tables will be held constant during the test case population creation process. The short name for this variable is “Probability.” The target incremental increase of probability for the target variable is 5 percent divided by the amount of variables involved in the mitigated fault. For example, after a two variable fault is successfully detected and mitigated, the probability of each of the sub-system (gene) variables being selected will increase by 2.5% and, therefore, decrease the probability of the remaining sub-system variable by 0.277% (which is 2.5%/9).

6.2.3 Controlled Variables

The controlled variables in the analysis of the adaptive genetic search algorithm are held constant to focus on the effects of the five identified independent variables. The values chosen for the controlled variable could be changed in order to conduct further research on the model's efficiency. The controlled variables are as follows:

Controlled Variables (Values in the Model Held Constant):

1. Test case population: number of chromosomes (potential test case candidates) created per Pool-1 pass (case study target = 5 to 8)
2. Test Case Sample: number of chromosomes (test cases) chosen per Pool-1 pass (case study target = 4)
3. Total amount of test resources units available (case study target 175,000 test resource units)
4. Initial amount of resources provided to each test Pool (case study target = 100,000 resource units in Pool 3)
5. Amount of fault mitigation resources added to Pool 3 over the system run (test case generation cycles) (case study target = 1 to 1 match to resources applied to Pool1 1 and 2)
6. Total and type of faults injected into the fault simulator (see case study for details; target = 32 faults with 4 latent)
7. Timing of Latent Defect releases (various - based on resource consumption; see case study for details)

Note: For controlled variables number four and five, the values chosen in the execution of the model were set relatively high enough to minimize the sensitivity of this model aspect compared to the independent variables.

6.2.4 Data Collection for Dependent Variables and Analysis

The data for dependent variables gathered to analyze the fault detection and the mitigation process. The dashboard designed in the computer program is shown in Figure 6.3. The data is presented in the model scorecard and transferred to a database. In addition to several data sources for model feedback, four primary data streams are collected for further analysis. These data streams include the following:

1. Cumulative resource consumption during system test runs
2. Total system risk mitigation(cumulative mitigated RPN growth) divided by total (cumulative) resource consumption
3. Cumulative resource consumption per fault discovery, including total consumption to find last fault
4. Test case count per fault discovery and total test case count to discover all faults

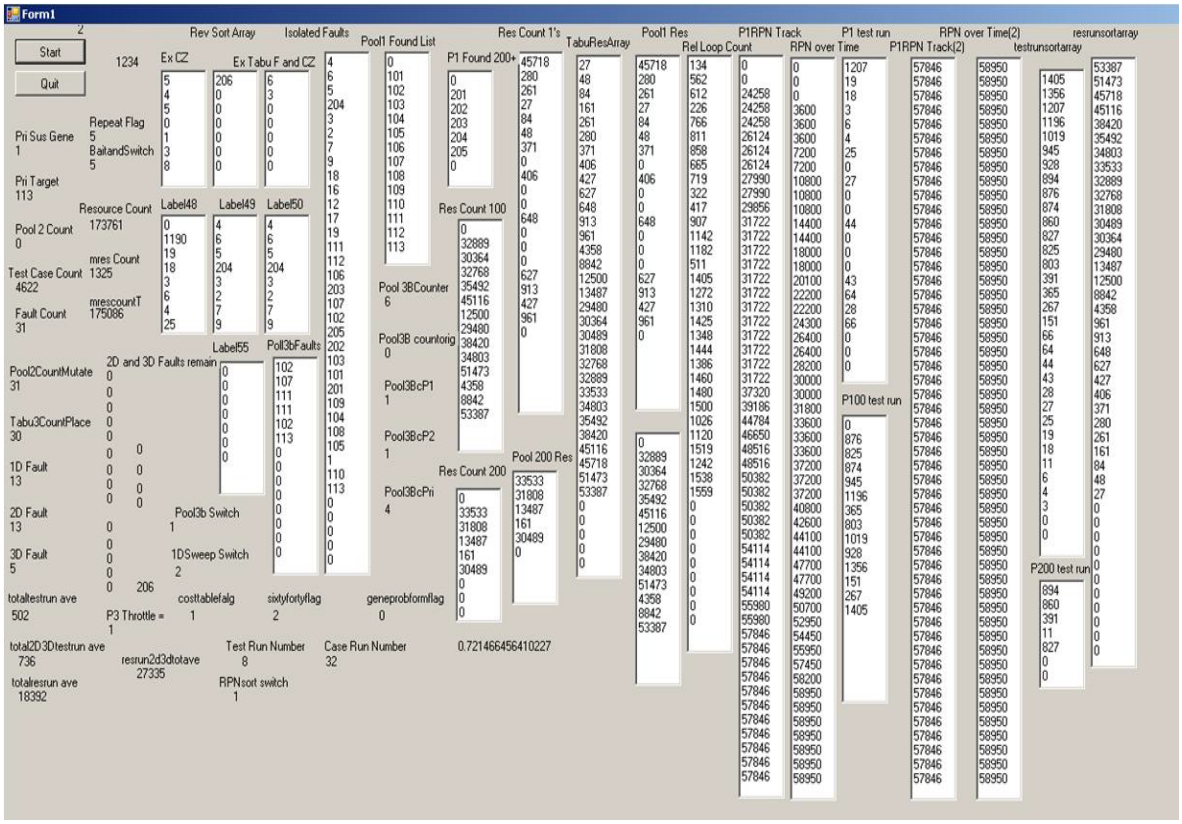


Figure 6.3: Screenshot of search algorithm tool – dashboard

Data Analysis

After the eight replications of each treatment are executed and data collected, continuous data rate samples and the average of the two data sets will be analyzed.

Continuous Data (Sampling) Plots

There are two primary continuous data sets that are sampled for each treatment. During the execution of the search algorithms, test resources are applied to the fault detection and mitigation process until they are fully consumed (the target total resource bank is 175,000 units). During the consumption process, incremental data samples are

taken every 1400 test units for the growth of system risk mitigated via cumulative RPN and the cumulative amount of mitigated faults. After eight runs of each treatment, the average value for each data point is calculated.

Fault Detection Sample Plots

For the case study, thirty-two faults are embedded in the system. Therefore, a plot of the faults discovered and mitigated over the consumption of the test resources will become asymptotic to a horizontal line on the y axis (Figure 6.3).

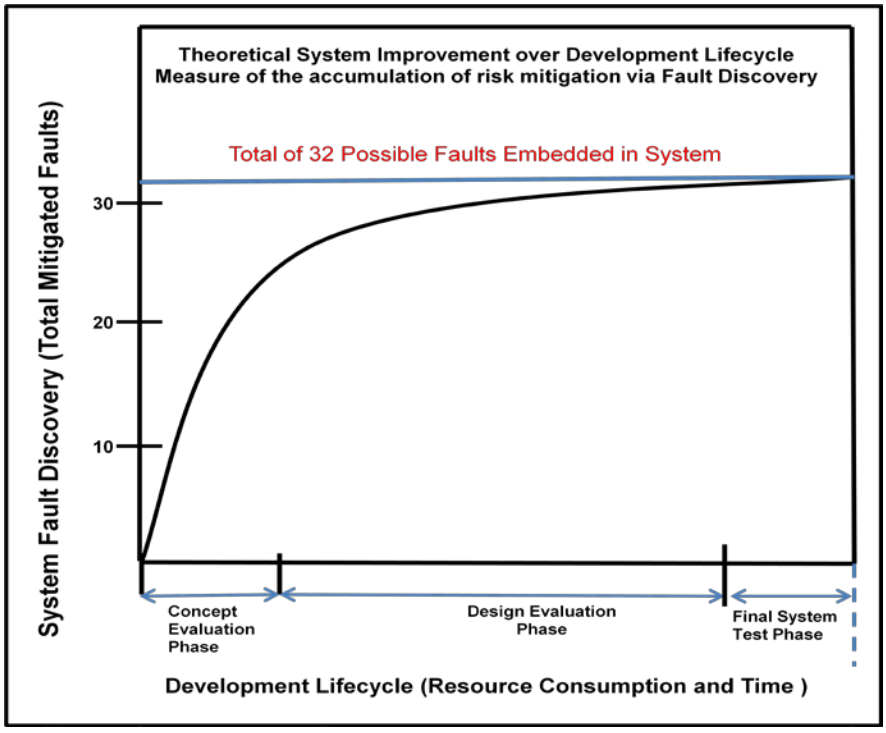


Figure 6.4: Ideal reliability growth based on detected fault count

In an ideal state, there is rapid acceleration of faults discovered and then the full amount of possible faults are discovered (the y intercept is 32). If the search algorithm is too greedy or inefficient, the resources will be consumed before all of the faults are detected (a lower y-intercept value).

Fault Detection Sample Plots

The second data sample is the cumulated mitigated risk over the course of the test resource consumption. This y-value is the accumulation of the RPN value assigned to each fault that was detected and mitigated. It serves as a proxy for system reliability growth during the product assurance process in the development life-cycle (Figure 6.4). For the case study, the average RPN value was 1865.6. With a total of 32 faults, the y-intercepts of the RPN system risk mitigation line is equal to 59,700.

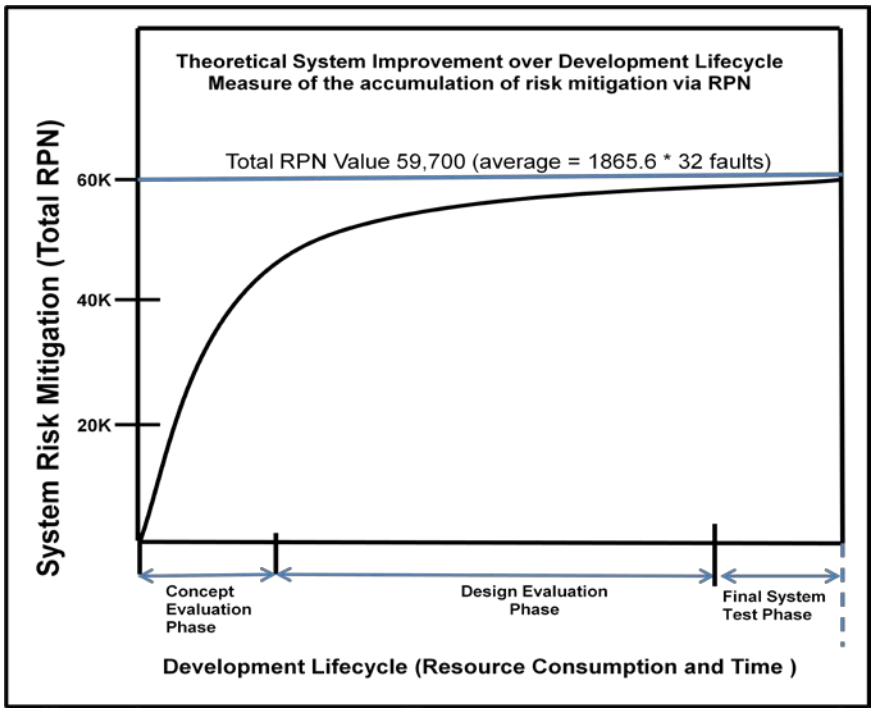


Figure 6.5: Ideal reliability growth based on mitigated risk

Discrete Data Analysis – Statistical DOE

In addition to data sampling during the search algorithm process, statistical analysis through a DOE was conducted for two targeted dependent variables. In the analysis of the search algorithm, there is a priority of objectives that should be considered in choosing the analysis metrics. The first priority of the search algorithm is to find as many embedded faults in the complex system as possible with a fixed amount of resources.

Undetected faults that escape to customers can have the greatest negative impact to the Half-Life Return Models metrics. In the analysis of the designed experiment, a penalty is applied to each treatment for each fault undetected during the test run.

The second priority of the search algorithm is the acceleration of the reliability growth curve given a fixed amount of resources. The third priority is the amount of test cases required to find the embedded faults. If the same amount of faults can be discovered for the same amount of test resources, but in half the test cases, the second path would be preferred. With that in mind, the reason for it being only the third priority is that the algorithm adaption may become too greedy during the aggressive search process (test case reduction) and completely miss embedded faults. Given the first priority is total fault detection and mitigation, the two dependent values that will be statically analyzed are described below.

1. Average Test Case Count: The average number of test cases required for the discovery of the embedded system faults. If a fault was left undetected, a 20% penalty is added onto the total amount of test cases executed before the resource bank was depleted. Even with the aid of advance algorithms, the search for faults in a complex system is still an exploratory process and, therefore, the actual number of test cases to find all the faults in the test case is an estimate. The 20% penalty for undetected fault serves as an approximation of the additional resources required and serves as a proxy that is accurate enough for the desired analysis sensitivity. In reality, it is possible that some treatments may become so greedy that the search process would become so constrained it would never find the remaining fault(s) regardless of the amount of test resources provided. An additional ten percent penalty is added onto the accumulation of test cases for each additional fault left undetected.

2. Average Total Resource Consumption: The second analysis is based on the average total number of test resources consumed at the time of the last fault

discovery. If a fault was left undetected, a 20% penalty is added onto the total amount of resource consumed before the resource bank was depleted. An additional ten percent penalty is added onto the accumulation of test resources consumed for each additional fault left undetected.

6.2.5 Complex System Simulator – Embedded Fault Locations

The complex system test case and fault simulator utilized in this case study is embedded with 32 faults distributed between the four primary fault types.

1. There are 13 independent, single variable faults
2. There are 13 two variable, dependent faults
3. There are 6 three variable, dependent faults
4. Of the 32 faults outlined above, four are latent and released in the simulator after a period of time

The location and supporting data for the case study faults are held constant for each test run to analyze the effectiveness of the various search models. The detailed for each fault location are described in Table 6.3. For example, the location of fault number 1 in the simulator is G1-2. This indicates the fault is located in the second variable of the first chromosome gene. The locations of the faults are not released to the search algorithms. The assigned locations of the various faults are primarily based on two factors. The first is that the locations are assigned in various locations to be certain that the algorithms are faced with a difficult search problem. In addition, many of the faults are clustered within a given sub-system or even targeted on a specific sub-system variable in order to imitate realistic design conditions. For example, a concentration of faults may be due to a relatively less experienced team assigned to the particular sub-system. Another reason could be due to a very challenging design requirement for a given sub-system variable.

Table 6.3: Complex system embedded fault location data

Complex System Simulator - Embedded Fault Data				
Fault Number	Fault ID No.	Fault Type	Assigned RPN	Fault Location(s)
1	1	1D - Latent	2100	G1-2
2	2	1D - Independent	1800	G3-2
3	3	1D - Independent	1800	G3-2
4		1D - Independent	3600	G1-3
5	5	1D - Independent	2100	G1-6
6	6	1D - Independent	2100	G1-4
7	7	1D - Independent	1800	G3-9
8	9	1D - Independent	3600	G4-3
9	12	1D - Independent	2100	G6-4
10	16	1D - Independent	2100	G6-3
11	17	1D - Independent	1200	G8-17
12	18	1D - Independent	3600	G4-4
13	19	1D - Independent	2100	G8-10
14	101	2D - Dependent	750	G1-2, G4-3
15	102	2D - Dependent	750	G1-3, G4-3
16	103	2D - Dependent	1500	G1-6, G3-9
17	104	2D - Dependent	3600	G1-5, G8-8
18	105	2D - Dependent	750	G1-6, G8-8
19	106	2D - Dependent	750	G5-3, G6-3
20	107	2D - Dependent	1500	G3-4, G4-4
21	108	2D - Dep Latent	3600	G3-2, G7-9
22	109	2D - Dep Latent	750	G3-3, G4-9
23	110	2D - Dependent	750	G4-6, G5-6
24	111	2D - Dependent	1500	G6-5, G7-5
25	112	2D - Dependent	3600	G7-2, G8-2
26	113	2D - Dep Latent	750	G2-5, G8-2
27	201	3D - Dependent	750	G1-2, G4-7, G7-1
28	202	3D - Dependent	750	G1-9, G6-4, G8-3
29	203	3D - Dependent	1500	G6-5, G7-2, G8-1
30	204	3D - Dependent	3600	G2-3, G3-1, G4-1
31	205	3D - Dependent	750	G3-5, G4-4, G5-5
32	206	3D - Dependent	750	G5-2, G6-8, G8-5

A graphical presentation of the fault location is presented below. Figure 6.5 and Table 6.4 detail the independent variable. The first chart is the schematic location and the second chart converts the fault location to a two dimensional array. Two variable – dependent faults are interactive faults involving variables from two different sub-system

genes. The schematic and array locations are presented in Figure 6.6 and Table 6.5 respectively. Three variable dependent faults data is presented in Figures 6.7 and Table 6.6. Latent defects are represented by the yellow graphics.

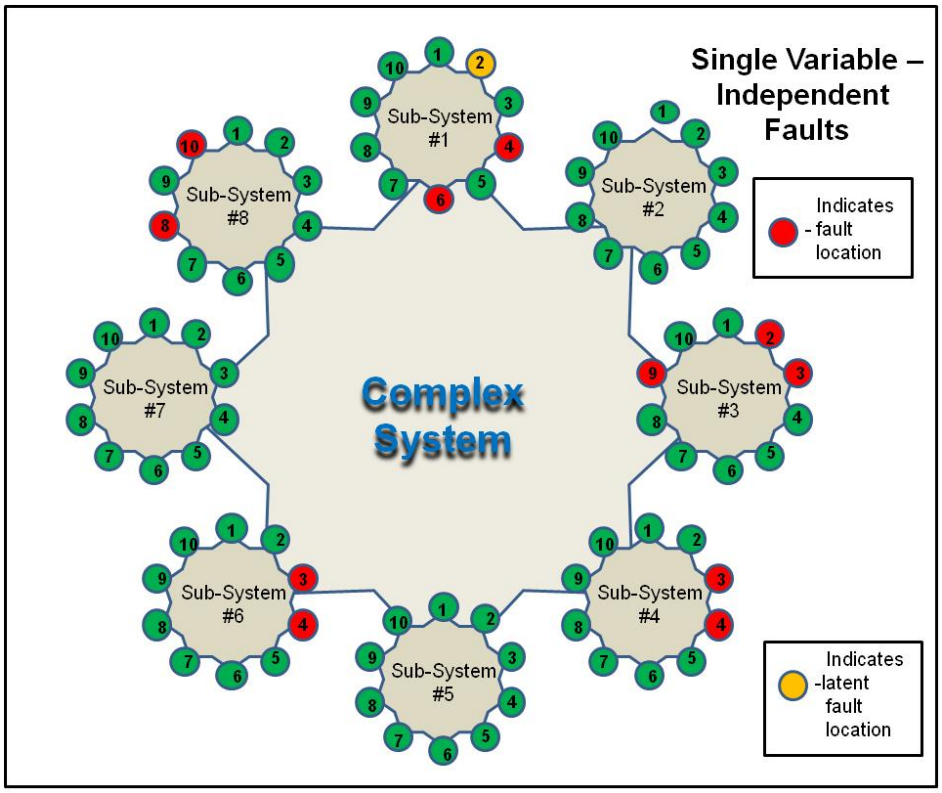


Figure 6.6: Graphical location of single variable, independent faults

Table 6.4: Array location of single variable, independent faults

Complex System Model - Independent Fault Locations							
Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Gene 6	Gene 7	Gene 8
1	1	1	1	1	1	1	1
2 – 1 (Latent)	2	2 - 3	2	2	2	2	2
3	3	3 - 2	3 - 9	3	3- 16	3	3
4 - 6	4	4	4- 18	4	4- 12	4	4
5	5	5	5	5	5	5	5
6 – 5	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8- 17
9	9	9 – 7	9	9	9	9	9
10	10	10	10	10	10	10	10 - 19

Gene Variable

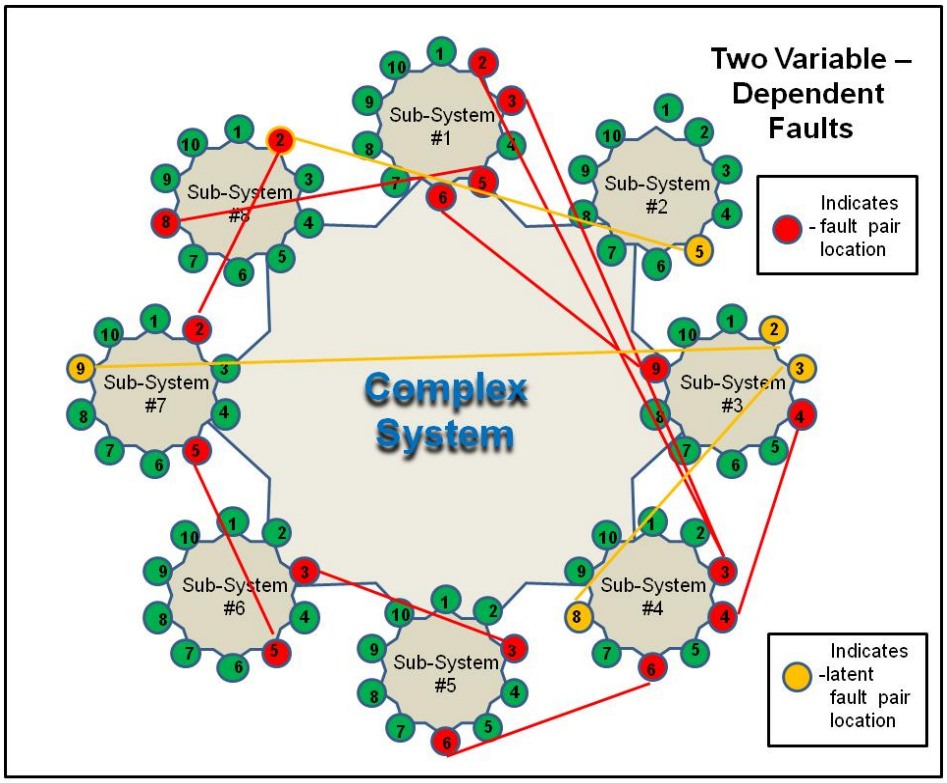


Figure 6.7: Graphical location of two variable, dependent faults

Table 6.5: Array location of two variable, dependent faults

Gene Variable

Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Gene 6	Gene 7	Gene 8
1	1	1	1	1	1	1	1
101	2	108(L)	2	2	2	112	112 - 113L
102	3	109(L)	101 102	106	106	3	3
4	4	107	107	4	4	4	4
104	113L	5	5	5	111	111	5
103 105	6	6	110	110	6	6	6
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	104 105
9	9	103	109L	9	9	108L	9
10	10	10	10	10	10	10	10

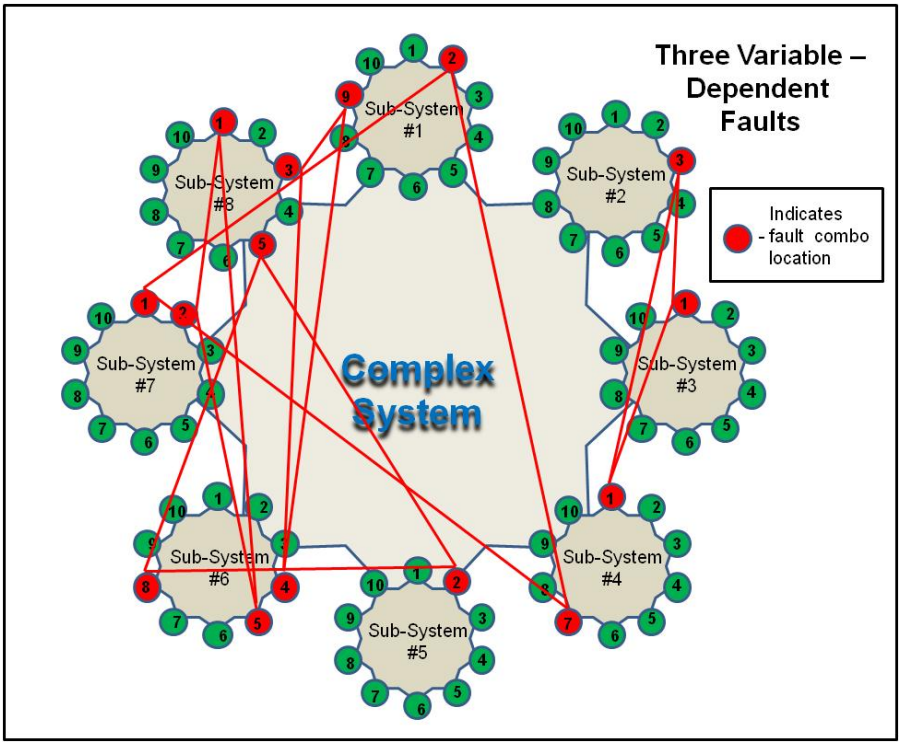
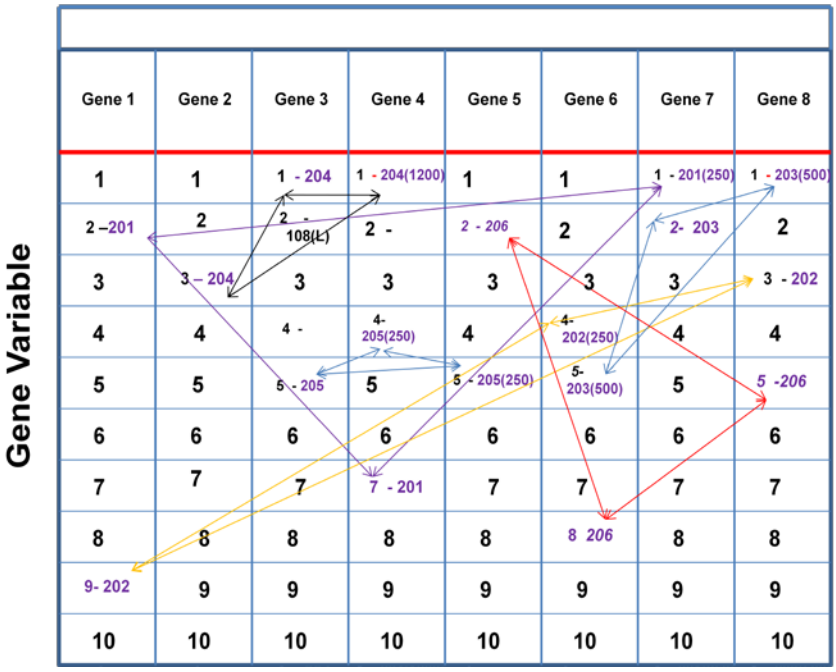


Figure 6.8: Graphical location of three variable, dependent faults

Table 6.6: Array location of three variable, dependent faults



6.3 Summary

This chapter presents the design of two case studies that illustrate models developed to aid the engineering team in sustainable product development. The first study compares generation to generation product designs relative to the sustainable products value proposition metrics. In the analysis of sustainable value creation, the definition of a common set of metrics can be a difficult task. For example, new technology has provided a boost to societies with the availability of advanced tools and solutions. This research refers to these new tools and technology as e-gains (gains produced via new electronic technology). With the continued exponential growth of new technology, a conundrum has developed, technology producers are in a cycle that encourages new product release and product turnover before the current product in use by the consumer, reaches it s useful end-of-life. A set of meaningful product design value metrics may differ greatly, depending on one's perspective.

In the sustainable products value proposition metrics used in the case study, emphasis is placed on the cost of the product over the entire lifecycle for the producer, consumer as well as society and the environment. Beyond the time value of money as the foundation for cost analysis, the time value of resources is used as the primary motivation to create Sustainable Lifetime Value. In the case study presented in this chapter, the comparison of the next generation design relative to the current product available to consumers, disclosed a 46% improvement in the relative value proposition design metrics.

As technology growths, so does the complexity of new products available to consumers. In order to improve long-term sustainable value of these new products, focus is placed on improving the verification of the designs relative to the intended value proposition. The second case study in this chapter is designed to exercise the adaptive genetic search algorithm for analysis. The intent of the search algorithm is to assist in the verification engineers in the test case generation process, in order to maximize fault detection and system risk reduction. In order to assist in the case study, a complex system fault simulator and designed experiment were designed to aid in the analysis relative to the research hypothesis and experiment variables. The results of the designed experiment are presented in chapter seven.

Chapter 7: Results and Discussion

7.1 Evaluation Priority and Criteria

Because of the complexity of the search model objectives, the results analysis of this case study is presented in a combination of formats. The adaptive genetic search algorithm is constructed based on the foundation of combinatory testing to take advantage of as many new two and three variable combinations as possible within each test run. Even with the use of combinations in the complex system presented in the case study, it would require 2800 unique tests to insure every two variable combination would be covered at least once. In addition, it would require 81,200 test cases to be sure every three variable combination would be tested at least once. If 100% confidence is required, there is no choice but to conduct the tests. Unfortunately, some product development teams have limited resources and must develop strategies that maximize the information gathered during the verification process within their risk tolerance. For example, a test case combination may be judged to have limited return on investment vs. the potential risk of the configuration. In the case study, the average cost of a test case is 44 resource units. For each treatment run, 175,000 test resource units are available in the algorithm bank. At an average cost of 44 units per test case, the average run would allow 3,977 test cases to be executed before resources were consumed. Therefore, the goal of the search algorithm is to take advantage of several factors that can potentially improve the efficiency and effectiveness of the search algorithm with limited test resources.

Determination of the designed experiment treatments' effectiveness will be based on the following prioritized criteria. The first priority of the search algorithm is to find as many embedded faults in the complex system simulator as possible with a fixed amount of resources. A weighted value of sixty percent of the total criteria is assigned to this goal. The treatments are ranked in order, starting with the lowest amount of resources

consumed at the point of the final detected fault. If all of the faults were not detected before the resources were consumed, a penalty was applied to the treatment. The second priority of the search algorithm is the acceleration of the reliability growth curve with a fixed amount of resources. In other words, discovering a fault earlier in the search process is desired. The average amount of resources consumed to discover all of the faults is the discrete metric used to compare the various search algorithm models. The treatments are ranked in order, starting with the lowest average amount of consumed resources, and a weighted value of thirty percent is applied to this criteria. The third priority is the amount of test cases required to find the embedded faults. Similar to the amount of resources consumed, the average amount of test cases required to discover all of the faults is calculated for each treatment. The treatments are ranked in order, starting with the lowest amount of test runs required to discover all of the embedded faults in the system, and a weighted value of ten percent is applied to this criteria. Each of the search algorithm priorities are presented separately, but the final summation will present the rank order of the overall weighted treatment results.

When creating a model to search for faults in a complex system, the statistical significance of an algorithm may be very effective on one system but not as effective on another due to several reasons. Faults are not intentionally injected into a design by the engineering team and, therefore, the amount of faults can vary greatly. Therefore, the goal of the adaptive genetic search algorithm is to seek improvement over the baseline approach of creating random test cases that seek faults in an unknown system. This approach is analogous to shooting a shotgun at a system and hoping to hit multiple targets. The intent is to take advantage of feedback to improve the search process but not create an algorithm that is too aggressive. There is still a high degree of test case diversity needed in order to ensure fault detection. Therefore, the assigned alpha (α) value for the general statistical significance of the search algorithm is set to $\alpha = 0.1$.

7.2 Treatment Results

7.2.1 Analysis Set-up

A two level, full factorial experiment was designed in order to test the effectiveness of the five independent variables in the adaptive search algorithm.

In the case study, an example of an experimental treatment is described by the ordered number set (10010). In this example, the high values are used on the first (actual cost of each variable) and fourth (conduct a sweep of all independent variables first) variables in the ordered set (Figure 7.1). The low values are assigned to the other three variables. This example happens to be treatment number 10 in the designed experiment. For each treatment, the discrete and continuous results are presented along with a summation of the interpretation of the effect.

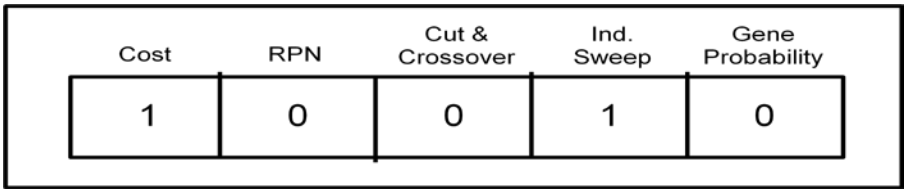


Figure 7.1: Example of experimental treatment

The continuous data results of two of the treatments will be used to compare to all of the other treatments in the designed experiment. The first treatment is number 1 (ordered set (00000)). This treatment is described as the control test case. Whereas this is the designed experiment test case assigned the low signal for all of the independent variables, it still takes advantage of combinatory testing as an efficient method to search for faults in the complex system. In essence, it takes advantage of a randomly generated and ordered set of sub-system variables to create the recommended test cases. In the case study, the locations of the faults are not known to the search algorithm, so a diverse set of test cases is necessary. Designed experiment analysis will seek to

understand if use of the five variables improves or decreases the efficiency of the search algorithm results. The second treatment used to compare against all others results is the variable combination identified in the initial experiment screenings (as well as the tabulated results of the designed experiment) as the most effective in attaining the desired goals. After repeating the designed experiment 8 times, treatment number 14 consistently had the highest ranked results in all three criteria and is considered the best of breed for comparison.

7.2.2 Initial Screening Results

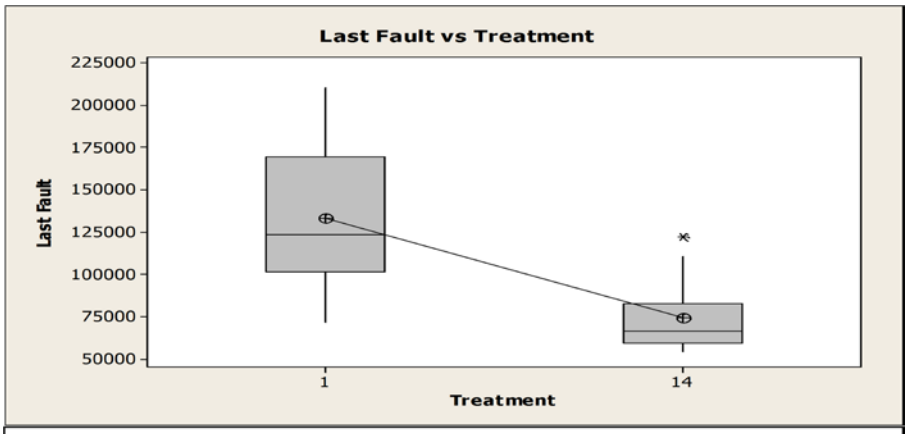
In analyzing the search algorithm model's effectiveness, the hypothesis for the initial screening test is that the independent variable combination represented by treatment number – 14 (10110), will have a statistically significant effect on improving the efficiency of the search model. Therefore, the null hypothesis (μ_0) for this comparison is that there is no statistical difference between the control treatment (00000) and treatment number 14 (10110). Thirty-two trials were completed in the adaptive search algorithm simulator to record the amount of resources and test cases required to discover the last fault. If all faults were not discovered, a penalty was assessed. The results of the experiment are presented in Table 7.1. In the table, if any of the treatments missed the discovery of a fault(s) after each run, the information was recorded and signified with red numbers.

Figure 7.2 presents statistical analysis that compares the amount of resources consumed at the time the last fault was discovered for each treatment run. Visual inspection of the bounding box graph shows that the amount of resources consumed in the fault discovery process for treatment 14 is less than the control (treatment number 1). The initial two-sample T-Test with a 95% confidence level had a P-Value of 0.000. Although the samples from each setting have similar characteristics, both samples are not normally distributed (see data summary in Figure 7.3). Therefore, an additional non-parametric statistical analysis (Mann-Whitney Test) was conducted to confirm the

rejection of the null hypothesis. The P-Value for this test was also 0.000, thus confirming that the results of the treatments are significantly different.

Table 7.1: Side by side comparison of best of breed treatment to control

Treatment No. 1	00000			Treatment No. 14	10110		
Run	Faults Missed	Last Fault Resource	Last Test Case	Run	Faults Missed	Last Fault Resource	Last Test Case
1	0	143424	2930	1	0	100169	2556
2	0	104160	2093	2	0	57951	1482
3	0	169728	3459	3	0	76857	1970
4	1	210124	4296	4	0	79318	2025
5	0	106896	2169	5	0	59011	1502
6	0	102816	2064	6	0	54408	1405
7	0	141072	2862	7	0	75680	1945
8	0	138480	2806	8	0	63796	1626
9	0	71520	1418	9	0	121841	3087
10	0	139200	2823	10	0	63904	1642
11	0	87312	1760	11	0	83367	2139
12	0	83760	1668	12	0	64759	1657
13	0	93984	1879	13	0	56984	1455
14	0	74352	1471	14	0	106528	2727
15	0	128304	2596	15	0	61129	1567
16	1	210125	4303	16	0	81027	2078
17	1	210125	4308	17	0	80712	2065
18	1	210125	4286	18	0	69412	1637
19	0	116880	2377	19	0	110399	2788
20	1	210009	4302	20	0	90514	2315
21	0	119760	2417	21	0	65963	1688
22	0	84144	1695	22	0	62618	1612
23	0	115968	2339	23	0	62611	1593
24	0	126384	2556	24	0	67249	1714
25	0	104352	2095	25	0	56740	1456
26	0	100704	2022	26	0	85702	2183
27	0	115968	2339	27	0	64341	1642
28	0	149424	3037	28	0	55332	1423
29	0	74688	1484	29	0	73470	1880
30	1	175104	3571	30	0	57222	1467
31	1	175056	3570	31	0	108696	2768
32	0	168304	3431	32	0	56864	1445



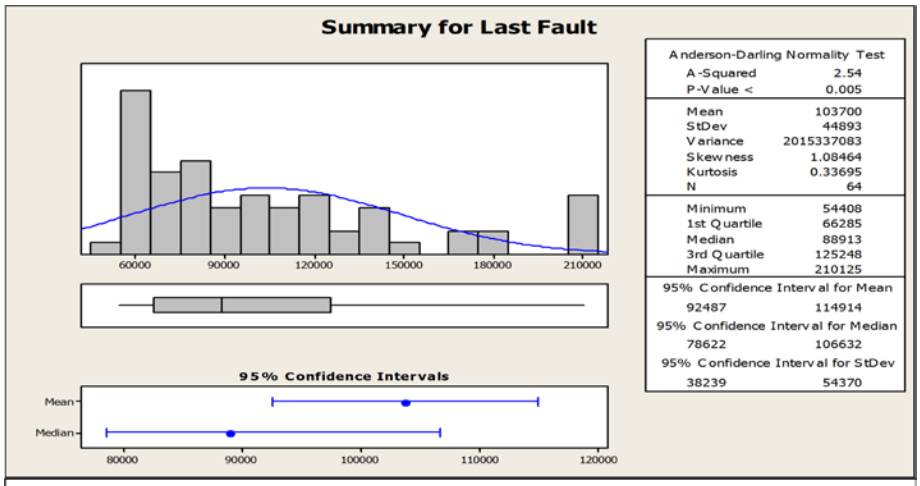
Two-Sample T-Test and CI: Last Fault, Treatment

```
Two-sample T for Last Fault

Treatment    N     Mean   StDev   SE Mean
1           32   133195   44279      7828
2           32    74205   18411      3255


Difference = mu (1) - mu (2)
Estimate for difference:  58990
95% CI for difference:  (41870, 76110)
T-Test of difference = 0 (vs not =): T-Value = 6.96  P-Value = 0.000
DF = 41
```

Figure 7.2: Statistical comparison of treatments 1 and 14



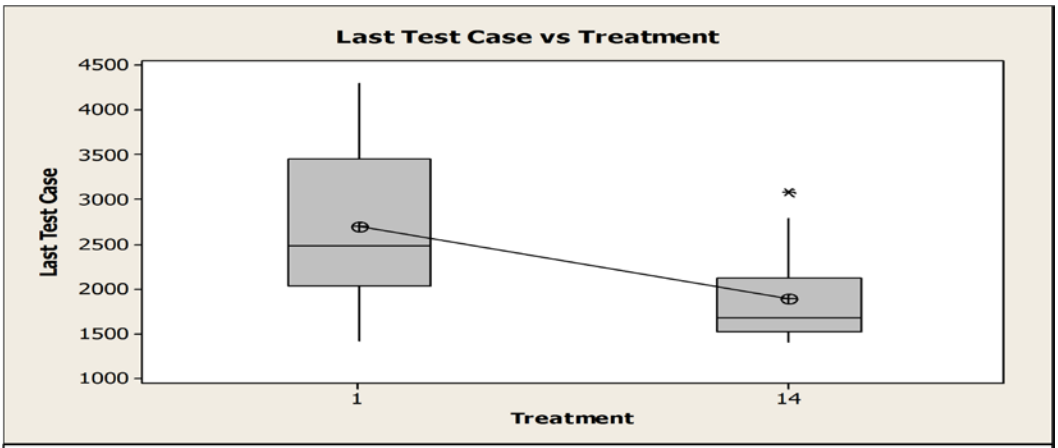
Mann-Whitney Test and CI: C6, C7

```
     N  Median
C6  32  123072
C7  32   66606


Point estimate for ETA1-ETA2 is 52172
90.3 Percent CI for ETA1-ETA2 is (39025,67373) - Zero is not in the C.I.
W = 1464.0
Test of ETA1 = ETA2 vs ETA1 not = ETA2 is significant at 0.0000 - Test
indicates the treatments are significantly different
The test is significant at 0.0000 (adjusted for ties)
```

Figure 7.3: Statistical analysis of last fault between treatment 1 and 14- Resource

Similarly, Figure 7.4 presents statistical analysis comparing the amount of test cases required to discover the last fault for each treatment run (column four in Table 7.1). Visual inspection of the bounding box graph shows that the amount of test cases required in the fault discovery process for treatment 14 is less than the control (treatment number 1). The initial two-sample T-Test with a 95% confidence level had a P-Value of 0.000. Although the samples from each setting have similar characteristics, both samples are not normally distributed (see data summary in Figure 7.5). Therefore, an additional non-parametric statistical analysis (Mann-Whitney Test) was conducted to confirm the rejection of the null hypothesis. The P-Value for this test was also 0.000, thus confirming the results of the treatments are significantly different. By taking advantage of three of the independent variables, improvements were attained in the adaptive genetic search algorithm.



Two-Sample T-Test and CI: Last Test Case, Treatment

Two-sample T for Last Test Case

Treatment	N	Mean	StDev	SE Mean
1	32	2701	920	163
2	32	1892	467	83

Difference = mu (1) - mu (2)
Estimate for difference: 809
95% CI for difference: (442, 1176)
T-Test of difference = 0 (vs not =): T-Value = 4.43 P-Value = 0.000
DF = 45

Figure 7.4: Statistical analysis of treatments 1 and 14 - Test Case

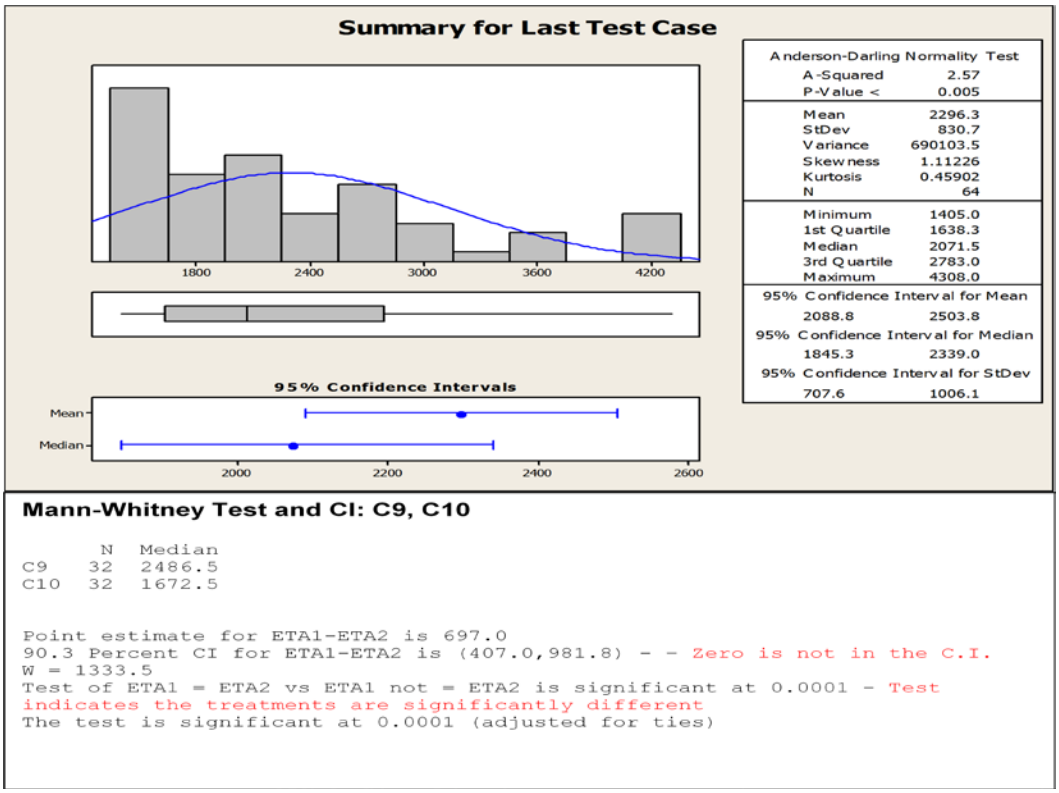


Figure 7.5: Statistical analysis of last test case between treatment 1 and 14

The initial results of the full factorial designed experiment are presented in Table 7.2. Initial observations provide insight into the effects of the independent variable, but further analysis of the designed experiment is required to provide general information on the effects and interactions that each variable has on the overall efficiency of the test development and search process.

7.2.3 Priority No. 1: Fault Detection Efficiency

Whereas there are three criteria identified in the analysis of the effectiveness of the search algorithm variables, the discrete data from each will be presented and then the aggregated weighted average of the three will be presented.

Table 7.2: Initial DOE Screening Results -Missed Faults

Treatment NO.	DOE Order	Total Undetected Faults
2	10000	0
4	11000	0
6	10100	0
8	11100	0
9	00010	0
10	10010	0
12	11010	0
14	10110	0
16	11110	0
20	11001	0
3	01000	1
5	00100	1
13	00110	1
22	10101	1
26	10011	1
28	11011	1
32	11111	1
7	01100	2
15	01110	2
19	01001	2
25	00011	2
29	00111	2
30	10111	2
1	00000	3
11	01010	3
24	11101	3
17	00001	4
18	10001	4
21	00101	5
31	01111	5
23	01101	6
27	01011	6

The first priority is the ability to discover as many faults as possible within a fixed amount of resources. The treatments are presented in Table 7.3 and are grouped (by color shading of number of faults left undetected) according to their effectiveness. Of the thirty-two combinations, only ten treatments discovered all of the embedded faults in the complex system after completing all eight runs in the simulator. Seven treatments missed one fault after repeating the treatment eight times and six treatments missed two faults. Nine treatments missed 3 or more faults after the eight test runs. Table 7.3 ranks the treatments within their respective groupings based on the average amount of resources required to discover the final fault. Based on the criteria, treatment number 14 (10110) was the most efficient variable combination by consuming an average of only 66,610 test resource units. After treatment 14, the next four treatment numbers, based on search efficiency and detecting all faults, were 12 (11010), 20 (11001), 10 (10010) and 6 (10100) respectively.

On the other extreme, the last four treatments in the ranking (21-(00101), 31-(01111), 27-(01011), 23-(01101)) left 5-6 faults undiscovered after 8 test runs. For all of the treatments ranked in the top five of the most effective combinations, the common factor is that all had the high signal for the cost variable and the low value for all but one of the GA-probability variable. The common factor for all of the treatments ranked in the bottom five is all combinations had the low signal for the cost variable and the high value for the GA-probability value. The cost and probability variables have a strong relationship. The other three variables have mixed effects but are involved with variable interactions effects. Therefore, the following section details the statistical analysis of the DOE that is focused on analysis of the five independent model variables against the first priority of detecting as many embedded faults as possible before all resources are consumed.

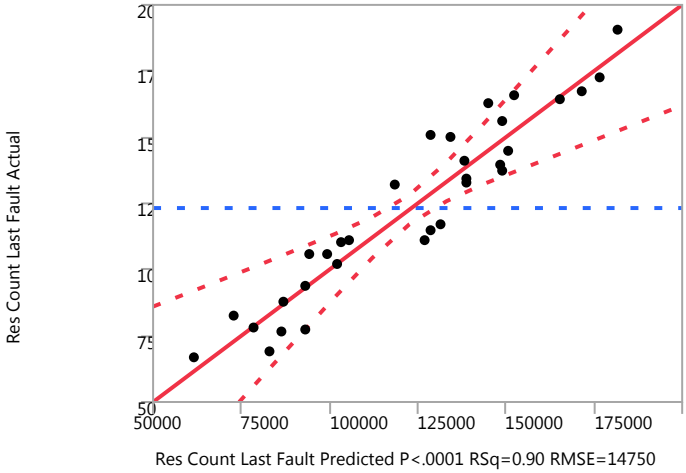
Table 7.3: DOE results - resource count to detect last fault

Treatment NO.	DOE Order	Total Undetected Faults	Adjusted Res Count Last Fault	Rank
14	10110	0	66610	1
12	11010	0	76167	2
20	11001	0	77173	3
10	10010	0	77524	4
6	10100	0	78754	5
4	11000	0	82352	6
8	11100	0	87725	7
2	10000	0	93810	8
16	11110	0	105814	9
9	00010	0	110406	10
28	11011	1	101469	11
26	10011	1	105332	12
13	00110	1	110895	13
22	10101	1	114554	14
32	11111	1	116829	15
3	01000	1	134078	16
5	00100	1	149892	17
30	10111	2	109677	18
25	00011	2	132874	19
7	01100	2	137069	20
15	01110	2	139145	21
29	00111	2	141076	22
19	01001	2	144787	23
24	11101	3	150512	24
1	00000	3	156032	25
11	01010	3	162376	26
18	10001	4	131955	27
17	00001	4	163752	28
21	00101	5	167308	29
31	01111	5	172333	30
27	01011	6	165486	31
23	01101	6	190402	32

With an α value of 0.1, several independent variables and their interactions have statistical significance in the effectiveness of the search algorithm (see details in Figure 7.6). These variables include the following: the use of actual test variable costs, not using the risk values for fault mitigation priority, the use of the independent variable sweep, and not modifying the Gene selection probability based on passed test case successes. In addition, there are two interactions that were statistically significant. Due

to the nature of a blind search algorithm, several of the other variable combinations, particularly interactions, appear to have an effect on the fault search efficiency. Within the full factorial experiment (repeated 8 times), the actual variation between treatment runs was significant enough to reduce confidence in the repeatability of the effect. Taking advantage of the actual cost and conducting independent variable tests before combinatory tests had a strong influence on minimizing resource consumption in the fault search algorithm.

Response Res Count Last Fault
Actual by Predicted Plot



Summary of Fit

RSquare	0.90325
RSquare Adj	0.812547
Root Mean Square Error	14750.44
Mean of Response	123255.3
Observations (or Sum Wgts)	32

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	15	3.25e+10	2.1667e+9	9.9583
Error	16	3481207616	217575476	**Prob > F**
C. Total	31	3.5982e+10		<.0001*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	123255.25	2607.534	47.27	<.0001*
Cost	-25364.19	2607.534	-9.73	<.0001*
RPN	4477.0625	2607.534	1.72	0.1053
Crossover	3531.9375	2607.534	1.35	0.1944
1D sweep	-4879.438	2607.534	-1.87	0.0797
Probability	13339.688	2607.534	5.12	0.0001*
Cost*RPN	-2613	2607.534	-1.00	0.3312
Cost*Crossover	1136.375	2607.534	0.44	0.6688
RPN*Crossover	6214.375	2607.534	2.38	0.0299*
Cost*1D sweep	1916.125	2607.534	0.73	0.4731
RPN*1D sweep	7099.5	2607.534	2.72	0.0151*
Crossover*1D sweep	-1610.375	2607.534	-0.62	0.5455
Cost*Probability	2206.875	2607.534	0.85	0.4098
RPN*Probability	-1198.125	2607.534	-0.46	0.6521

Term	Estimate	Std Error	t Ratio	Prob>|t|
Crossover*Probability	5209.5	2607.534	2.00	0.0630
1D sweep*Probability	-1081	2607.534	-0.41	0.6840

Sorted arameter Estimates

Term	Estimate	Std Error	t Ratio	t Ratio	Prob>|t|
Cost	-25364.19	2607.534	-9.73		<.0001*
Probability	13339.688	2607.534	5.12		0.0001*
RPN*1D sweep	7099.5	2607.534	2.72		0.0151*
RPN*Crossover	6214.375	2607.534	2.38		0.0299*
Crossover*Probability	5209.5	2607.534	2.00		0.0630
1D sweep	-4879.438	2607.534	-1.87		0.0797
RPN	4477.0625	2607.534	1.72		0.1053
Crossover	3531.9375	2607.534	1.35		0.1944
Cost*RPN	-2613	2607.534	-1.00		0.3312
Cost*Probability	2206.875	2607.534	0.85		0.4098
Cost*1D sweep	1916.125	2607.534	0.73		0.4731
Crossover*1D sweep	-1610.375	2607.534	-0.62		0.5455
RPN*Probability	-1198.125	2607.534	-0.46		0.6521
Cost*Crossover	1136.375	2607.534	0.44		0.6688
1D sweep*Probability	-1081	2607.534	-0.41		0.6840

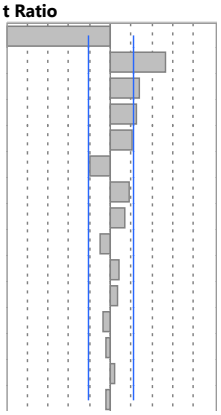


Figure 7.6: DOE results to analyze fault detection efficiency

7.2.4 Priority No. 2: Early Fault Detection

The second priority in evaluating these experimental results was choosing the variable combinations that discover a maximum amount of faults with the least amount of test resources. The discrete metric identified as the comparative signal was the calculated average of the consumed resource count at the point each of the faults were discovered. The lower the relative average number signals a more effective variable combination. Table 7.4 presents the treatments ranked in order within their respective fault detection capabilities. The top five (most effective test combinations) in order were as follows: treatment 14 (10110); treatment 10 (10010); treatment 6 (10100); treatment 2 (10000); and treatment 12 (11010). In the first priority (fault detection capability), treatment 14 was also the most efficient, but treatments 6 and 2 did not make the top five list. Although the goals of maximum fault detection and early fault detection take advantage of similar strategic characteristics, the goals are not the same. All five variable combinations were assigned the high value in the first (cost) position

and the low value in the fifth (GA-probability) positions. The other positions had mixed values.

The bottom five (least effective capability) ranked in order 28th through 32th were as follows: treatment 17 (00001), treatment 21 (00101); treatment 31 (01111); treatment 27 (01011); and treatment 23 (01101). Unlike the top five combinations, the bottom five for the second criteria is the same as the first priority, including the order. Each of the bottom five treatments had the low value in the first position and the high value in the fifth position. These results show a strong relationship between cost and GA-probability variables and potential interactions between the other combinations. The following section details the DOE statistical analysis. Focus in on analysis of the five independent model variables against the second priority of detecting the most embedded faults early with the least amount of resources and before all resources are consumed.

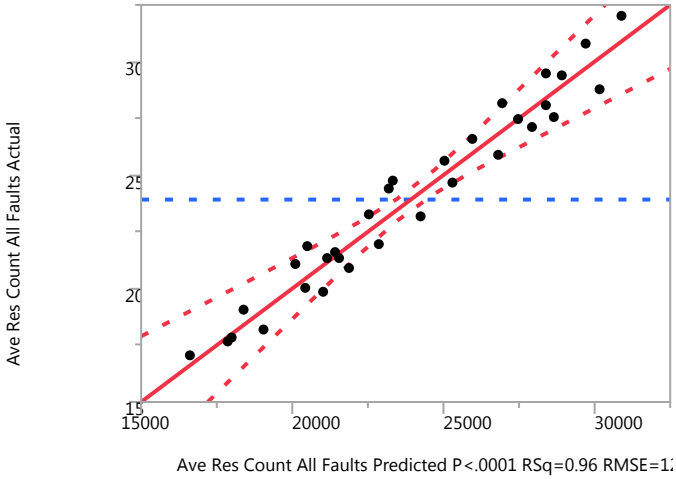
Table 7.4: DOE results – average resource count to detect all faults

Treatment NO.	DOE Order	Total Undetected Faults	Adjusted Ave Res Count	Rank
14	10110	0	16975	1
10	10010	0	17595	2
6	10100	0	17770	3
2	10000	0	18164	4
12	11010	0	19981	5
16	11110	0	21343	6
8	11100	0	21568	7
4	11000	0	21788	8
20	11001	0	21870	9
9	00010	0	23101	10
22	10101	1	19781	11
32	11111	1	20882	12
26	10011	1	21077	13
28	11011	1	21308	14
13	00110	1	24401	15
5	00100	1	25558	16
3	01000	1	27394	17
30	10111	2	19044	18
29	00111	2	24587	19
25	00011	2	25892	20
15	01110	2	27041	21
7	01100	2	27533	22
19	01001	2	28700	23
24	11101	3	24690	24
1	00000	3	26512	25
11	01010	3	28086	26
18	10001	4	23262	27
17	00001	4	30731	28
21	00101	5	28040	29
31	01111	5	29373	30
27	01011	6	29428	31
23	01101	6	31997	32

With an α value of 0.1, several independent variables and their interactions have statistical significance in determining the effectiveness of the search algorithm (see

details in Figure 7.7). These variables include the following: the use of actual test variable costs, use of the risk values for fault mitigation priority, use of the independent variable sweep, and not modifying the Gene selection probability based on passed test case successes. In addition, there is one interaction that was statistically significant. Due to the nature of a blind search algorithm, several of the other variable combinations, particularly interactions, appear to have an effect on the fault search efficiency. Within the full factorial experiment (repeated 8 times), the actual variation between treatment runs was significant enough to reduce confidence in the repeatability of the effect. Taking advantage of the actual cost and conducting independent variable tests before combinatory tests had a strong influence on minimizing the average resource consumption in the fault search algorithm.

Response Ave Res Count All Faults
Actual by Predicted Plot



Summary of Fit

RSquare	0.955475
RSquare Adj	0.913732
Root Mean Square Error	1225.947
Mean of Response	23921
Observations (or Sum Wgts)	32

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	15	516028849	34401923	22.8896
Error	16	24047149	1502946.8	**Prob > F**
C. Total	31	540075998		<.0001*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	23921	216.7189	110.38	<.0001*
Cost	-3477.375	216.7189	-16.05	<.0001*
RPN	1265.375	216.7189	5.84	<.0001*
Crossover	-134.5625	216.7189	-0.62	0.5434
1D sweep	-788.875	216.7189	-3.64	0.0022*
Probability	1120.375	216.7189	5.17	<.0001*
Cost*RPN	-30.25	216.7189	-0.14	0.8907
Cost*Crossover	-52.4375	216.7189	-0.24	0.8119
RPN*Crossover	501.5625	216.7189	2.31	0.0343*
Cost*1D sweep	120.875	216.7189	0.56	0.5847
RPN*1D sweep	282.75	216.7189	1.30	0.2105
Crossover*1D sweep	-41.8125	216.7189	-0.19	0.8494
Cost*Probability	-74.75	216.7189	-0.34	0.7346
RPN*Probability	-275.75	216.7189	-1.27	0.2214
Crossover*Probability	-107.5625	216.7189	-0.50	0.6264
1D sweep*Probability	-303.625	216.7189	-1.40	0.1803

Sorted Parameter Estimates

Term	Estimate	Std Error	t Ratio	t Ratio	Prob>|t|
Cost	-3477.375	216.7189	-16.05		<.0001*
RPN	1265.375	216.7189	5.84		<.0001*
Probability	1120.375	216.7189	5.17		<.0001*
1D sweep	-788.875	216.7189	-3.64		0.0022*
RPN*Crossover	501.5625	216.7189	2.31		0.0343*
1D sweep*Probability	-303.625	216.7189	-1.40		0.1803
RPN*1D sweep	282.75	216.7189	1.30		0.2105
RPN*Probability	-275.75	216.7189	-1.27		0.2214
Crossover	-134.5625	216.7189	-0.62		0.5434
Cost*1D sweep	120.875	216.7189	0.56		0.5847
Crossover*Probability	-107.5625	216.7189	-0.50		0.6264
Cost*Probability	-74.75	216.7189	-0.34		0.7346
Cost*Crossover	-52.4375	216.7189	-0.24		0.8119
Crossover*1D sweep	-41.8125	216.7189	-0.19		0.8494
Cost*RPN	-30.25	216.7189	-0.14		0.8907

Figure 7.7: DEO results for analysis of average resources consumed to detect all faults

7.2.5 Priority No. 3: Average Test Case Count

The third priority in evaluating the experimental results was focused on choosing the variable combinations that discovers a maximum amount of faults with the least amount of test cases. The discrete metric identified as the comparative signal was the calculated average of the total amount of test cases required to find all of the faults. The lower the relative average number signals a more effective variable combination. Figure 100 presents the treatments ranked in order within the respective fault detection capabilities. The top five (most effective test combinations) in order were as follows: treatment 6 (10100); treatment 9 (00010); treatment 14 (10110); treatment 10 (10010); and treatment 2 (10000). In the first priority (fault detection capability), treatment 14 was also the most efficient, but treatments 9 and 2 did not make the top five list. While the goals of maximum fault detection and minimum amount of test cases are to take advantage of similar strategic characteristics, the goals are not the same. All five variable combinations were assigned the low value in the second (risk prioritization) position as well as the low value in the fifth (GA-probability) positions. The other

positions had mixed values. An engineer may choose to focus on the results of this criterion, if minimal testing takes priority over total verification expense.

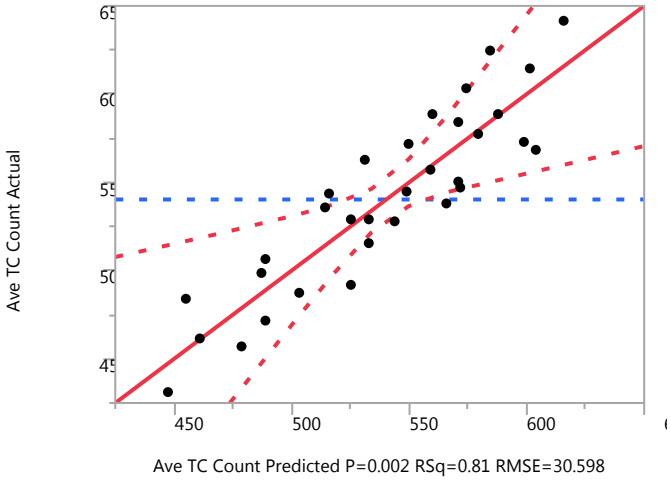
The bottom five (least effective capability), ranked in order 28th through 32th, were the following: treatment 17 (00001), treatment 21 (00101); treatment 31 (01111); treatment 27 (01011); and treatment 23 (01101). Unlike the top five combinations, the bottom five for the second criteria is the same as the first priority and the second priority, including the order. Each of the bottom five treatments had the low value in the first position and the low value in the fifth position. These results show a strong relationship between cost and GA-probability variables and potential interactions between the other combinations. The following section details the statistical analysis of the designed experiment. The focus is on the analysis of the five independent model variables relative to the third priority (detecting the most embedded faults possible, with the least amount of test cases, and before all resources are consumed).

Table 7.5: DOE results – test case count for last fault

Treatment NO.	DOE Order	Total Undetected Faults	Adjusted Ave TC Count	Rank
6	10100	0	431	1
9	00010	0	457	2
14	10110	0	461	3
10	10010	0	471	4
2	10000	0	498	5
12	11010	0	527	6
8	11100	0	528	7
4	11000	0	535	8
20	11001	0	547	9
16	11110	0	603	10
13	00110	1	484	11
22	10101	1	491	12
5	00100	1	506	13
3	01000	1	544	14
32	11111	1	568	15
26	10011	1	571	16
28	11011	1	577	17
7	01100	2	550	18
29	00111	2	487	19
25	00011	2	515	20
15	01110	2	538	21
30	10111	2	543	22
19	01001	2	572	23
1	00000	3	528	24
11	01010	3	562	25
24	11101	3	624	26
18	10001	4	584	27
17	00001	4	614	28
21	00101	5	557	29
31	01111	5	588	30
23	01101	6	641	31
27	01011	6	588	32

With an α value of 0.1, several independent variable and their interactions have statistical significance in determining the effectiveness of the search algorithm (see details in Figure 7.8). These variables include the following: using the risk values for fault mitigation priority and not modifying the Gene selection probability based on passed test case successes. In addition, there are two interactions that were significantly significant. Due to the nature of a blind search algorithm, several of the other variable combinations, particularly interactions, appear to have an effect on the fault search efficiency, but, within the full factorial experiment (repeated 8 times), the actual variation between treatment runs was significant enough to reduce confidence in the repeatability of the effect. Unlike the analysis of the previous two priorities, the cost and independent sweep variables do not have a significant effect on the reduction of test cases, but there is statistical significance between their interactions.

Response Ave TC Count
Actual by Predicted Plot



Summary of Fit

RSquare	0.81386
RSquare Adj	0.639353
Root Mean Square Error	30.5982
Mean of Response	540.3125
Observations (or Sum Wgts)	32

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	15	65496.875	4366.46	4.6638
Error	16	14980.000	936.25	**Prob > F**
C. Total	31	80476.875		0.0020*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	540.3125	5.409049	99.89	<.0001*
Cost	-5.375	5.409049	-0.99	0.3352
RPN	27.9375	5.409049	5.16	<.0001*
Crossover	-2.8125	5.409049	-0.52	0.6102
1D sweep	-6.5625	5.409049	-1.21	0.2426
Probability	26.375	5.409049	4.88	0.0002*
Cost*RPN	0.75	5.409049	0.14	0.8915
Cost*Crossover	-1	5.409049	-0.18	0.8556
RPN*Crossover	14.5625	5.409049	2.69	0.0160*
Cost*1D sweep	11.75	5.409049	2.17	0.0452*
RPN*1D sweep	7.1875	5.409049	1.33	0.2026
Crossover*1D sweep	3.0625	5.409049	0.57	0.5791
Cost*Probability	1.8125	5.409049	0.34	0.7419
RPN*Probability	-6.5	5.409049	-1.20	0.2470
Crossover*Probability	-1.5	5.409049	-0.28	0.7851
1D sweep*Probability	-5.5	5.409049	-1.02	0.3244

Sorted Parameter Estimates

Term	Estimate	Std Error	t Ratio	t Ratio	Prob>|t|
RPN	27.9375	5.409049	5.16		<.0001*
Probability	26.375	5.409049	4.88		0.0002*
RPN*Crossover	14.5625	5.409049	2.69		0.0160*
Cost*1D sweep	11.75	5.409049	2.17		0.0452*
RPN*1D sweep	7.1875	5.409049	1.33		0.2026
1D sweep	-6.5625	5.409049	-1.21		0.2426
RPN*Probability	-6.5	5.409049	-1.20		0.2470
1D sweep*Probability	-5.5	5.409049	-1.02		0.3244
Cost	-5.375	5.409049	-0.99		0.3352
Crossover*1D sweep	3.0625	5.409049	0.57		0.5791
Crossover	-2.8125	5.409049	-0.52		0.6102
Cost*Probability	1.8125	5.409049	0.34		0.7419
Crossover*Probability	-1.5	5.409049	-0.28		0.7851
Cost*Crossover	-1	5.409049	-0.18		0.8556
Cost*RPN	0.75	5.409049	0.14		0.8915

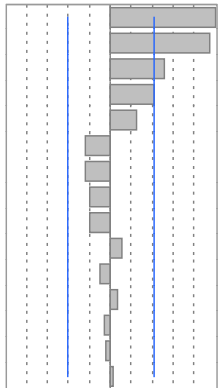


Figure 7.8: DOE results to analyze test case count to find last fault

Combining the results of all three priorities sheds light on the most significant variables with regard for designing the most effective search algorithm, but the results of each

priority are slightly different. Tables 7.6 and 7.7 present the aggregated results from all three criteria.

Table 7.6: Aggregated results table for three discrete priority metrics

Treatment NO.	DOE Order	Total Undetected Faults	Adjusted Ave TC Count	Adjusted Ave Res Count	Adjusted Res Count Last Fault
1	00000	3	528	26512	156032
2	10000	0	498	18164	93810
3	01000	1	544	27394	134078
4	11000	0	535	21788	82352
5	00100	1	506	25558	149892
6	10100	0	431	17770	78754
7	01100	2	550	27533	137069
8	11100	0	528	21568	87725
9	00010	0	457	23101	110406
10	10010	0	471	17595	77524
11	01010	3	562	28086	162376
12	11010	0	527	19981	76167
13	00110	1	484	24401	110895
14	10110	0	461	16975	66610
15	01110	2	538	27041	139145
16	11110	0	603	21343	105814
17	00001	4	614	30731	163752
18	10001	4	584	23262	131955
19	01001	2	572	28700	144787
20	11001	0	547	21870	77173
21	00101	5	557	28040	167308
22	10101	1	491	19781	114554
23	01101	6	641	31997	190402
24	11101	3	624	24690	150512
25	00011	2	515	25892	132874
26	10011	1	571	21077	105332
27	01011	6	588	29428	165486
28	11011	1	577	21308	101469
29	00111	2	487	24587	141076
30	10111	2	543	19044	109677
31	01111	5	588	29373	172333
32	11111	1	568	20882	116829

7.2.6 Weighted Rank Summary

After the weighted values of the three assessment priorities are applied to the results, the aggregated rank order was calculated and presented in Figure 102. The top five (most effective test combinations) in order were as follows:

1. Treatment No. 14 (10110)
2. Treatment No. 12 (11010)
3. Treatment No. 10 (1010)
4. Treatment No. 6 (10100)
5. Treatment No. 20 (11001)

The results show a clear positive effect from taking advantage of actual cost in calculating the expense of each test case vs. treating all test case expenses as equal expense. The remainder of the results show there are competing interactions between the experiment variable that produced varied results depending on the priority. The most unusual result is treatment number 20 is the 5^{th} most effective combination for the search algorithm. The majority of treatments in the top of the rankings did not alter the probability of the genetic algorithm selection process. The results suggest cases that took advantage of the Gene selection probability became too aggressive and exceeded the limits of test case diversity required to search for the embedded faults. The exception is treatment number 20, which combines actual cost with prioritizing fault mitigation and altering Gene selection probability. The bottom five (least effective test combinations) in order were as follows:

5. Treatment No. 17 (00001)
4. Treatment No. 21 (00101)
3. Treatment No. 31 (01111)
2. Treatment No. 27 (01011)
1. Treatment No. 23 (01101)

It is important to acknowledge that the control case for the combinatory test case generation is treatment number 1 (00000). By only taking advantage of the random test

case generation process for each chromosome, this combination of variables was rated the 8th worst of the 32 combinations. These results imply that taking advantage of one or more variables led to improvements in the search efficiency on 24 of the treatments but were less effective on 7 of the treatments. Attempting to improve the search efficiency of the adaptive genetic search algorithm with the high value on the probability modification and the low value of the cost variable generally produced poor results. Another interesting result is the rank position of treatment number 9 (00010). The only modification made to the test case generator in the search algorithm was the signal to systematically sweep (and test) each independent variable first to be sure they were functioning properly before spending resources on the complex system combinatory testing. This data validate the common sense rule of thumb that it is cost effective to be sure all Sub-systems are functioning as expected before submitting the complex system for verification testing.

7.3 Continuous Data Results Analysis

In the following section, continuous data results are presented for tracking fault detection and risk mitigation over the total consumption. Five specific treatments were chosen to summarize the results. The graphs and data for the remaining treatments are presented in the appendix. Whereas the final result of any given treatment may resemble that of another, the curve shapes over the consumption of resources may be very different. For example, treatment number 14 was the most effective combination of test variables, but, considering the potential variation in complex system fault patterns, there is minimal statistical difference between the top five treatments.

The five treatments chosen for the summary are the following: No. 1 (00000) because it is the control case; No. 32 (11111) because it was the hypothesized best case; treatment No. 23 (01101) because it turned out to be the least efficient case; No. 2 (10000) because it isolates the sensitivity of the cost variable in the search efficiency; and No. 14 (10110) because it is the most efficient variable combination. For each treatment, two

charts are presented that highlight the goals of fault detection and risk mitigation over the consumption of the test resources. In addition to the graphs, the data from each of the test runs is presented in a table along with brief commentary of the results relative to the best of breed (number 14 – (10110) and control treatments (number 1 – (00000). Analyzing the growth of each curve during the course of the resource consumption provides insight into the greediness and early effectiveness of the treatment.

Table 7.7 Results table of the ranked order of search algorithm analysis treatments

Treatment No.	DOE	Missed Faults Rank	1. Res Last Fault Rank	Weight (60%)	2. Ave Res Rank	Weight (30%)	3. Ave Test Case Rank	Weight (10%)	Overall Rank
14	10110	11-17	1	0.6	1	0.3	3	0.3	1.2
12	11010	11-17	2	1.2	5	1.5	6	0.6	3.3
10	10010	Top 10	4	2.4	2	0.6	4	0.4	3.4
6	10100	Top 10	5	3	3	0.9	1	0.1	4
20	11001	18-23	3	1.8	9	2.7	9	0.9	5.4
2	10000	Top 10	8	4.8	4	1.2	5	0.5	6.5
4	11000	Top 10	6	3.6	8	2.4	8	0.8	6.8
8	11100	Top 10	7	4.2	7	2.1	7	0.7	7
9	00010	Top 10	10	6	10	3	2	0.2	9.2
16	11110	11-17	16	9.6	6	1.8	10	1	12.4
28	11011	27-28	11	6.6	14	4.2	17	1.7	12.5
26	10011	24-26	12	7.2	13	3.9	16	1.6	12.7
22	10101	18-23	14	8.4	11	3.3	12	1.2	12.9
13	00110	11-17	13	7.8	15	4.5	11	1.1	13.4
32	11111	31-32	15	9	12	3.6	15	1.5	14.1
3	01000	Top 10	16	9.6	17	5.1	14	1.4	16.1
5	00100	Top 10	17	10.2	16	4.8	13	1.3	16.3
30	10111	29-30	18	10.8	18	5.4	22	2.2	18.4
25	00011	24-26	19	11.4	20	6	20	2	19.4
7	01100	Top 10	20	12	22	6.6	18	1.8	20.4
29	00111	29-30	22	13.2	19	5.7	19	1.9	20.8
15	01110	11-17	21	12.6	21	6.3	21	2.1	21
19	01001	18-23	23	13.8	23	6.9	23	2.3	23
24	11101	24-26	24	14.4	24	7.2	26	2.6	24.2
1	00000	Top 10	25	15	25	7.5	24	2.4	24.9
11	01010	11-17	26	15.6	26	7.8	25	2.5	25.9
18	10001	18-23	27	16.2	27	8.1	27	2.7	27
17	00001	11-17	28	16.8	28	8.4	28	2.8	28
21	00101	18-23	30	18	29	8.7	29	2.9	29.6
31	01111	31-32	30	18	30	9	30	3	30
27	01011	27-28	32	19.2	31	9.3	32	3.2	31.7
23	01101	18-23	32	19.2	32	9.6	31	3.1	31.9

7.3.1 Treatment 1 (00000) Results

Treatment 1 is assigned the low signal values for all five variables in the designed experiment.

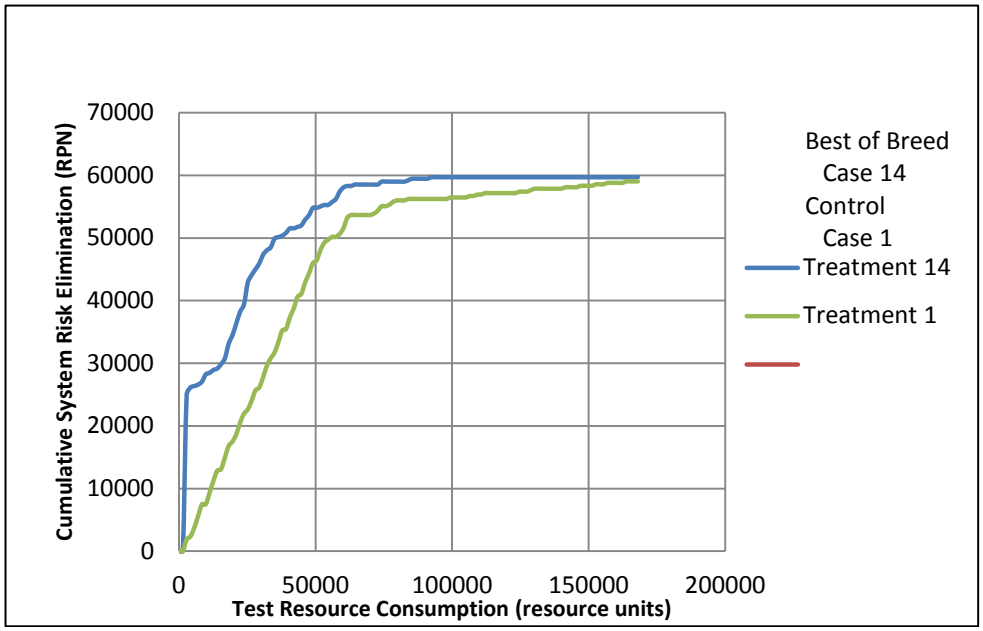


Figure 7.9: Risk mitigation, treatment 1 vs. 14

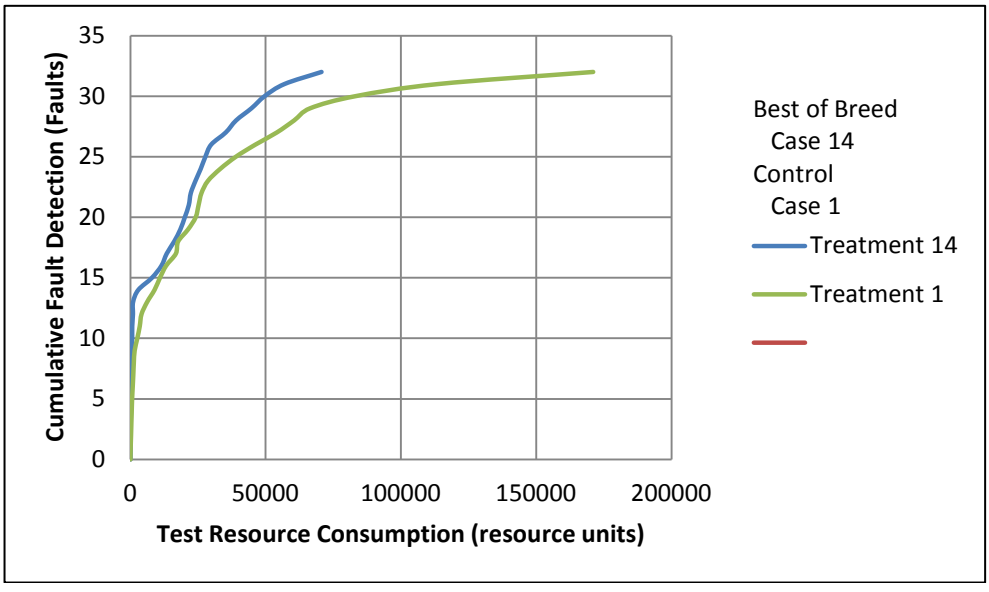


Figure 7.10: Cumulative fault detection, treatment 1 vs. 14

Summary

After repeating the treatment eight times, a total of 3 faults were left undetected in the simulation (see data Table 7.8 for details).

After eight treatment runs, the average amount of resources consumed to detect the last fault in the complex system simulator was 156032 resource units out of a possible 175000. This treatment ranked 25th out of the 32 treatments in this category.

The average amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs, was 26512 resource units. This treatment ranked 25th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs, was 528. This treatment ranked 24th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 25th out of 32.

Comparing treatments, combination number 14 took advantage of early independent and dependent fault detection and demonstrated accelerated risk mitigation growth. In addition, treatment number 14 discovered all of the faults with approximately half the amount of resources required in treatment number 1.

Table 7.8: Compiled DOE results for treatment number 1

Treatment # 1									
Treatment Order 00000	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	353	368	575	556	491	472	506	546	483
Average Resource Count	18159	18674	28776	27904	24573	23896	25408	27382	24347
Resource Count at Last Fault	83808	75792	155040	162528	145968	109104	140832	151872	128118
No. of undetected faults	0	0	0	0	1	1	0	1	
Total Resource Consumption					175008	175104		175104	
Total Test Case					3588	3571		3648	
Adjusted Ave TC Count	353	368	575	556	610	591	506	666	528
Adjusted Ave Res Count	18159	18674	28776	27904	30368	29716	25408	33093	26512
Adjusted Res Count Last Fault	83808	75792	155040	162528	210010	210125	140832	210125	156032

7.3.2 Treatment 32 (11111) Results

Treatment 32 is assigned the high signal values for all five variables in the designed experiment.

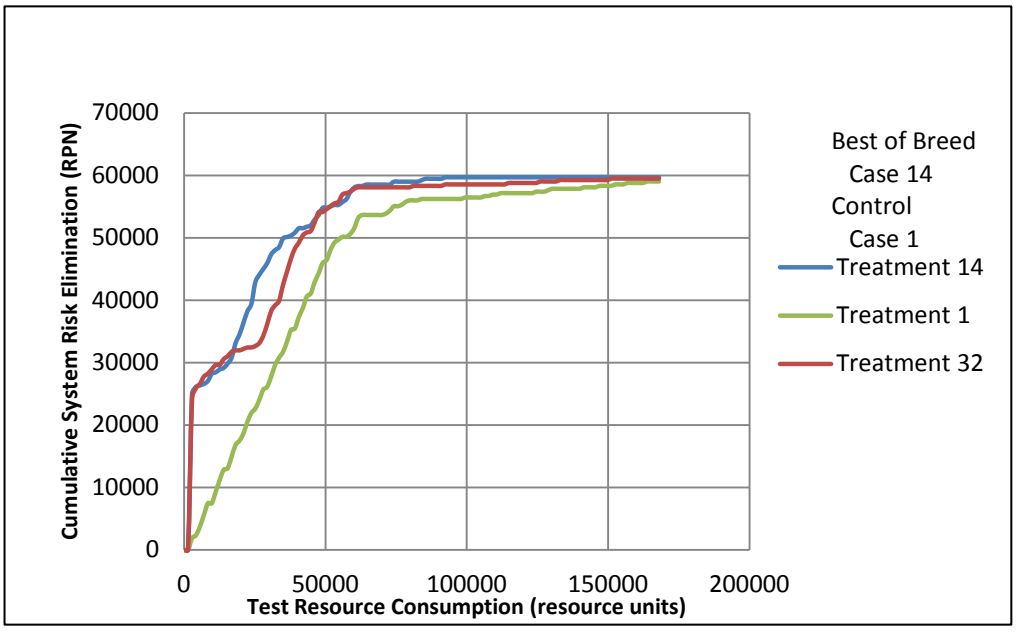


Figure 7.11: Risk mitigation comparison, treatments 1, 14, and 32

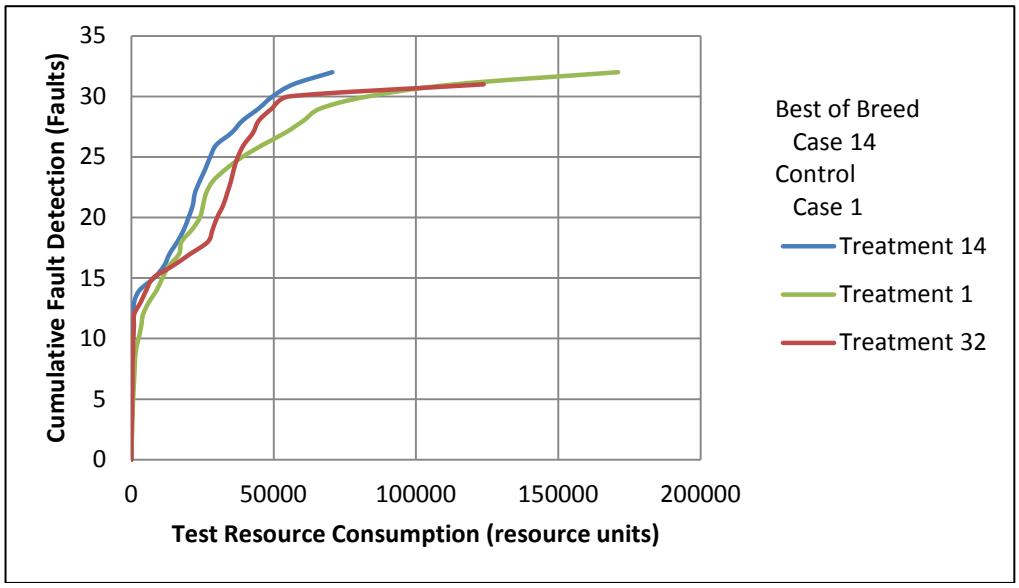


Figure 7.12: Cumulative fault detection comparison, treatments 1, 14, and 32

Table 7.9: Compiled DOE results for treatment number 32

Treatment # 32									
Treatment Order 11111	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	575	465	573	537	488	538	559	652	548
Average Resource Count	20941	16704	20881	19496	18598	19611	20573	24210	20127
Resource Count at Last Fault	79395	52936	113407	113088	54662	90205	124598	149163	97182
No. of undetected faults	0	1	0	0	0	0	0	0	
Total Resource Consumption		175097		0					
Total Test Case		4652							
Adjusted Ave TC Count	575	625	573	537	488	538	559	652	568
Adjusted Ave Res Count	20941	22748	20881	19496	18598	19611	20573	24210	20882
Adjusted Res Count Last Fault	79395	210116	113407	113088	54662	90205	124598	149163	116829

Summary

After repeating the treatment eight times a total of 1 fault was left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 116829 resource units out of a possible 175000. This treatment ranked 15th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 20882 resource units. This treatment ranked 12th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 568. This treatment ranked 15th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 15th out of 32.

Comparing treatments, combination number 32 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

7.3.3 Treatment 23 (01101) Results

Treatment 23 is assigned the high signal values for all five variables in the designed experiment except the first (cost) and fourth (Ind. sweep) positions.

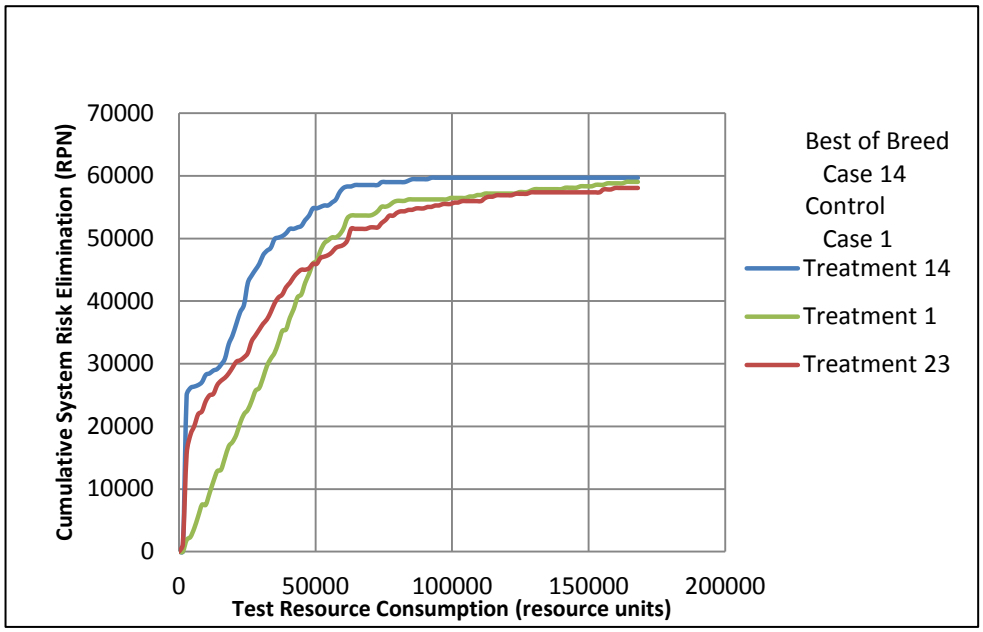


Figure 7.13: Risk mitigation comparison, treatments number 1, 14, and 23

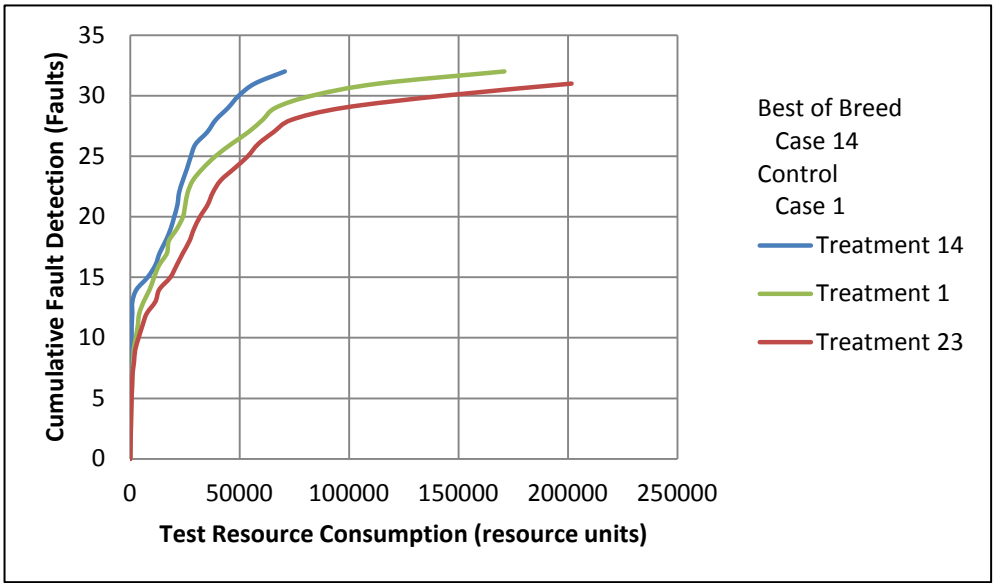


Figure 7.14: Cumulative fault detection comparison, treatments number 1, 14, and 23

Table 7.10: Compiled DOE results for treatment number 23

Treatment # 23									
Treatment Order 01101	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	569	418	570	608	486	574	564	626	552
Average Resource Count	28683	20990	28752	30466	24427	28821	28075	31158	27672
Resource Count at Last Fault	153792	79623	157920	121680	101664	126624	153936	174576	133727
No. of undetected faults	0	2	0	1	1	0	2	0	
Total Resource Consumption		175104		175056	175008		175056		
Total Test Case		3592		3571	3569		3572		
Adjusted Ave TC Count	569	661	570	723	605	574	797	626	641
Adjusted Ave Res Count	28683	32811	28752	36079	30226	28821	39450	31158	31997
Adjusted Res Count Last Fault	153792	245146	157920	210067	210010	126624	245078	174576	190402

Summary

After repeating the treatment eight times a total of 6 faults were left undetected in the simulation. The treatment ranked last out of the 32 combinations. The search process can become too aggressive.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 190402 resource units out of a possible 175000. This treatment ranked 32th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 31997 resource units. This treatment ranked 32th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 641. This treatment ranked 31st out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 32nd out of 32. The variable combination in this treatment is less effective than treatment number 1.

Comparing treatments, combination number 23 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

7.3.4 Treatment 2 (10000) Results

Treatment 2 is assigned the low signal values for all five variables in the designed experiment except the first position (cost).

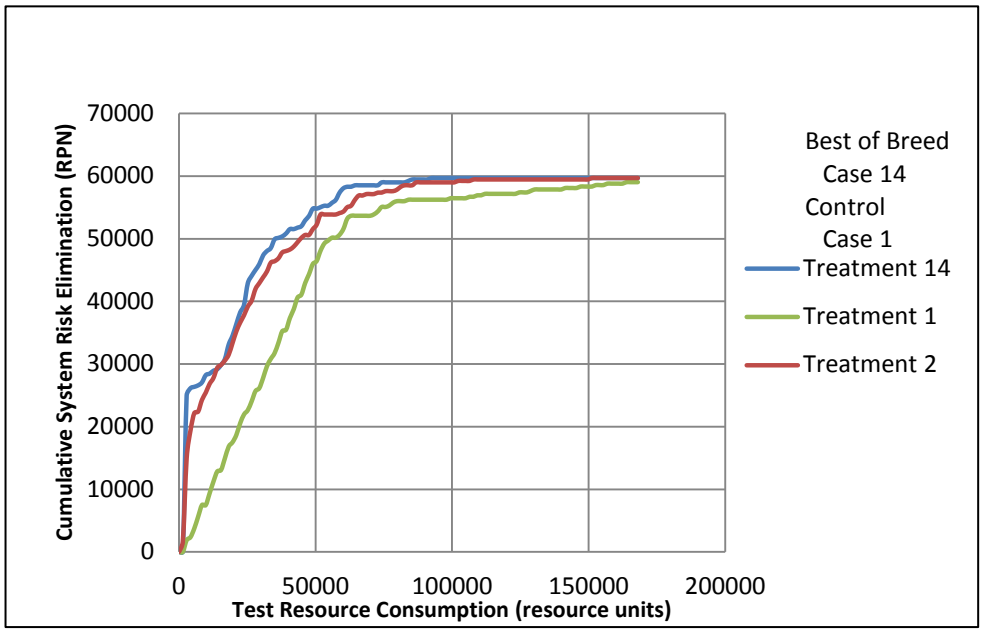


Figure 7.15: Risk mitigation comparison, treatments 1, 14, and 2

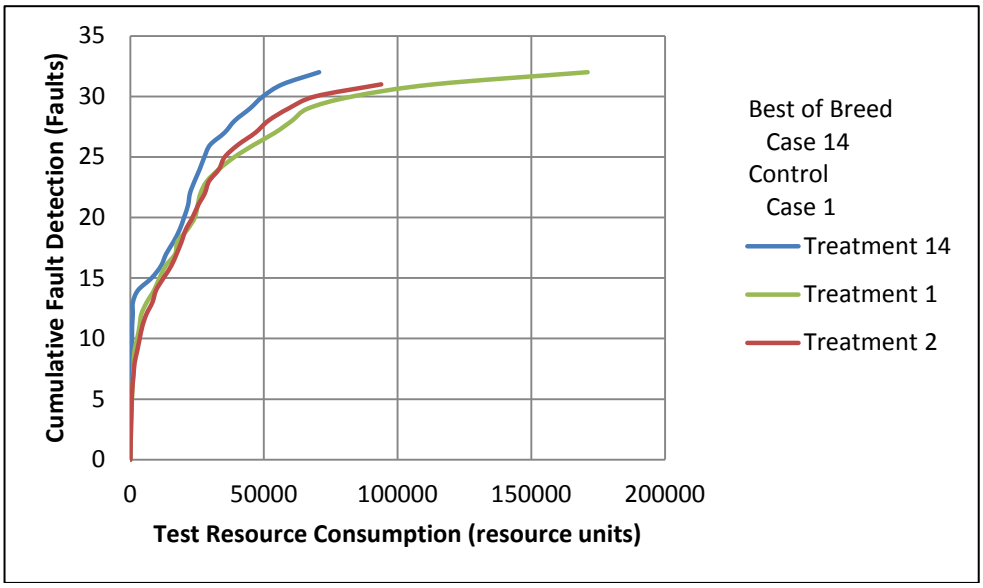


Figure 7.16: Cumulative fault detection comparison, treatments 1, 14, and 2

Table 7.11: Compiled DOE results for treatment 2

Treatment # 2									
Treatment Order 10000	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	482	485	515	526	461	498	533	487	498
Average Resource Count	19374	19688	20963	21344	1900	20418	21740	19886	18164
Resource Count at Last Fault	78651	149556	85067	99878	65871	84260	106249	80949	93810
No. of undetected faults	0	0	0	0	0	0	0	0	
Total Resource Consumption									
Total Test Case									
Adjusted Ave TC Count	482	485	515	526	461	498	533	487	498
Adjusted Ave Res Count	19374	19688	20963	21344	1900	20418	21740	19886	18164
Adjusted Res Count Last Fault	78651	149556	85067	99878	65871	84260	106249	80949	93810

Summary

After repeating the treatment eight times a total of 0 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 93810 resource units out of a possible 175000. This treatment ranked 8th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 18164 resource units. This treatment ranked 4th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 498. This treatment ranked 5th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 6th out of 32.

Comparing treatments, combination number 2 takes advantage of the most effective variable (cost) but is not as aggressive as number 14. The risk mitigation curve is very similar to the best of breed (BOB). Due to the slightly less aggressive search process, the average amount of resources consumed to detect all faults took approximately 40% more than the BOB. This treatment is consistently more effective than treatment 1.

7.3.5 Treatment 14 (10110) Results

Treatment 14 is assigned the high signal values for all five variables in the designed experiment except the second (risk priority) and fifth (GA-probability) positions.

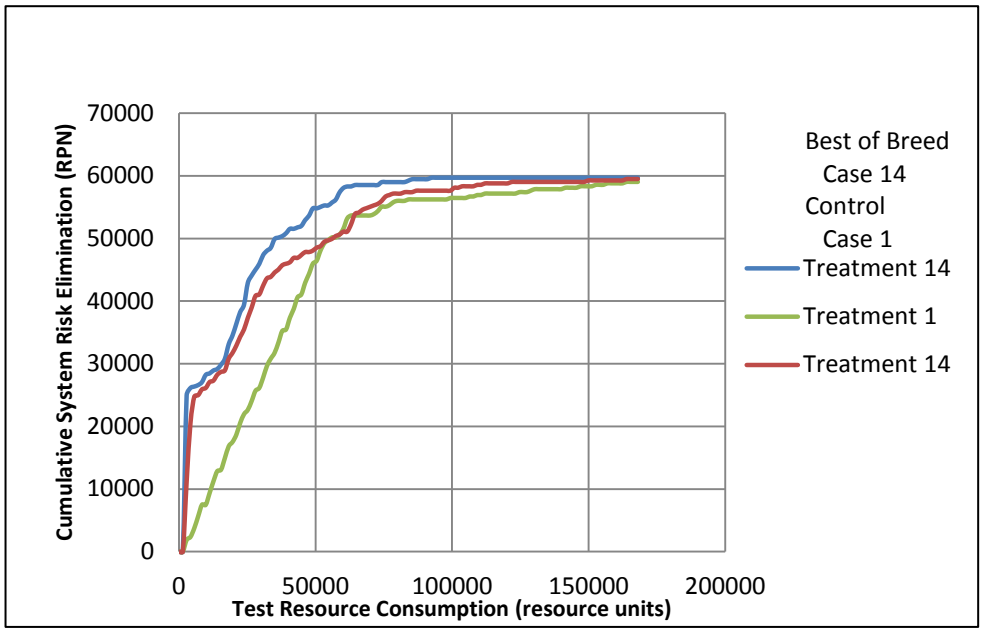


Figure 7.17: Risk mitigation comparison, treatments 1 and 14

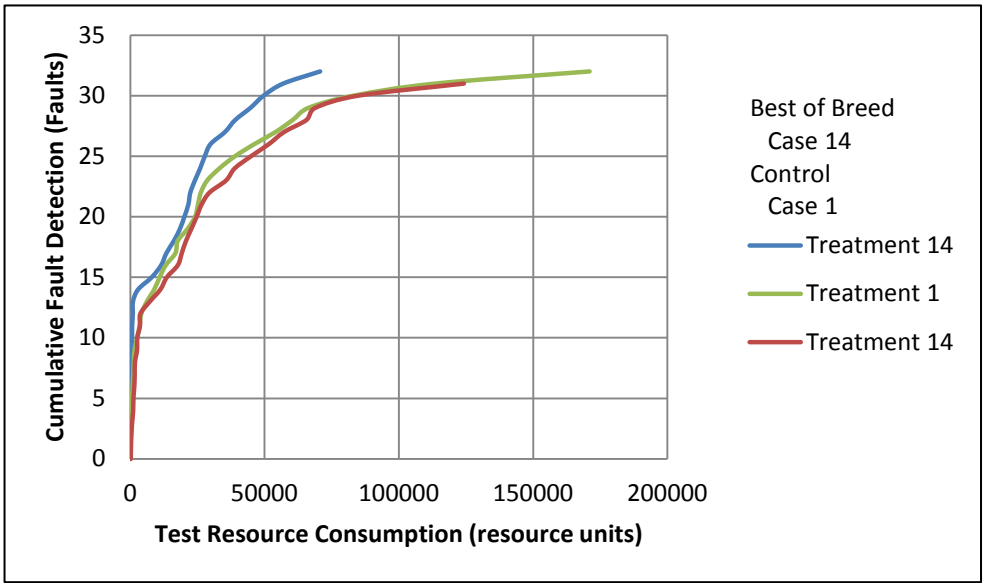


Figure 7.18: Cumulative fault detection comparison, treatments 1 and 14

Table 7.12: Compiled DOE results for treatment number 14

Treatment # 14									
Treatment Order 10110	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	470	450	532	501	435	379	449	475	461
Average Resource Count	17319	16457	19797	18611	16007	13650	16495	17464	16975
Resource Count at Last Fault	61795	57261	71882	81372	60104	58277	83910	58277	66610
No. of undetected faults	0	0	0	0	0	0	0	0	
Total Resource Consumption									
Total Test Case									
Adjusted Ave TC Count	470	450	532	501	435	379	449	475	461
Adjusted Ave Res Count	17319	16457	19797	18611	16007	13650	16495	17464	16975
Adjusted Res Count Last Fault	61795	57261	71882	81372	60104	58277	83910	58277	66610

Summary

After repeating the treatment eight times a total of 0 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 66610 resource units out of a possible 175000. This treatment ranked 1st out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 16975 resource units. This treatment ranked 1st out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 461. This treatment ranked 3rd out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 1st out of 32.

Comparing treatments, combination number 14 took advantage of early independent and dependent fault detection and demonstrated accelerated risk mitigation growth. In addition, treatment number 14 discovered all of the faults with approximately half the amount of resources required in treatment number 1.

7.4 Reliability Growth Curve – Discovery Zone Breakdown

In the product assurance process, the reliability growth curve is a standard metric used to tracked product assurance progress during the design process. As faults are discovered and mitigated over the development life-cycle, system reliability increases. One underlying premise of this dissertation's model is the timing of the discovery of the four main fault types. The model's fault discovery process sheds light on how the timing of these discoveries follow a distinct pattern in the normal life-cycle. The actual risk mitigation curve for treatment 14 (recognized as the most efficient algorithm) is presented in Figure 7.19.

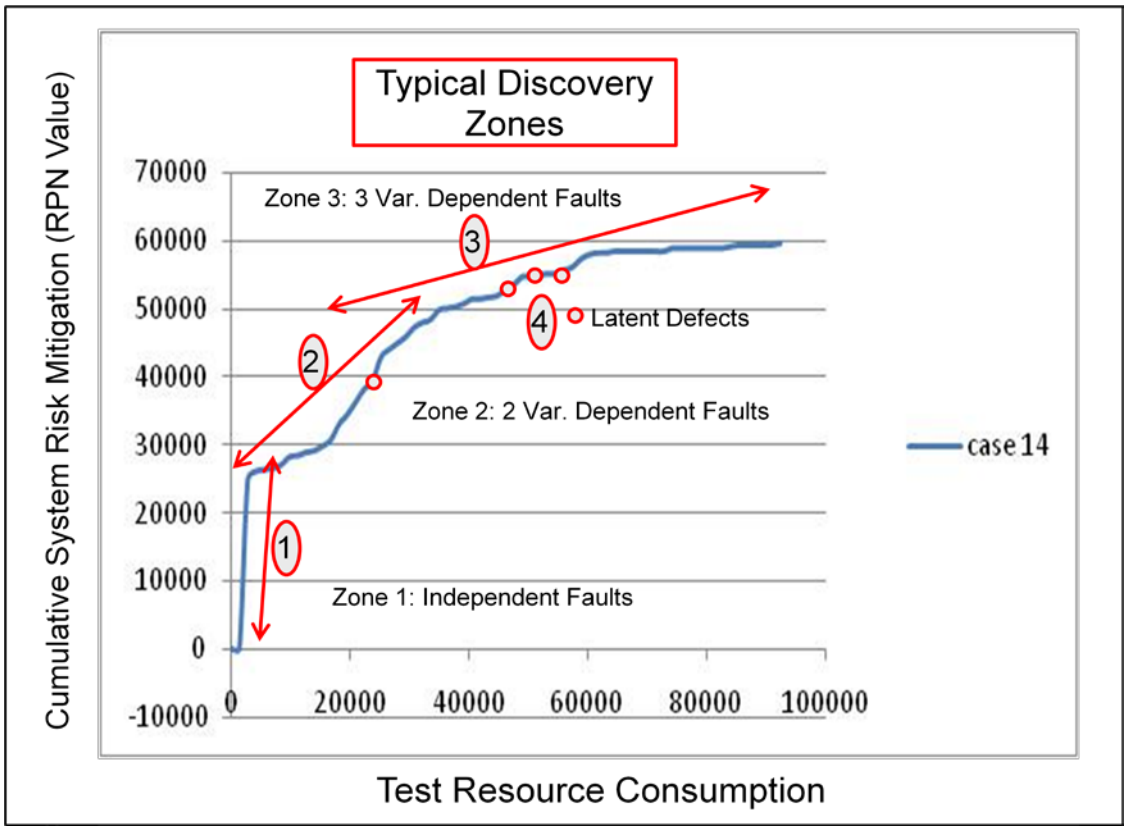


Figure 7.19: Major Zones of Complex Systems Reliability Growth Curve

The algorithm takes advantage of testing all possible independent sub-system variables before the adaptive genetic search algorithm takes place. This process flushes out any non-functioning independent Sub-systems and sub-system variables. In doing so, test resources are not consumed on a component that has already been deemed defective from previous testing. It was hypothesized that it is natural that the majority of two variable faults will be discovered before the majority of three variable faults. In part, the search algorithm, takes advantage of randomly generated test cases and, therefore, a three variable fault can be discovered at any time. It just takes more time to cover the potential number of possible combinations. It is also important to note that this algorithm does not guarantee that every test combination will be covered before the resources are consumed. It is especially true if the algorithm takes advantage of the actual variable costs instead of using the same cost for each variable to ensure total test case diversity. In the adaptive search algorithm presented in this dissertation, the model

was programmed to choose the lower cost test cases from a generated sample as programmed by the user.

Data identifying the point when the first fault of each type was identified and the point when the last fault was identified for each type is presented in Figure 7.13. In the algorithm, fault detection is tracked by the amount of resources that have been consumed at the point in time when the fault was detected. The actual fault type zones in treatment 14 follow the pattern expected when the last two-variable fault was identified well before the last three variable fault was located.

Table 7.13: Risk mitigation fault zone data

Major Fault Zones - Treatment 14		
Fault Type	Resource Count First Fault Detected	Resource Count Last Fault Detected
1 Variable - Independent	27	961
2 Variable - Dependent	5153	33528
3 Variable - Dependent	11389	62126
Latent Defect No. 1	46385	NA
Latent Defect No. 108	54443	NA
Latent Defect No. 109	24378	NA
Latent Defect No. 113	56009	NA

7.5 Hypotheses Analysis - Summary

The original hypotheses speculated that the combination of five identified independent variables in the adaptive genetic search algorithm would improve the fault detection and mitigation process as compared to simply relying on creating complex system test cases based on a random generation process. A two level - five variable full factorial experiment was designed to analyze the effectiveness of the potential models. The base case (treatment number 1 (00000)) was included in the designed experiment and ranked 25th out of the 32 treatment combinations. These results indicate that seven test variable combinations created search algorithms that decreased the search efficiency as

relative to the base case. The original hypotheses speculated that taking advantage of all five independent variables (treatment number 32 – (11111)) would improve the search efficiency relative to the base case. The final position of this treatment was 15th in the rank order, ten positions above the base case. Treatment number 14 (10110) was identified as the most efficient combination of the independent variables. Although the alpha level (α) was set to 0.1 due to the reality that embedded faults in complex systems are always different, a p-value equal to 0.000 was obtained in statistical analysis that compared the results of the control treatment vs. the most efficient treatment. Therefore, the null hypothesis is rejected and the treatment 14 combination of variables did significantly improve the search for embedded faults in the complex system simulator. Because the statistical confidence of the independent variables (along with the related interactions) in the analysis of the designed experiment differed depending on the targeted priority, one null hypothesis of the individual independent variable could not be rejected due to the lack of statistical evidence.

In general, the null hypotheses for the independent variables referred to as Common Cost and sub-system Test Sweep can be rejected. The action of using actual cost for all sub-system variable, as well as sweeping through all independent sub-system variables before engaging the genetic search algorithm, significantly improves the efficiency of the search algorithm compared to treatment number 1. There is also evidence that changing the probability of the chromosome Gene selection process, based on previous success, did significantly affect the search algorithm but in a negative way. The search algorithm became too greedy. The same was true for the act of prioritizing the allocation of fault mitigation resources to the discovered fault with the highest risk rating. Although these actions independently do not improve the search process, there is statistical evidence that the interactions of the variables could improve the results of the search algorithms depending on the results priority.

The results indicate that efficiency gains are possible in the fault search process which will enable improvement in the product assurance process. With improvements in the

fault detection and mitigation process, the driving metrics of the Half-Life Return Model are also improved. This includes the potential reduction of the development lifecycle cost and/or schedule. It can also directly affect the satisfaction of customers which can lead towards extending life of the product platform in the field. This combination of results is the foundation for Sustainable Value Creation.

Chapter 8: Conclusions and Future Work

This dissertation is focused on research in the field of sustainable product development. In free enterprise markets, producers seek to develop products that drive a profit for their respective business, as well as provide the best solution for the customer. In this process, a value proposition is developed by the producer for the consumer that is designed to overcome the risks of the business venture vs. the potential reward for both the producer as well as the consumer. Products and design platforms that are abandoned before the end of their useful life, create waste and reduce asset value for society and the environment, in addition to the producer and consumer. Design teams that fail to take a longer term perspective on the effect their product development process has on the overall product life in the field, miss the opportunity to improve the creation of sustainable value for their respective stakeholders. There is a need for research that improves the toolset for the engineering community that aids in the sustainable product design process.

8.1 Contributions of This Dissertation

Current literature and related development tools available to the engineering community often fail to assist the design team in bridging the gap between sustainable design metrics and financial success. This dissertation, presents a model that identifies the primary drivers which lead to closing the loop towards Sustainable Lifetime Value Creation. The problems addressed in this dissertation and the unique contributions are divided into two parts. The first section focuses on the integration and analysis of data sets from a more sustainable value proposition and product utilization. The Half-Life Return Model (HLRM) is presented, designed to provide feedback to producers in the pursuit of improving the return on investment for the primary stakeholders. Metrics are identified in the model, designed to aid the development team in analyzing the financial success of the product relative to the product half-life in the field.

The second section applies the concepts presented in the first section with focus placed on the effects specific feedback variables have on the efficiency of the product development process. An Adaptive Genetic Search Algorithm is presented, designed to improve fault detection and mitigation during the product delivery process. A computer simulation is used to study the effectiveness of the primary aspects introduced in the search algorithm, in order to attempt to improve the reliability growth of the system during the development life-cycle.

In summary, the main contributions of this dissertation are as follows:

Sustainable Value Proposition

- Reformulated the concept of a value proposition between producers and consumers to reflect the additional sustainability focus areas of society and the environment. This new Sustainable Value Proposition is designed to compare relative design concepts, in order to drive sustainable improvements in next generation products.

  The sustainable products value proposition seeks a balanced approach towards the integration of total cost of ownership, social and environmental improvements, and an expanded definition of product life drivers. The driving metrics identified in three impact areas are focused on reducing the potential risk of relative product offerings. In the development process, engineers need to not only look at the total cost for the consumer, but also take a broader and more holistic cost view, in order to identify product designs concepts that may be at higher risk for long-term sustainability and waste streams.

- Defined six driving cost aspects for the producer in the Sustainable Value Proposition. This includes measuring the commonality and convertibility cost opportunities of design concepts from a platform to platform as well as gen-to-gen perspective. In essence expanding the definition of the total cost of product development.

- Defined six driving aspects for the consumer in the Sustainable Value Proposition. This includes expanding the definition of cost of ownership beyond the initial purchase and operation of the product, to include the total cost to the consumer over the lifecycle and potentially multiple lifecycles of the consumers needs.

- Defined six driving aspects of the product design from a societal and environmental perspective. This will close the loop between the consumer and producer in the Sustainable Value Proposition.

Half-Life Return Model

- Designed and presented the Half-Life Return Model. Producers gain insight into the goal of Sustainable Value Creation by integrating data from the product profit and loss curve on top of the data from product half-life tracking.

- Defined the drivers of the product delivery process that will improve the financial return on investment, for the development team in the Half-Life Return Model.

Product Assurance Model: Adaptive Genetic Search Algorithm

- Defined a detailed list of fault types discovered in the verification of complex systems in order to improve the fault detection and mitigation model

- Developed an improved product assurance feedback loop model, by Integrating the process of fault detection and mitigation along with product risk management into one system.

- With the goal of improving the velocity of quick learning (cycles) between the product design and system verification teams, this model integrates risk management and resource consumption into the product assurance process.

- Developed an Adaptive Genetic Search Algorithm designed to improve fault detection and mitigation during the product delivery process. A computer simulation was used to study the effectiveness of primary aspects introduced in the search algorithm, in order to attempt to improve the reliability growth of the system during the development life-cycle.

- The results of the experiment designed to validate the search algorithm, confirm some of the hypotheses, but shed light on the sensitivity of overly aggressive product validation strategies.

    - In the case study, the most efficient combination of variables in the adaptive genetic search algorithm improved the fault detection efficiency by 44%, relative to the control treatment.

    - The use of the test case generation process that takes advantage of cost benefits between potential (competing) samples from the test case population, has the greatest efficiency effect on the improvement of the fault detection process.

    - The results of the case study also confirm the benefits of early fault detection as well as test case diversity in the overall efficiency a product verification strategy.

    - Depending on the specific complex system to be verified, the search algorithm can result in interactions between the independent variables. The results of the experiment show the potential benefits of creating child test cases from previously successful test cases. At the same time, the results also show that the modification of the chromosome gene selection probability based on previous success, created a test case generation strategy that became too aggressive.

- As a result of improving the fault detection and mitigation process during the design lifecycle, the improvement of several key metrics in the Half-Life Return

Model are enabled. These include shorter verification cycles and/or the ability to increase the utilization of your test resources. With increased verification throughput, product quality and customer satisfaction increase. Finally, the net result is an increase in Sustainable Lifetime Value.

8.2 Future Work

Further research on the sensitivity of some of the variables held constant, may increase the knowledge of feedback in the adaptive search process. Beyond test case generation, the use of risk, cost, system coverage, and feedback in an adaptive search can be applied to many other applications that seek multiple value targets.

Whereas this research is focused on the process of developing sustainable product and processes in high technology industries, the results can be applied to other fields. The first part of this dissertation can be applied to any producer who seeks to drive additional shareholder value and is faced with a dynamic market. Future research focused on the sensitivity of the metrics identified in the Half-Life Return Model will improve the ability to apply these tools in other industries. This includes the potential validation of the model with field data, comparing Half-Life Return Model results to the producers shareholder return on investment. In addition, it would be useful to continue Half-Life Return Model research based on the effects that external factors (non-controllable) have on the model when comparing different industry types.

Educating development communities about the aspects of value creation from a sustainability perspective is an important next step from this research. By taking a broader and more holistic approach to value creation during the product development process, an improved perspective can be achieved regarding risk management from a shareholder return on investment.

The race continues between the e-gain benefits of new technology and the research for new tools that will aid in the long-term development of more sustainable products and

processes. A central goal of this research is to begin to build a new paradigm for development engineers. It can be a paradigm that sheds light on the realization that product designs can be more sustainable from both financial and environmental perspectives. By focusing on the main drivers of each sustainable value proposition aspect, the development community improves their role in creating truly sustainable value.

Appendix

Treatment 3 (01000) Results

Treatment 3 is assigned the low signal values for all five variables in the designed experiment except the second position (risk prioritization).

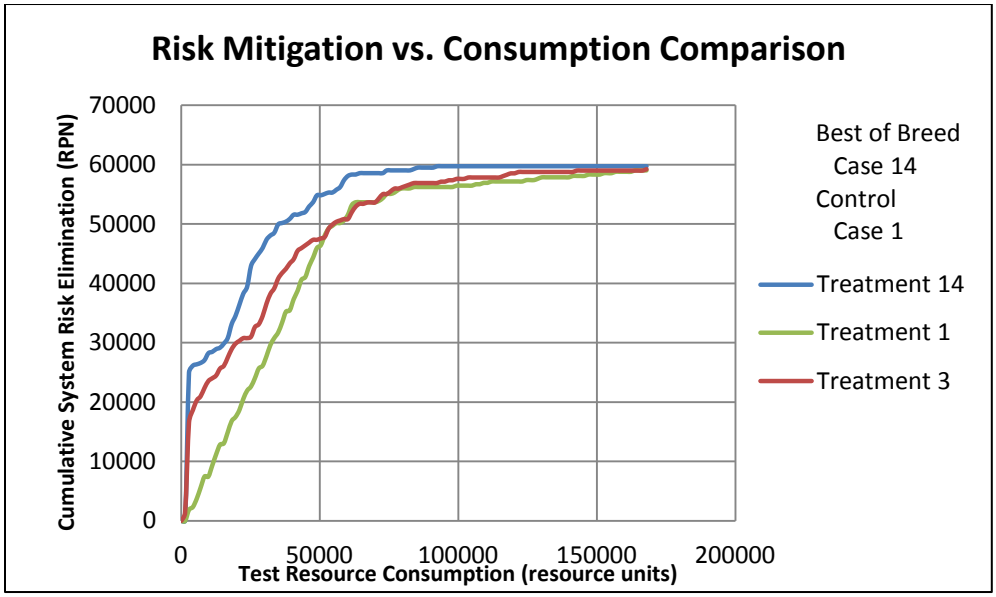


Figure A.1: Risk mitigation comparison, treatments 1, 14, and 3

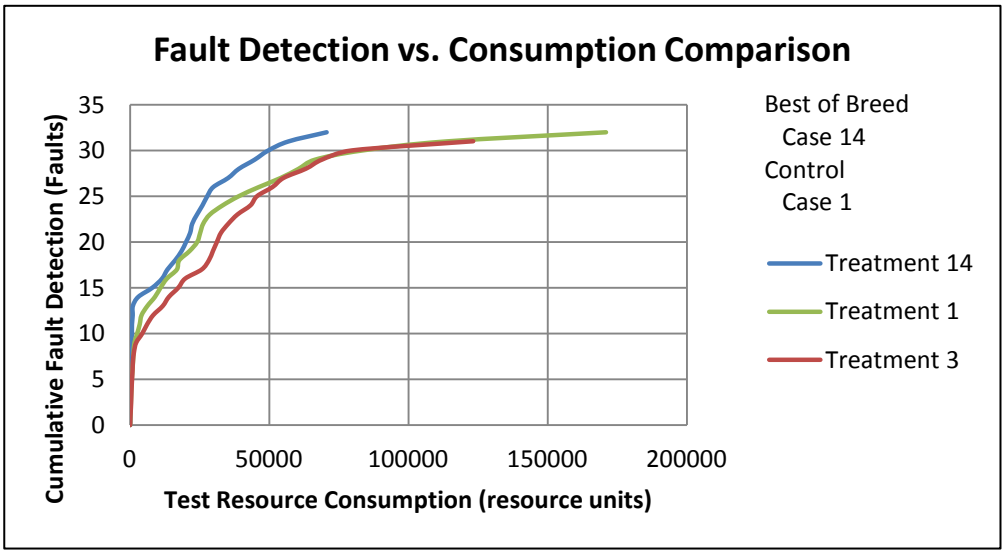


Figure A.2: Cumulative fault detection comparison, treatments 1, 14, and 3

Table A.1: Compiled DOE results for treatment number 3

Treatment # 3									
Treatment Order 01000	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	623	541	589	493	584	475	476	451	529
Average Resource Count	30969	27000	29382	26500	29061	23842	23890	22911	26694
Resource Count at Last Fault	166032	119328	140880	115200	172848	114768	97776	101760	128574
No. of undetected faults	1	0	0	0	0	0	0	0	
Total Resource Consumption	175056								
Total Test Case	3696								
Adjusted Ave TC Count	742	541	589	493	584	475	476	451	544
Adjusted Ave Res Count	36566	27000	29382	26500	29061	23842	23890	22911	27394
Adjusted Res Count Last Fault	210067	119328	140880	115200	172848	114768	97776	101760	134078

Summary

After repeating the treatment eight times a total of 1 fault was left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 134078 resource units out of a possible 175000. This treatment ranked 16th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 27394 resource units. This treatment ranked 17th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 544. This treatment ranked 14th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 16th out of 32.

Comparing treatments, combination number 3 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 4 (11000) Results

Treatment 4 is assigned the low signal values for all five variables in the designed experiment except the first (cost) and second (risk prioritization) positions.

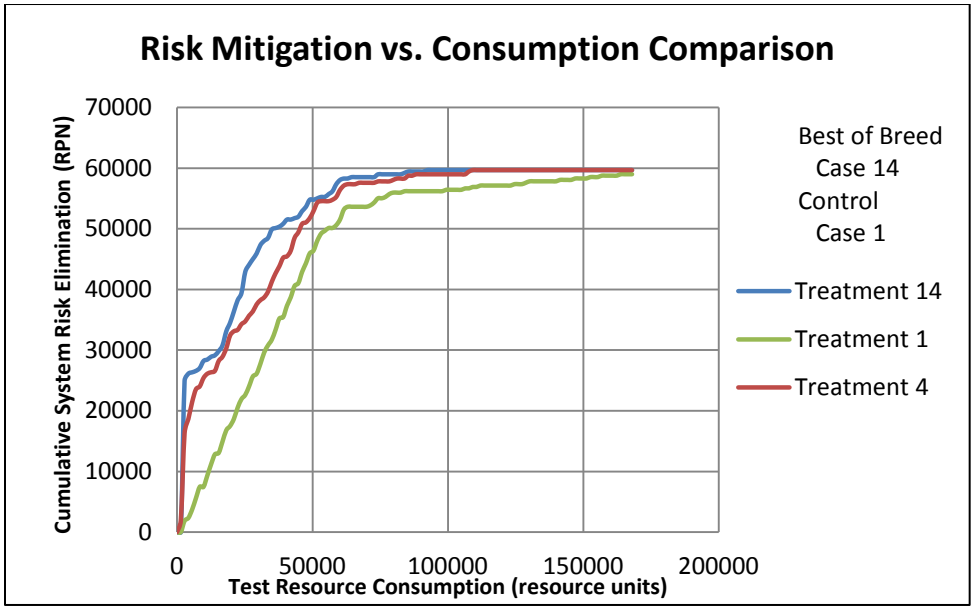


Figure A.3: Risk mitigation comparison, treatments 1, 14, and 4

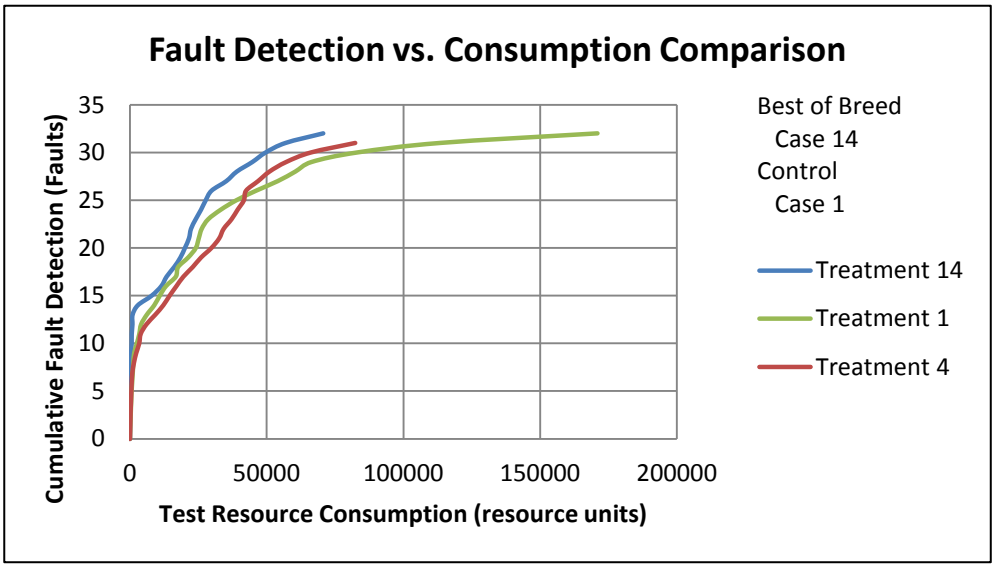


Figure A.4: Cumulative fault detection comparison, treatments 1, 14, and 4

Table A.2: Compiled DOE results for treatment 4

Treatment # 4									
Treatment Order 11000	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	565	496	489	518	574	534	525	579	535
Average Resource Count	22792	20234	20199	20979	23238	21819	21403	23643	21788
Resource Count at Last Fault	106089	58237	71724	64623	105730	86158	59392	106863	82352
No. of undetected faults	0	0	0	0	0	0	0	0	
Total Resource Consumption									
Total Test Case									
Adjusted Ave TC Count	565	496	489	518	574	534	525	579	535
Adjusted Ave Res Count	22792	20234	20199	20979	23238	21819	21403	23643	21788
Adjusted Res Count Last Fault	106089	58237	71724	64623	105730	86158	59392	106863	82352

Summary

After repeating the treatment eight times a total of 0 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 82352 resource units out of a possible 175000. This treatment ranked 6th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 21788 resource units. This treatment ranked 8th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 535. This treatment ranked 8th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 7th out of 32.

Similar to combination number 2, treatment 4 takes advantage of the most took effective variable (cost) but is not as aggressive as number 14. The fault detection process is actually less effective than both treatments 1 and 14 in the early stages but catches up quickly after the middle stages. In the end, the average amount of resources consumed to detect all faults took approximately 25% more than the best of breed.

Treatment 5 (00100) Results

Treatment 5 is assigned the low signal values for all five variables in the designed experiment except the third (GA-crossover) positions.

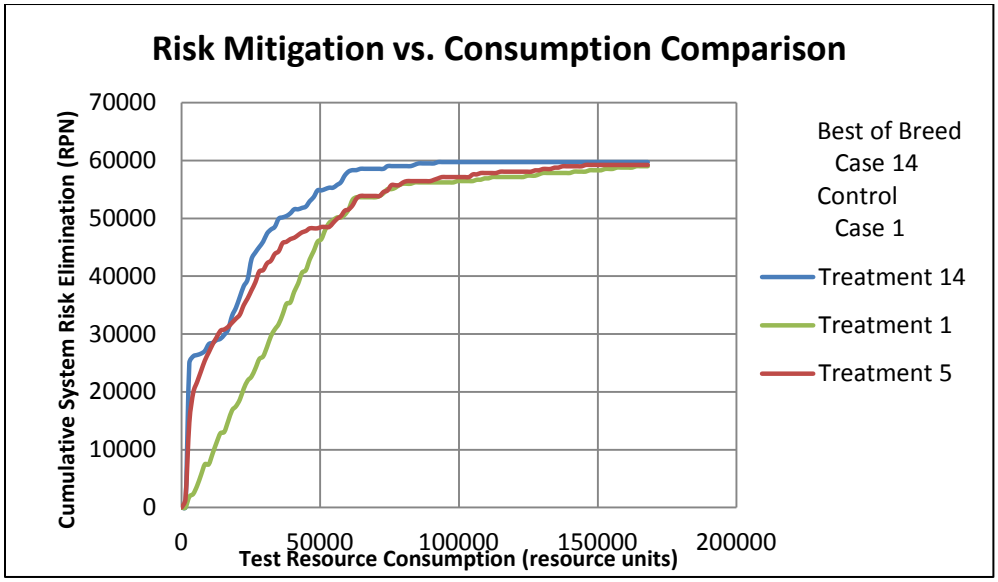


Figure A.5: Risk mitigation comparison, treatments 1, 14, and 5

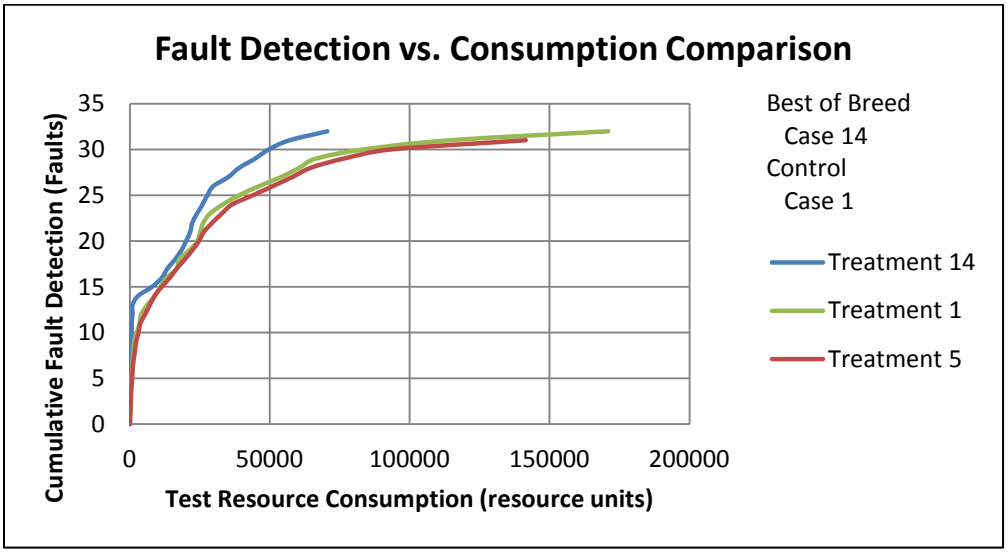


Figure A.6: Cumulative fault detection comparison, treatments 1, 14, and 5

Table A.3: Compiled DOE results for treatment number 5

Treatment # 5									
Treatment Order 00100	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	506	517	463	429	415	513	447	508	475
Average Resource Count	25652	26182	23588	21809	21326	25752	22802	25761	24109
Resource Count at Last Fault	103008	132048	143232	103392	127824	167760	112656	135264	128148
No. of undetected faults	0	0	1	0	0	0	0	0	
Total Resource Consumption	0		175104						
Total Test Case	0		3696						
Adjusted Ave TC Count	506	517	587	429	415	513	447	508	490
Adjusted Ave Res Count	25652	26182	29417	21809	21326	25752	22802	25761	24838
Adjusted Res Count Last Fault	0	132048	210125	103392	127824	167760	112656	135264	123634

Summary

After repeating the treatment eight times a total of 1 fault was left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 123634 resource units out of a possible 175000. This treatment ranked 17th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 24838 resource units. This treatment ranked 16th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 490. This treatment ranked 13th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 17th out of 32.

Comparing treatments, combination number 5 is similar to treatment 3 which shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages. Although this treatment does track the same fault detection progress and treatment 1, it does provide earlier risk mitigation growth.

Treatment 6 (10100) Results

Treatment 6 is assigned the low signal values for all five variables in the designed experiment except the first (cost) and third (GA-crossover) positions.

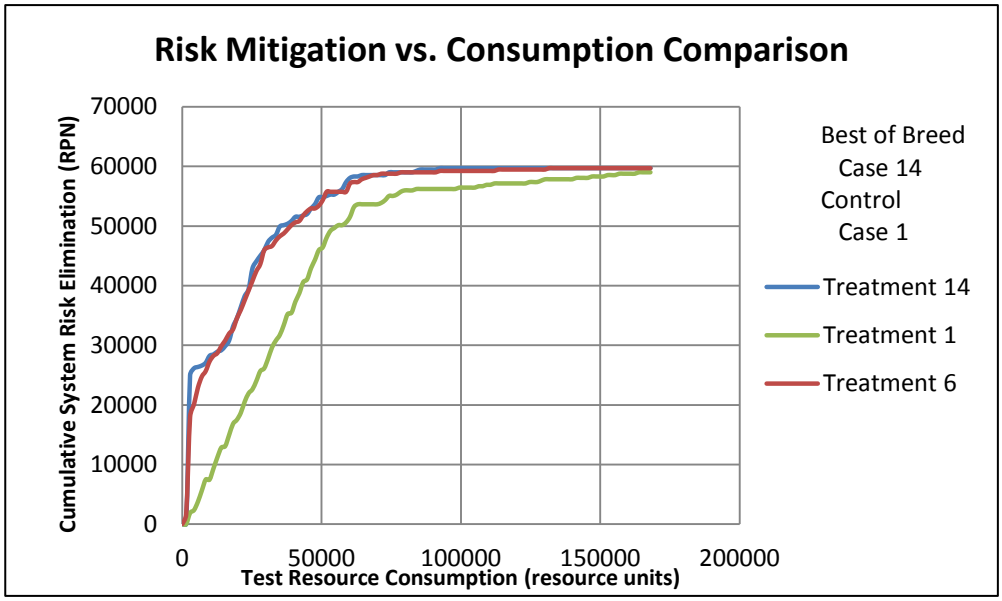


Figure A.7: Risk mitigation comparison, treatments 1, 14, and 6

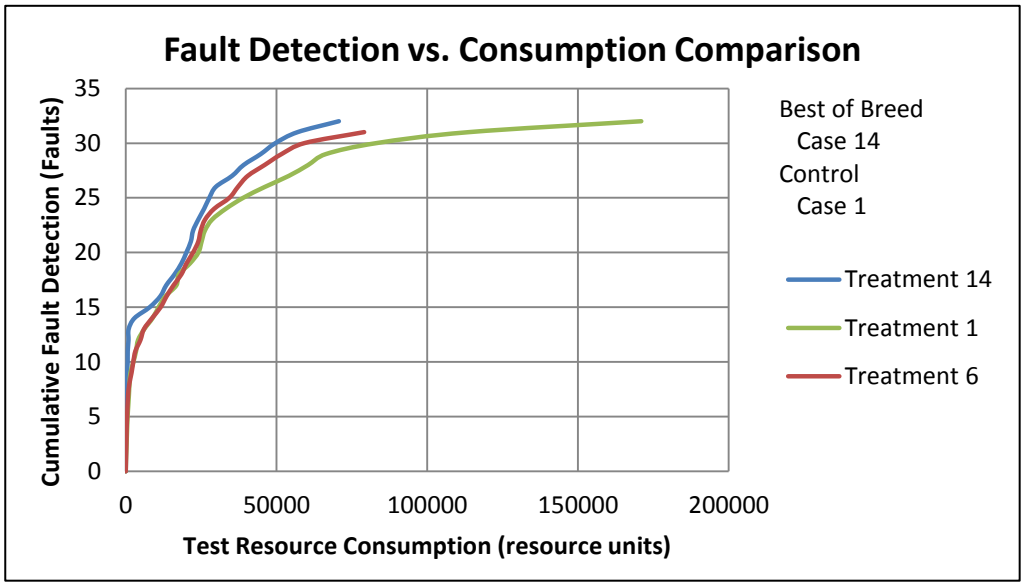


Figure A.8: Cumulative fault detection comparison, treatments 1, 14, and 6

Table A.4: Compiled DOE results for treatment number 6

Treatment # 6									
Treatment Order 10100	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	394	398	553	464	417	444	360	417	431
Average Resource Count	16359	16355	22538	19288	17155	18216	14920	17328	17770
Resource Count at Last Fault	65455	57907	128802	82832	69315	110811	57455	58798	78922
No. of undetected faults	0	0	0	0	0	0	0	0	
Total Resource Consumption									
Total Test Case									
Adjusted Ave TC Count	394	398	553	464	417	444	360	417	431
Adjusted Ave Res Count	16359	16355	22538	19288	17155	18216	14920	17328	17770
Adjusted Res Count Last Fault	65455	57907	128802	82832	69315	110811	57455	57455	78754

Summary

After repeating the treatment eight times a total of 0 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 78754 resource units out of a possible 175000. This treatment ranked 5th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 17770 resource units. This treatment ranked 3th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 431. This treatment ranked 1st out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 4th out of 32.

Comparing treatments, combination number 6 is a very effective treatment compared to treatment number 14. Although this treatment ranked fourth overall on the weighted scale, it was the third most efficient in average resource consumption and the most effective in the amount of test cases required to discover the faults.

Treatment 7 (01100) Results

Treatment 7 is assigned the low signal values for all five variables in the designed experiment except the second (risk prioritization) and third (GA-crossover) positions.

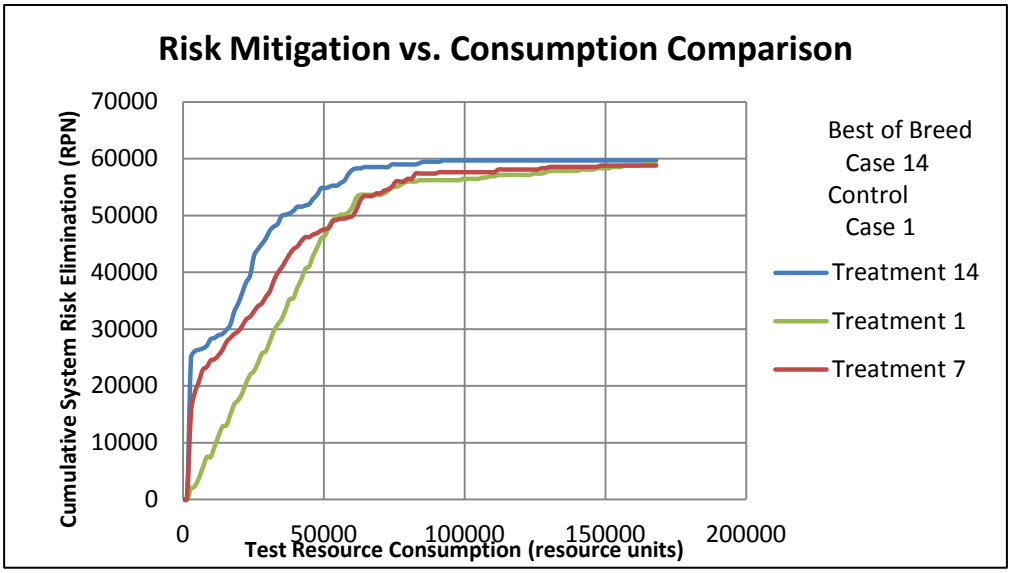


Figure A.9: Risk mitigation comparison, treatments 1, 14, and 7

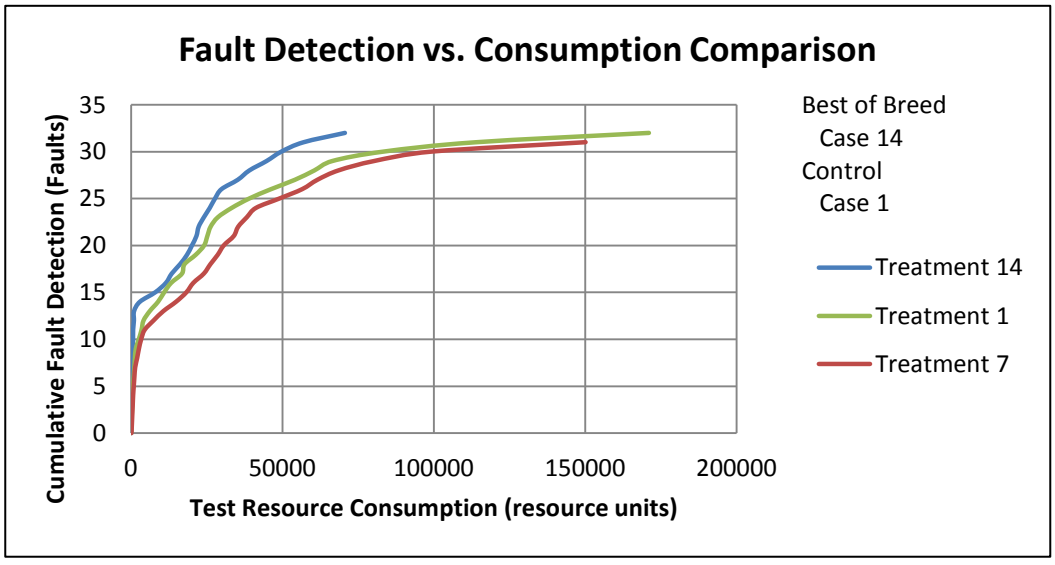


Figure A.10: Cumulative fault detection comparison, treatments 1, 14, and 7

Table A.5: Compiled DOE results for treatment number 7

Treatment # 7									
Treatment Order 01100	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	476	411	498	519	616	520	550	574	521
Average Resource Count	23996	20714	25170	26025	30626	26229	27482	28668	26114
Resource Count at Last Fault	125616	73248	72336	88416	172704	81120	146160	170640	116280
No. of undetected faults	0	0	0	0	1	1	0	0	
Total Resource Consumption					175104	175008			
Total Test Case					3648	3696			
Adjusted Ave TC Count	476	411	498	519	734	642	550	574	550
Adjusted Ave Res Count	23996	20714	25170	26025	36235	31972	27482	28668	27533
Adjusted Res Count Last Fault	125616	73248	72336	88416	210125	210010	146160	170640	137069

Summary

After repeating the treatment eight times a total of 2 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 137069 resource units out of a possible 175000. This treatment ranked 20th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 27533 resource units. This treatment ranked 22th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 550. This treatment ranked 18th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 20th out of 32.

Comparing treatments, combination number 7 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 8 (11100) Results

Treatment 8 is assigned the high signal values for all five variables in the designed experiment except the forth (Ind. sweep) and third (GA-probability) positions.

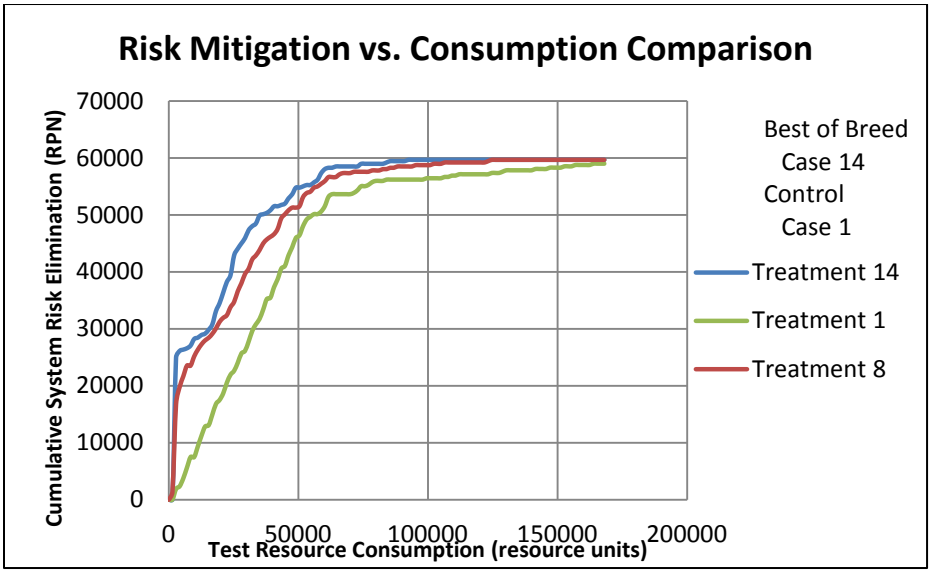


Figure A.11: Risk mitigation comparison, treatments 1, 14, and 8

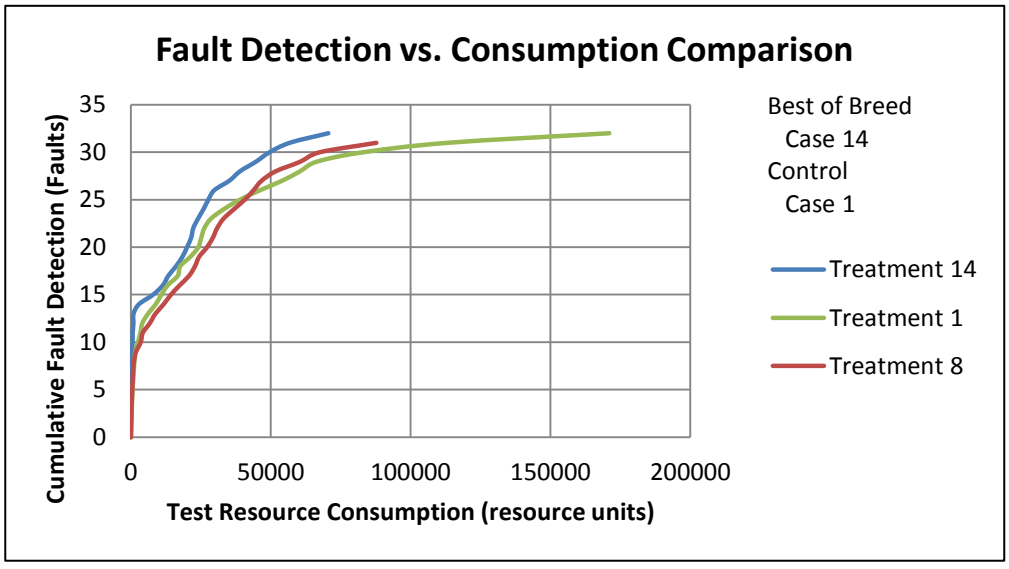


Figure A.12: Cumulative fault detection comparison, treatments 1, 14, and 8

Table A.6: Compiled DOE results for treatment number

Treatment # 8									
Treatment Order 11100	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	519	467	548	404	546	529	570	641	528
Average Resource Count	21052	18994	22187	16536	22612	21738	23219	26207	21568
Resource Count at Last Fault	63675	76056	82953	57582	85488	92488	121391	122164	87725
No. of undetected faults	0	0	0	0	0	0	0	0	
Total Resource Consumption									
Total Test Case									
Adjusted Ave TC Count	519	467	548	404	546	529	570	641	528
Adjusted Ave Res Count	21052	18994	22187	16536	22612	21738	23219	26207	21568
Adjusted Res Count Last Fault	63675	76056	82953	57582	85488	92488	121391	122164	87725

Summary

After repeating the treatment eight times a total of 0 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 87825 resource units out of a possible 175000. This treatment ranked 7th out of the 32 treatments in this category. The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 21568 resource units. This treatment ranked 7th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 528. This treatment ranked 7th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 8th out of 32.

Comparing treatments, combination number 8 takes advantage of the most took effective variable (cost) but is not as aggressive as number 14. The risk mitigation curve is very similar to the best of breed (BOB). Due to the slightly less aggressive search process, the average amount of resources consumed to detect all faults took approximately 30% more than the BOB. This treatment is consistently more effective than treatment 1.

Treatment 9 (00010) Results

Treatment 9 is assigned the low signal values for all five variables in the designed experiment except the forth (Ind. sweep) position.

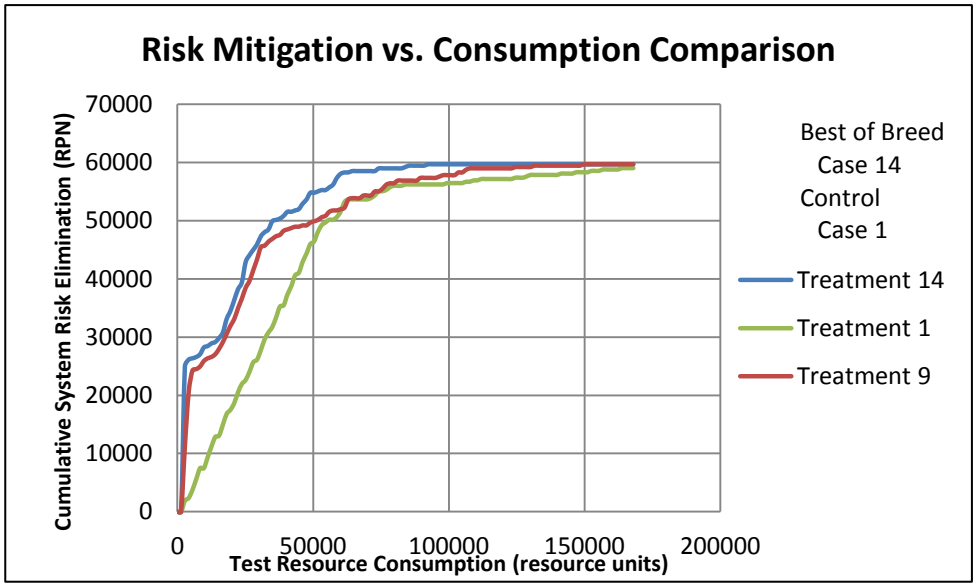


Figure A.13: Risk mitigation comparison, treatments 1, 14, and 9

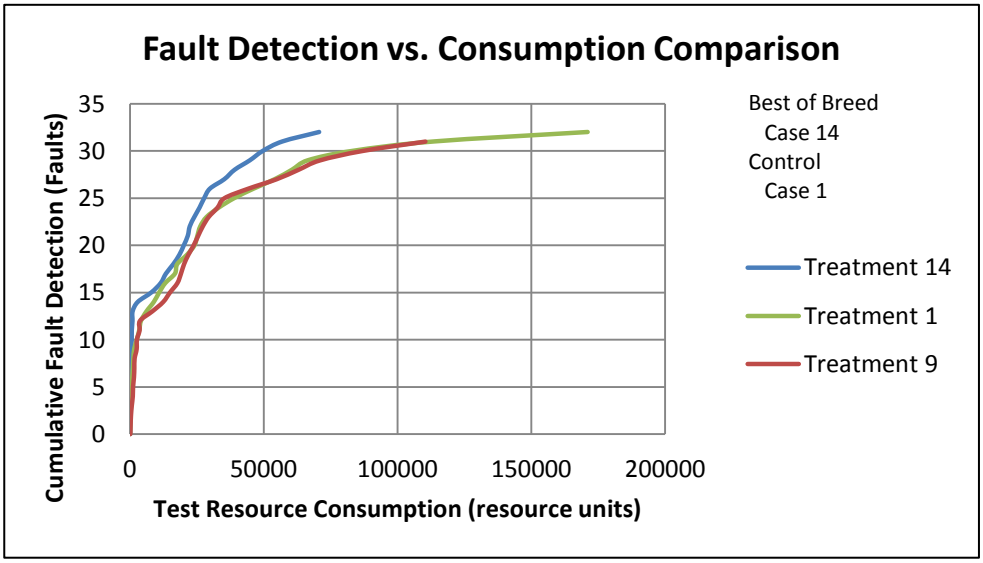


Figure A.14: Cumulative fault detection comparison, treatments 1, 14, and 9

Table A.7: Compiled DOE results for treatment number 9

Treatment # 9									
Treatment Order 00010	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	489	481	485	441	491	413	396	459	457
Average Resource Count	24896	24312	24354	22208	24704	21070	20157	23106	23101
Resource Count at Last Fault	147744	105024	100992	129264	122688	95712	76608	105216	110406
No. of undetected faults	0	0	0	0	0	0	0	0	
Total Resource Consumption									
Total Test Case									
Adjusted Ave TC Count	489	481	485	441	491	413	396	459	457
Adjusted Ave Res Count	24896	24312	24354	22208	24704	21070	20157	23106	23101
Adjusted Res Count Last Fault	147744	105024	100992	129264	122688	95712	76608	105216	110406

Summary

After repeating the treatment eight times a total of 0 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 110406 resource units out of a possible 175000. This treatment ranked 10th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 23101 resource units. This treatment ranked 6th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 457. This treatment ranked 2th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 9th out of 32.

This is a unique combination of variable because it ranks second in test case efficiency and sixth in the average amount of resource consumed but tenth in the amount of resources to discover the last fault. This is an efficient algorithm combination but runs a higher risk of consuming the resources before all faults are detected than the top five.

Treatment 10 (10010) Results

Treatment 10 is assigned the low signal values for all five variables in the designed experiment except the first (cost) and forth (Ind. sweep) positions.

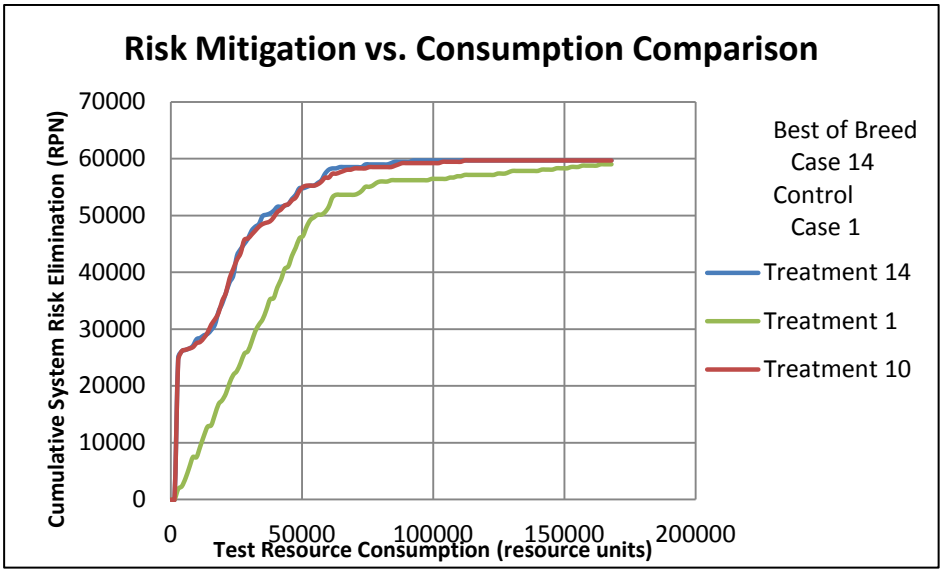


Figure A.15: Risk mitigation comparison, treatments 1, 14, and 10

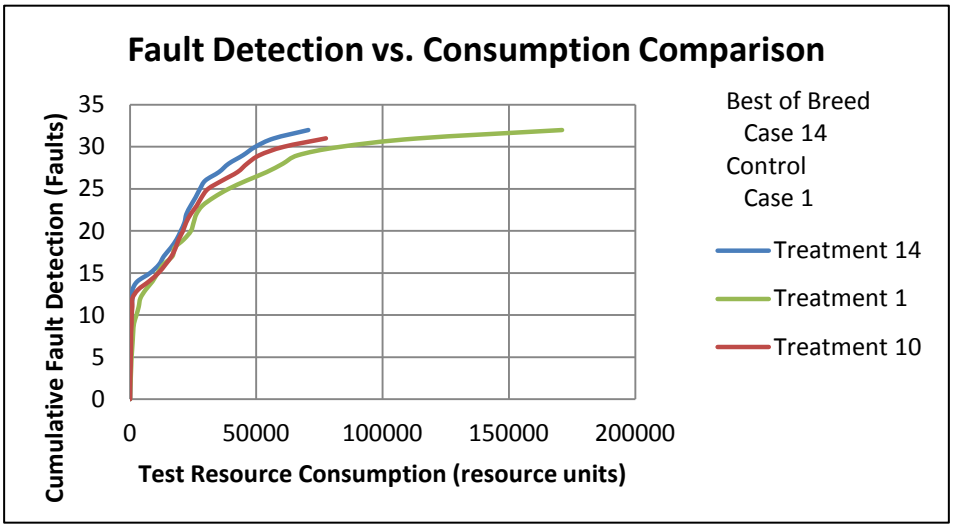


Figure A.16: Cumulative fault detection comparison, treatments 1, 14, and 10

Table A.8: Compiled DOE results for treatment number 10

Treatment # 10									
Treatment Order 10010	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	450	412	523	438	483	399	513	550	471
Average Resource Count	16817	15173	19614	16416	18039	14771	19203	20725	17595
Resource Count at Last Fault	83908	56855	84746	59145	67879	56315	101884	109462	77524
No. of undetected faults	0	0	0	0	0	0	0	0	
Total Resource Consumption									
Total Test Case									
Adjusted Ave TC Count	450	412	523	438	483	399	513	550	471
Adjusted Ave Res Count	16817	15173	19614	16416	18039	14771	19203	20725	17595
Adjusted Res Count Last Fault	83908	56855	84746	59145	67879	56315	101884	109462	77524

Summary

After repeating the treatment eight times a total of 0 faults were left undetected in the simulation. This treatment is very similar to treatment number 14, the best of breed.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 77524 resource units out of a possible 175000. This treatment ranked 4th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 17595 resource units. This treatment ranked 2th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 471. This treatment ranked 4th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 3rd out of 32.

Comparing treatments, combination number 10 is a very effective treatment compared to treatment number 14. On average, this treatment consumed approximately 15% more resources to detect the faults in the simulator.

Treatment 11 (01010) Results

Treatment 11 is assigned the low signal values for all five variables in the designed experiment except the second (risk prioritization) and forth (Ind. sweep) positions.

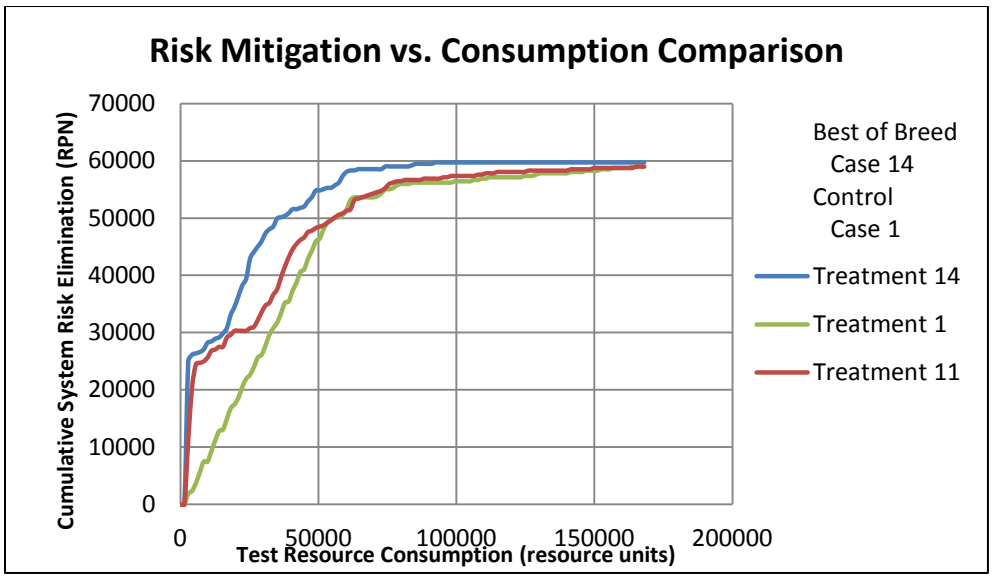


Figure A.17: Risk mitigation comparison, treatments 1, 14, and 11

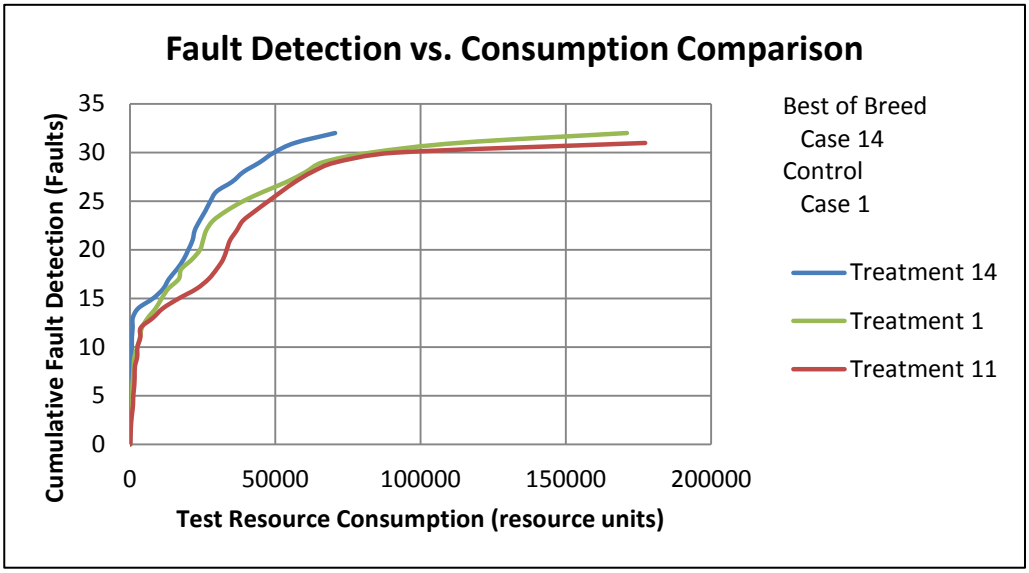


Figure A.18: Cumulative fault detection comparison, treatments 1, 14, and 11

Table A.9: Compiled DOE results for treatment number 11

Treatment # 11									
Treatment Order 01010	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	470	543	521	427	618	492	484	577	517
Average Resource Count	23812	27250	26061	21439	30848	24635	24341	28809	25899
Resource Count at Last Fault	139440	105504	112272	70272	147744	108864	73728	163728	115194
No. of undetected faults	0	0	0	1	0	1	1	0	
Total Resource Consumption				175104		175056	175104		
Total Test Case				3590		3696	3648		
Adjusted Ave TC Count	470	543	521	548	618	615	606	577	562
Adjusted Ave Res Count	23812	27250	26061	27335	30848	30430	30147	28809	28086
Adjusted Res Count Last Fault	139440	105504	112272	210125	147744	210067	210125	163728	162376

Summary

After repeating the treatment eight times a total of 3 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 162376 resource units out of a possible 175000. This treatment ranked 26th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 28086 resource units. This treatment ranked 26th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 562. This treatment ranked 25th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 26th out of 32.

Comparing treatments, combination number 3 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 12 (11010) Results

Treatment 12 is assigned the high signal values for all five variables in the designed experiment except the third (GA-crossover) and fifth (GA-probability) positions.

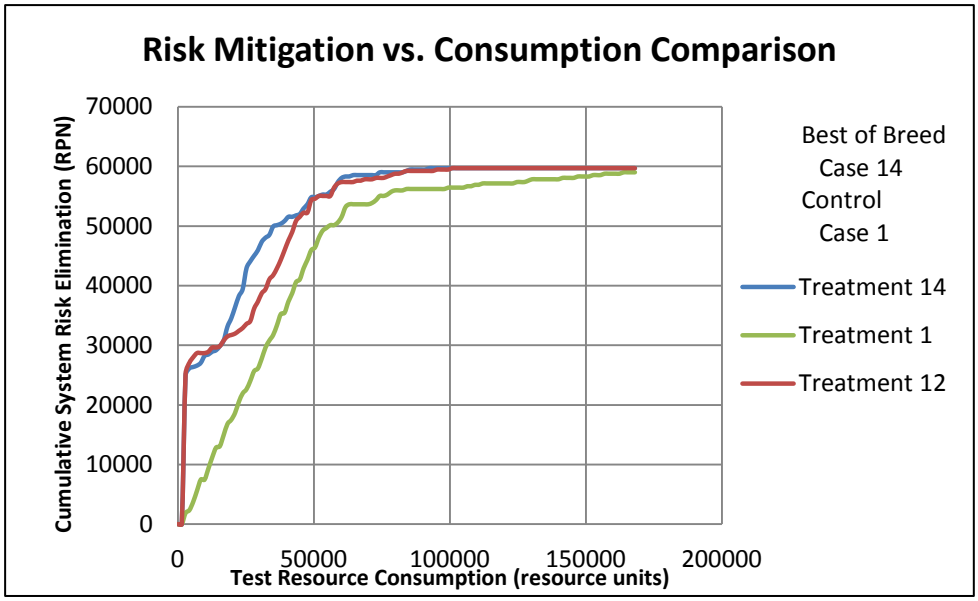


Figure A.19: Risk mitigation comparison, treatments 1, 14, and 12

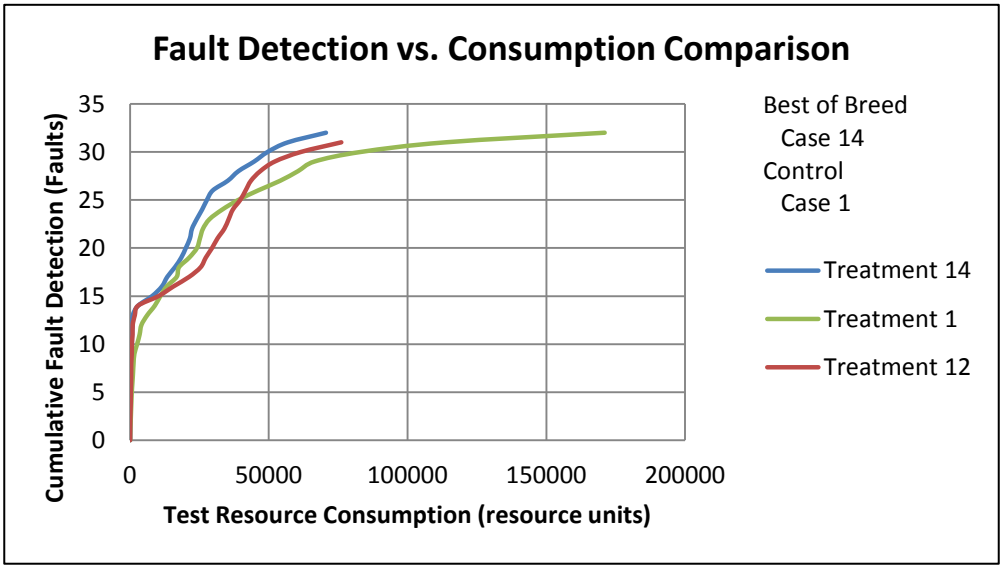


Figure A.20: Cumulative fault detection comparison, treatments 1, 14, and 12

Table A.10: Compiled DOE results for treatment number 12

Treatment # 12									
Treatment Order 11010	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	575	492	504	582	467	465	584	543	527
Average Resource Count	21883	18770	18924	22076	17731	17612	22422	20426	19981
Resource Count at Last Fault	80031	71407	93839	77626	55606	57257	98239	75333	76167
No. of undetected faults	0	0	0	0	0	0	0	0	
Total Resource Consumption									
Total Test Case									
Adjusted Ave TC Count	575	492	504	582	467	465	584	543	527
Adjusted Ave Res Count	21883	18770	18924	22076	17731	17612	22422	20426	19981
Adjusted Res Count Last Fault	80031	71407	93839	77626	55606	57257	98239	75333	76167

Summary

After repeating the treatment eight times a total of 0 faults were left undetected in the simulation. This treatment is very similar to treatment number 14, (the best of breed) with little statistical difference between the two in the category of resources required to discover the last fault.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 76167 resource units out of a possible 175000. This treatment ranked 2th out of the 32 treatments in this category. The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 19981 resource units. This treatment ranked 5th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 527. This treatment ranked 6th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 2rd out of 32.

Comparing treatments, combination number 10 is a very effective treatment compared to treatment number 14. On average, this treatment consumed approximately 15% more resources to detect the faults in the simulator.

Treatment 13 (00110) Results

Treatment 13 is assigned the low signal values for all five variables in the designed experiment except the third (GA-crossover) and forth (Ind. sweep) positions.

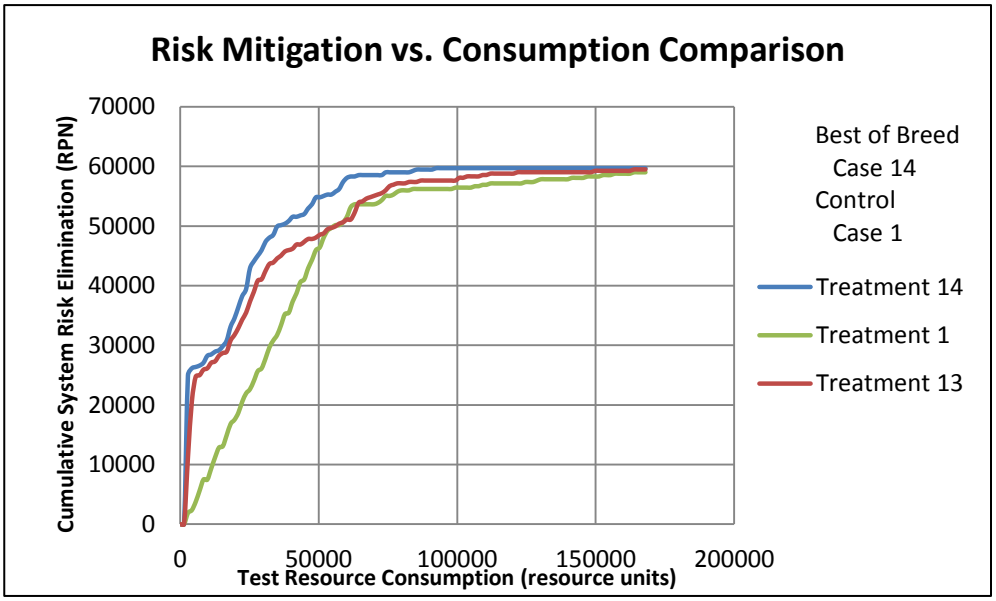


Figure A.21: Risk mitigation comparison, treatments 1, 14 and 13

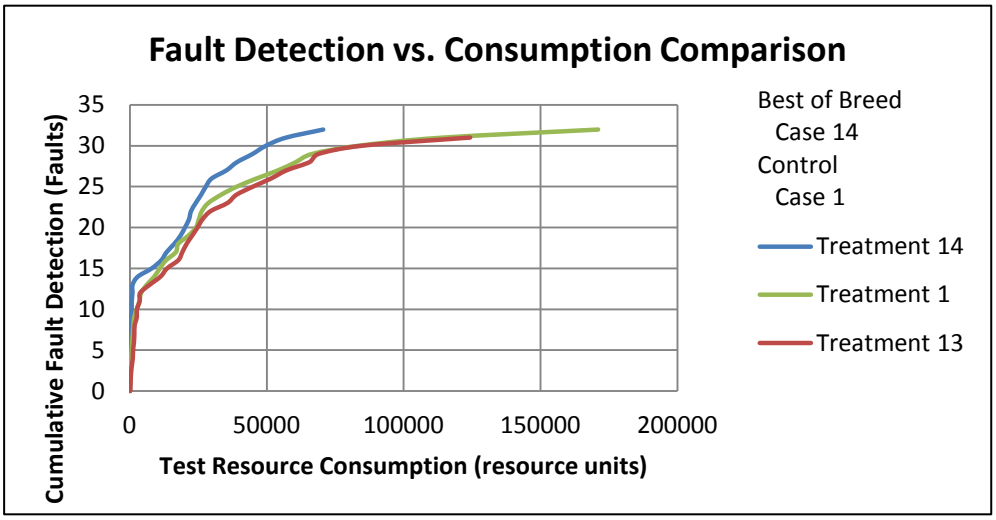


Figure A.22: Cumulative fault detection comparison, treatments 1, 14, and 13

Table A.11: Compiled DOE results for treatment number 13

Treatment # 13									
Treatment Order 00110	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	448	484	581	384	502	467	450	435	469
Average Resource Count	22702	24488	29024	19671	25137	23500	22838	22074	23679
Resource Count at Last Fault	73248	161472	120096	80160	147840	101136	109584	98064	111450
No. of undetected faults	0	0	0	0	1	0	0	0	
Total Resource Consumption					175008				
Total Test Case					3648				
Adjusted Ave TC Count	448	484	581	384	623	467	450	435	484
Adjusted Ave Res Count	22702	24488	29024	19671	30914	23500	22838	22074	24401
Adjusted Res Count Last Fault	73248	161472	120096	80160	210010	57257	109584	75333	110895

Summary

After repeating the treatment eight times a total of 1 fault was left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 110895 resource units out of a possible 175000. This treatment ranked 13th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 24401 resource units. This treatment ranked 15th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 484. This treatment ranked 11th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 14th out of 32.

Comparing treatments, combination number 13 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 15 (01110) Results

Treatment 15 is assigned the high signal values for all five variables in the designed experiment except the first (cost) and fifth (GA-probability) positions.

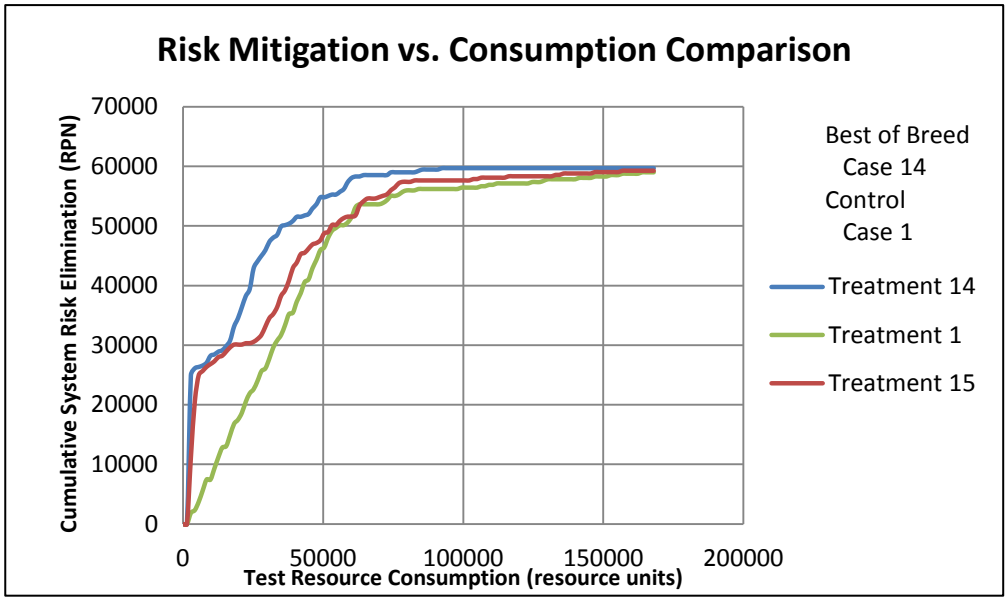


Figure A.23: Risk mitigation comparison, treatments number 1, 14, and 15

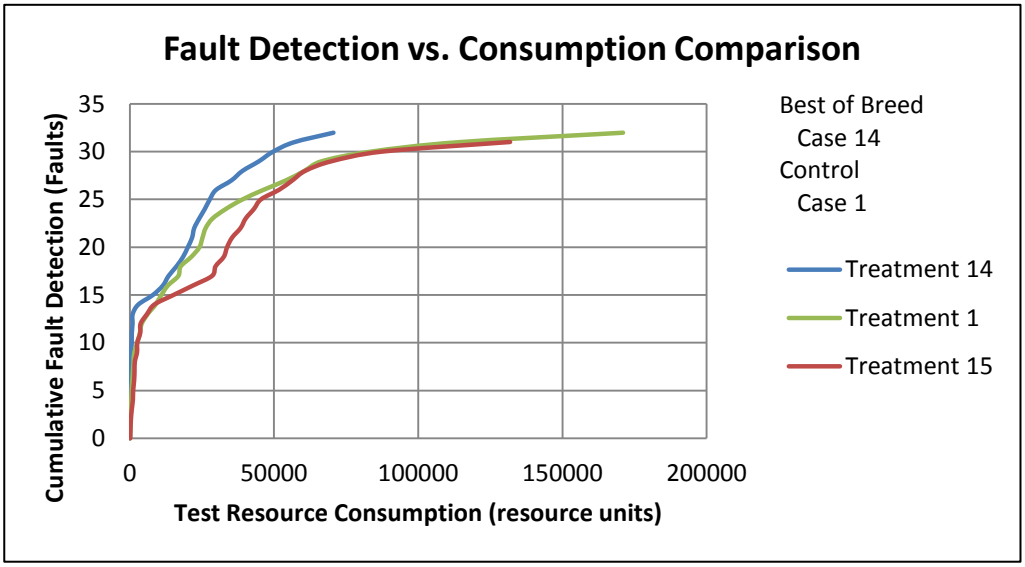


Figure A.24: Cumulative fault detection comparison, treatments number 1, 14, and 15

Table A.12: Compiled DOE results for treatment number 15

Treatment # 15									
Treatment Order 01110	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	476	495	601	548	486	508	429	515	507
Average Resource Count	23988	25095	30200	27418	24647	25725	21690	25954	25590
Resource Count at Last Fault	80736	154656	134352	101472	145008	74832	75840	151872	114846
No. of undetected faults	1	0	0	0	1	0	0	0	
Total Resource Consumption	175104				175008				
Total Test Case	3696				3696				
Adjusted Ave TC Count	600	495	601	548	609	508	429	515	538
Adjusted Ave Res Count	29805	25095	30200	27418	30440	25725	21690	25954	27041
Adjusted Res Count Last Fault	210125	154656	134352	101472	210010	74832	75840	151872	139145

Summary

After repeating the treatment eight times a total of 2 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 139145 resource units out of a possible 175000. This treatment ranked 21st out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 27041 resource units. This treatment ranked 21st out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 538. This treatment ranked 21st out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 22nd out of 32.

Comparing treatments, combination number 15 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 16 (11110) Results

Treatment 16 is assigned the high signal values for all five variables in the designed experiment except the first (cost) position.

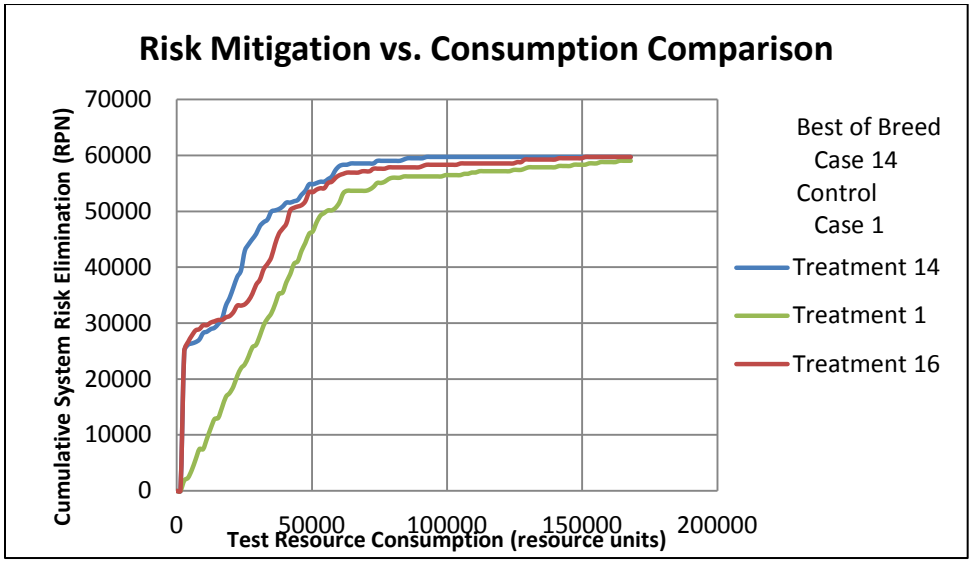


Figure A.25: Risk mitigation comparison, treatments 1, 14, and 16

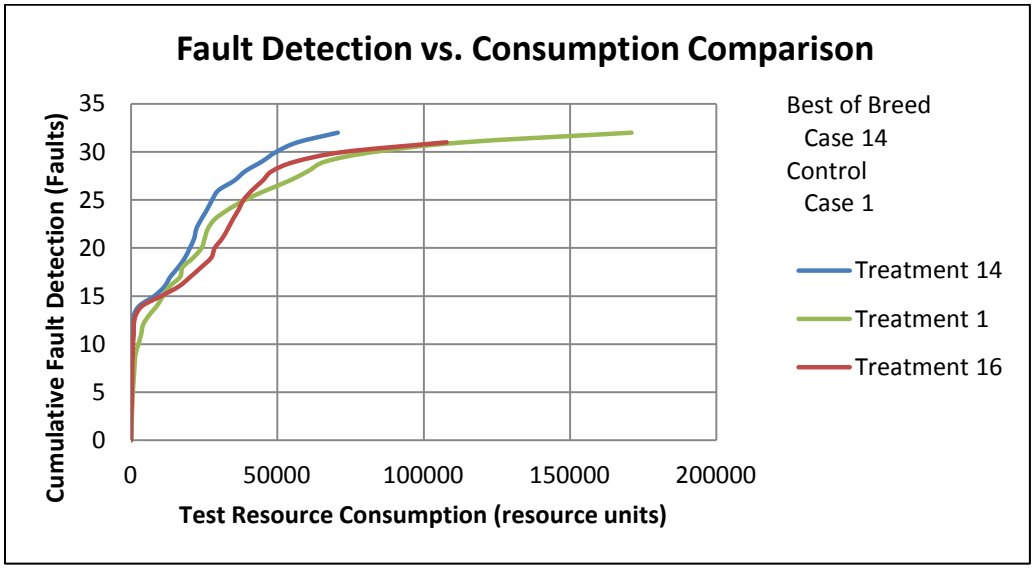


Figure A.26: Cumulative fault detection comparison, treatments 1, 14, and 16

Table A.13: Compiled DOE results for treatment number 16

Treatment # 16									
Treatment Order 11110	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	831	512	510	527	677	536	561	670	603
Average Resource Count	20430	19285	19144	19618	25726	20076	21273	25192	21343
Resource Count at Last Fault	70933	88464	70698	126493	126499	90969	149058	139535	107831
No. of undetected faults	0	0	0	0	0	0	0	0	
Total Resource Consumption									
Total Test Case									
Adjusted Ave TC Count	831	512	510	527	677	536	561	670	603
Adjusted Ave Res Count	20430	19285	19144	19618	25726	20076	21273	25192	21343
Adjusted Res Count Last Fault	70933	88464	70698	126493	126499	74832	149058	139535	105814

Summary

After repeating the treatment eight times a total of 0 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 105814 resource units out of a possible 175000. This treatment ranked 16th out of the 32 treatments in this category. The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 21343 resource units. This treatment ranked 6th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 603. This treatment ranked 10th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 10th out of 32.

Comparing treatments, combination number 16 is similar to treatment 14 but not as aggressive, especially in the middle of the fault detection process . Due to the slightly less aggressive search process, the average amount of resources consumed to detect the last fault took approximately 60% more than the treatment 14, but only 21% more resources for the average point of detecting all faults.

Treatment 17 (00001) Results

Treatment 17 is assigned the low signal values for all five variables in the designed experiment except the fifth (GA-probability) position.

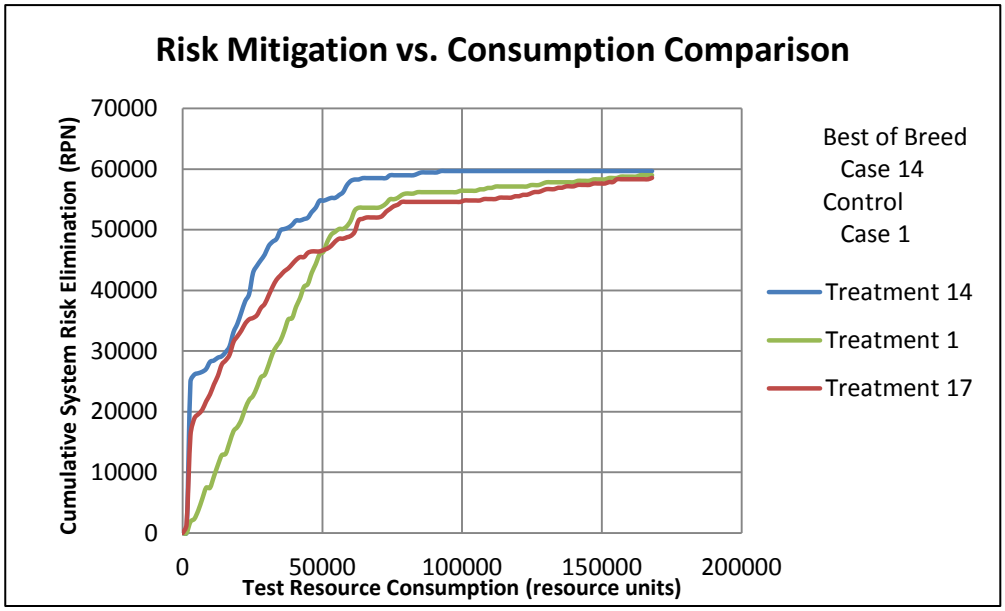


Figure A.27: Risk mitigation comparison, treatments 1, 14, and 17

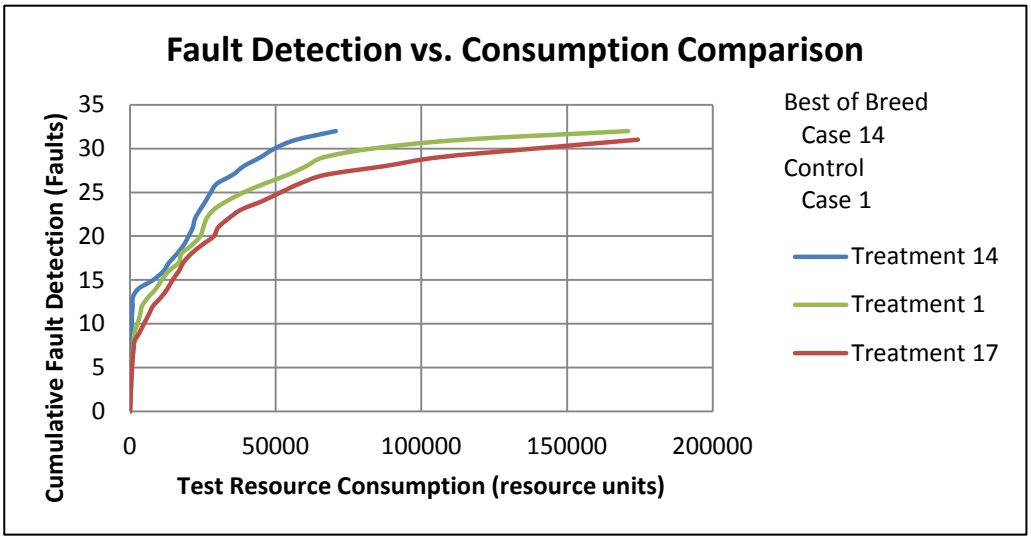


Figure A.28: Cumulative fault detection comparison, treatments 1, 14, and 17

Table A.14: Compiled DOE results for treatment number 17

Treatment # 17									
Treatment Order 00001	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	572	540	429	656	621	539	624	464	556
Average Resource Count	28541	27284	21675	32663	31199	26973	31222	23584	27893
Resource Count at Last Fault	150432	145440	74448	154032	165744	139056	171504	105072	138216
No. of undetected faults	1	0	0	0	0	2	1	0	
Total Resource Consumption	175104					175104	175008		
Total Test Case	3590					3588	3570		
Adjusted Ave TC Count	689	540	429	656	621	774	738	464	614
Adjusted Ave Res Count	34215	27284	21675	32663	31199	38420	36809	23584	30731
Adjusted Res Count Last Fault	210125	145440	74448	154032	165744	245146	210010	105072	163752

Summary

After repeating the treatment eight times a total of 4 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 163752 resource units out of a possible 175000. This treatment ranked 28th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 30731 resource units. This treatment ranked 28th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 614. This treatment ranked 28th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 28th out of 32. The variable combination in this treatment is less effective than treatment number 1.

Comparing treatments, combination number 17 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 18 (10001) Results

Treatment 18 is assigned the low signal values for all five variables in the designed experiment except the first (cost) and fifth (GA-probability) positions.

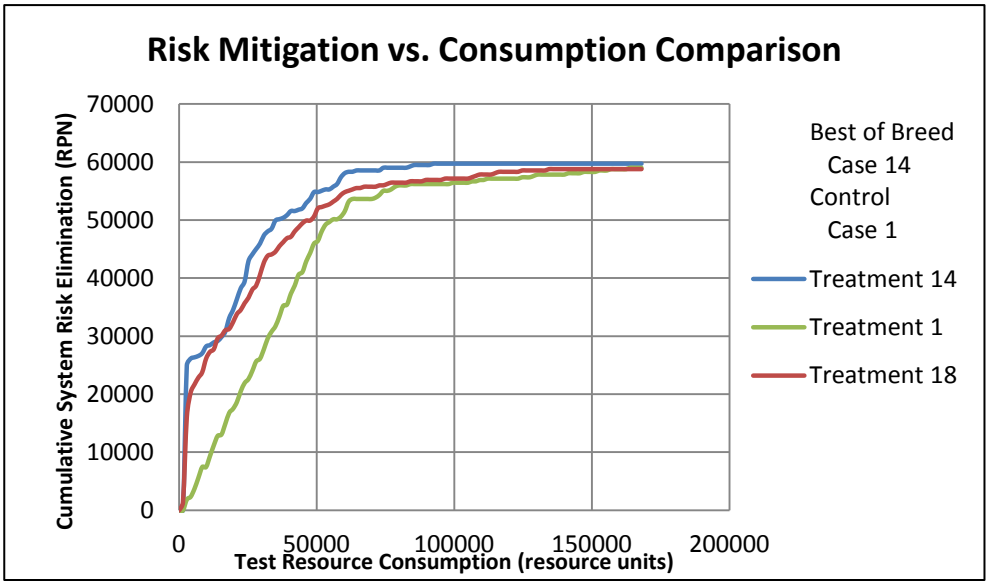


Figure A.29: Risk mitigation comparison, treatments number 1, 14, and 18

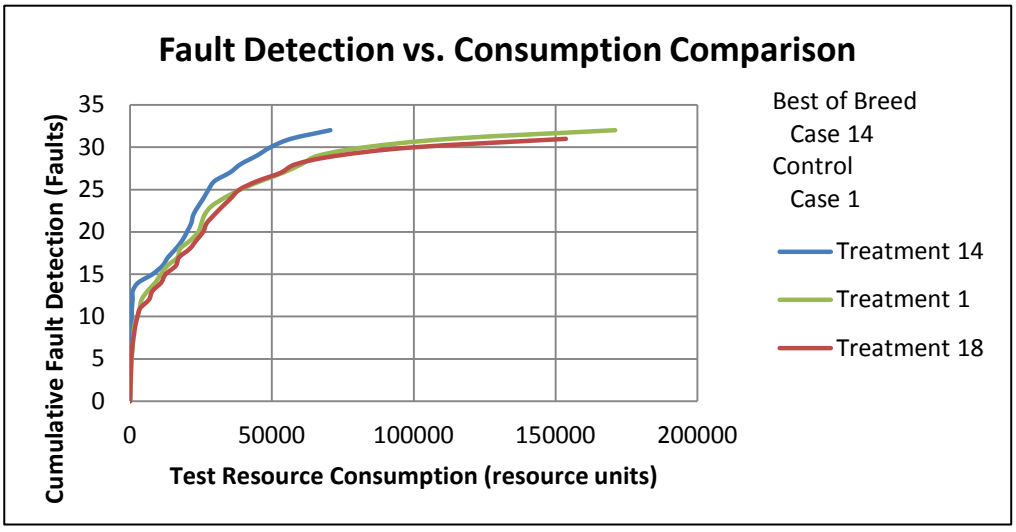


Figure A.30: Cumulative fault detection comparison, treatments number 1, 14, and 18

Table A.15: Compiled DOE results for treatment number 18

Treatment # 18									
Treatment Order 10001	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	484	445	511	473	516	685	626	467	526
Average Resource Count	19230	17686	20581	19175	20823	26698	25087	18924	21026
Resource Count at Last Fault	70569	60931	106174	94116	104158	131653	121930	75630	95645
No. of undetected faults	1	0	0	2	0	0	0	0	
Total Resource Consumption	175080			175047					
Total Test Case	4559			4520					
Adjusted Ave TC Count	640	445	511	782	516	685	626	467	584
Adjusted Ave Res Count	25195	17686	20581	31105	20823	26698	25087	18924	23262
Adjusted Res Count Last Fault	210096	60931	106174	245066	104158	131653	121930	75630	131955

Summary

After repeating the treatment eight times a total of 3 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 131955 resource units out of a possible 175000. This treatment ranked 27th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 23262 resource units. This treatment ranked 27th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 584. This treatment ranked 27th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 27th out of 32. The variable combination in this treatment is less effective than treatment number 1.

Comparing treatments, combination number 18 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 19 (01001) Results

Treatment 19 is assigned the low signal values for all five variables in the designed experiment except the second (risk prioritization) and fifth (GA-probability) positions.

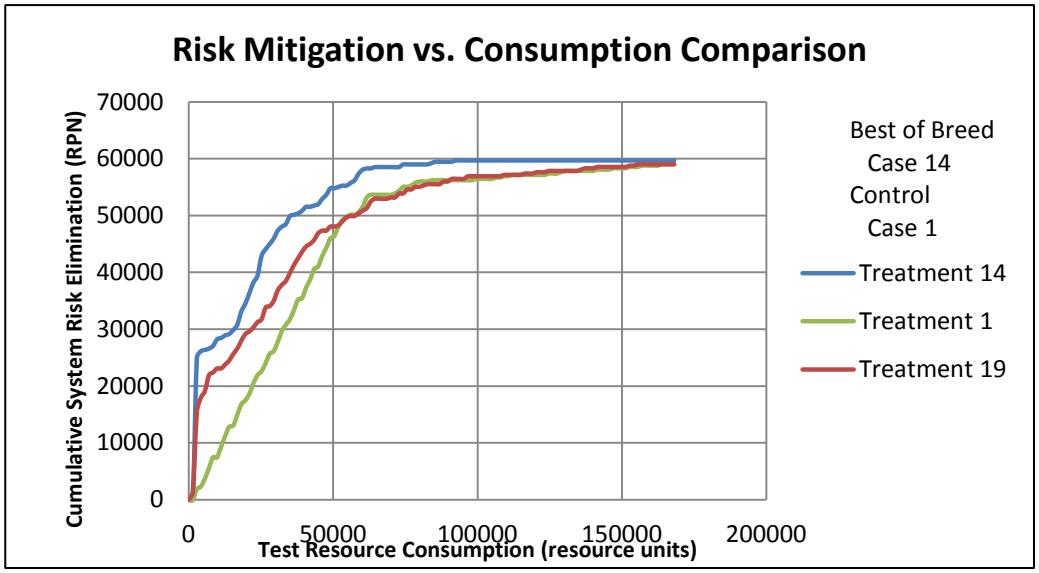


Figure A.31: Risk mitigation comparison for treatments 1, 14, and 19

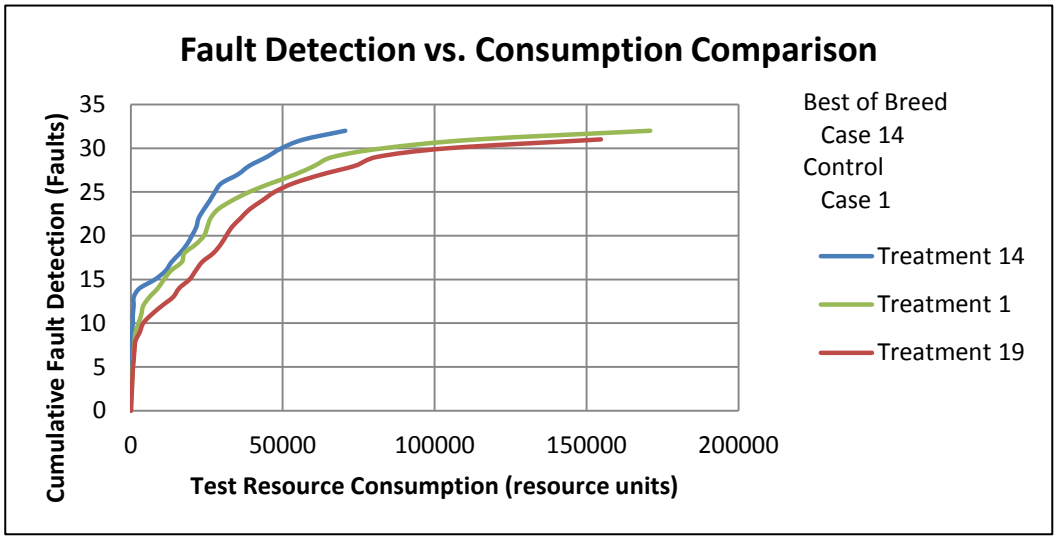


Figure A.32: Cumulative fault detection comparison, treatments number 1, 14, and 19

Table A.16: Compiled DOE results for treatment number 19

Treatment # 19									
Treatment Order 01001	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	677	495	485	487	510	668	418	598	542
Average Resource Count	34026	25161	24330	24412	25696	33433	21152	29782	27249
Resource Count at Last Fault	154032	81216	88656	149856	123216	170736	73248	135600	122070
No. of undetected faults	0	0	1	1	0	0	0	0	
Total Resource Consumption			175104	175104					
Total Test Case			3590	3570					
Adjusted Ave TC Count	677	495	604	606	510	668	418	598	572
Adjusted Ave Res Count	34026	25161	30136	30216	25696	33433	21152	29782	28700
Adjusted Res Count Last Fault	154032	81216	210125	210125	123216	170736	73248	135600	144787

Summary

After repeating the treatment eight times a total of 2 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 144787 resource units out of a possible 175000. This treatment ranked 23rd out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 28700 resource units. This treatment ranked 23th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 572. This treatment ranked 23th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 23th out of 32. The variable combination in this treatment is similar to the effectiveness of treatment number 1.

Comparing treatments, combination number 19 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 20 (11001) Results

Treatment 20 is assigned the high signal values for all five variables in the designed experiment except the third (GA-crossover) and forth (Ind. sweep) positions.

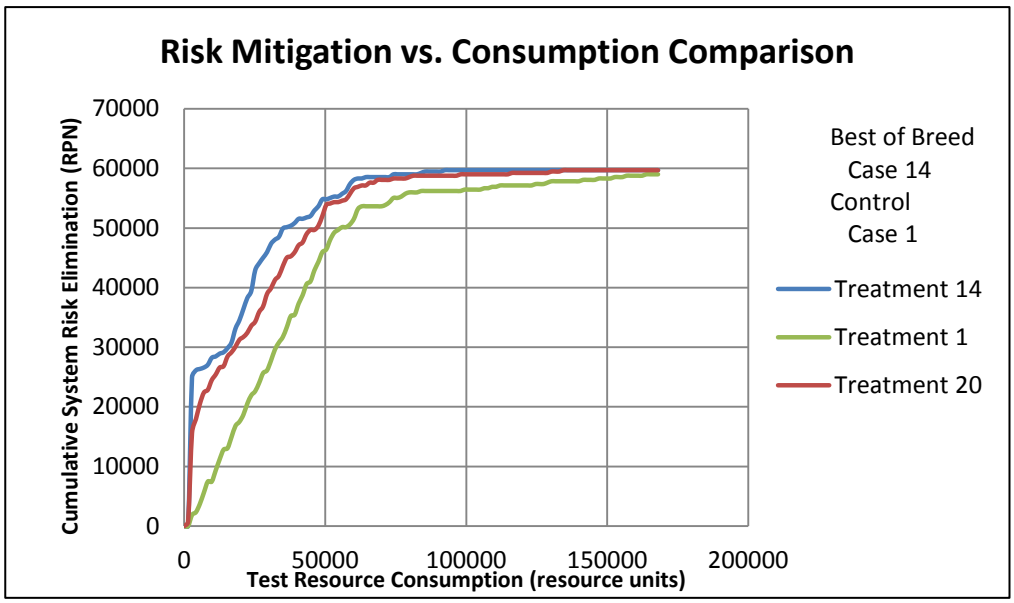


Figure A.33: Risk mitigation comparison, treatments 1, 14, and 20

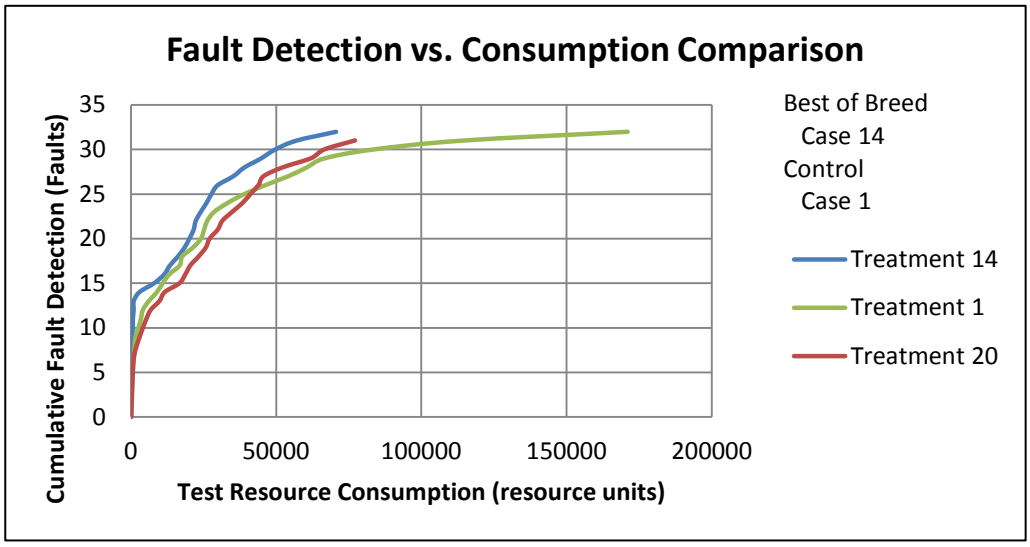


Figure A.34: Cumulative fault detection comparison, treatments 1, 14, and 20

Table A.17: Compiled DOE results for treatment number 20

Treatment # 20									
Treatment Order 11001	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	488	557	491	740	574	479	571	477	547
Average Resource Count	19728	22162	19602	29664	22696	19119	23120	18866	21870
Resource Count at Last Fault	57791	78593	56711	132526	78375	56841	96521	60029	77173
No. of undetected faults	0	0	0	0	0	0	0	0	
Total Resource Consumption									
Total Test Case									
Adjusted Ave TC Count	488	557	491	740	574	479	571	477	547
Adjusted Ave Res Count	19728	22162	19602	29664	22696	19119	23120	18866	21870
Adjusted Res Count Last Fault	57791	78593	56711	132526	78375	56841	96521	60029	77173

Summary

After repeating the treatment eight times a total of 0 faults were left undetected in the simulation. This treatment ranks in the top 5. The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 77173 resource units out of a possible 175000. This treatment ranked 3rd out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 21870 resource units. This treatment ranked 9th out of the 32 treatments in this category. The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 547. This treatment ranked 9th out of the 32 treatments in this category. The overall ranking of this treatment relative to the weighted average of the three priorities is 5th out of 32.

Comparing treatments, combination number 20 is a very effective treatment compared to treatment number 14 when comparing the total amount of resources required to detect the last fault. Although this treatment ranked fourth overall on the weighted scale, it was the 9th most efficient in average resource consumption and the 9th most effective in the amount of test cases required to discover the faults. In the early stages, treatment number 1 is actually more effective at fault detection. The results indicate this treatment may be more sensitive to the fault locations in a complex system than the other top five.

Treatment 21 (00101) Results

Treatment 21 is assigned the low signal values for all five variables in the designed experiment except the third (GA-crossover) and fifth (GA-probability) positions.

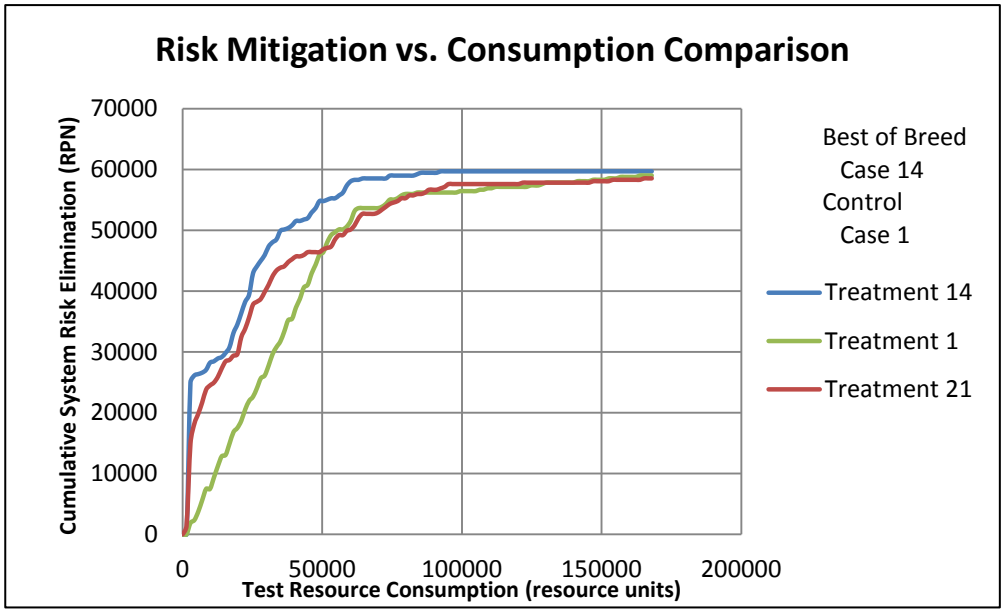


Figure A.35: Risk mitigation comparison, treatments 1, 14, and 21

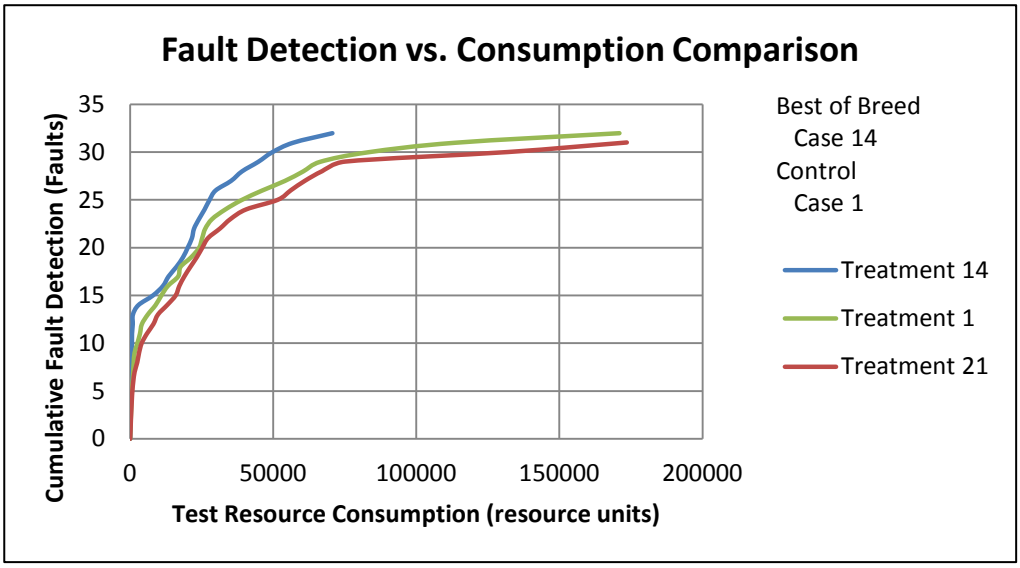


Figure A.36: Cumulative fault detection comparison, treatments 1, 14 and 21

Table A.18: Compiled DOE results for treatment number 21

Treatment # 21									
Treatment Order 00101	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	489	467	400	575	441	531	454	499	482
Average Resource Count	24567	23694	20446	28850	22312	26848	22890	25302	24364
Resource Count at Last Fault	144720	85104	79152	93552	92304	151824	76416	163104	110772
No. of undetected faults	0	0	2	0	1	0	2	0	
Total Resource Consumption			175056		175056		175008		
Total Test Case			3572		3569		3571		
Adjusted Ave TC Count	489	467	643	575	561	531	693	499	557
Adjusted Ave Res Count	24567	23694	32297	28850	28179	26848	34585	25302	28040
Adjusted Res Count Last Fault	144720	85104	245078	93552	210067	151824	245011	163104	167308

Summary

After repeating the treatment eight times a total of 5 faults were left undetected in the simulation. The search process can become too aggressive.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 167308 resource units out of a possible 175000. This treatment ranked 30th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 28040 resource units. This treatment ranked 29th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 557. This treatment ranked 29th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 29th out of 32. The variable combination in this treatment is less effective than treatment number 1.

Comparing treatments, combination number 21 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 22 (10101) Results

Treatment 22 is assigned the high signal values for all five variables in the designed experiment except the second (risk prioritization) and forth (Ind. sweep) positions.

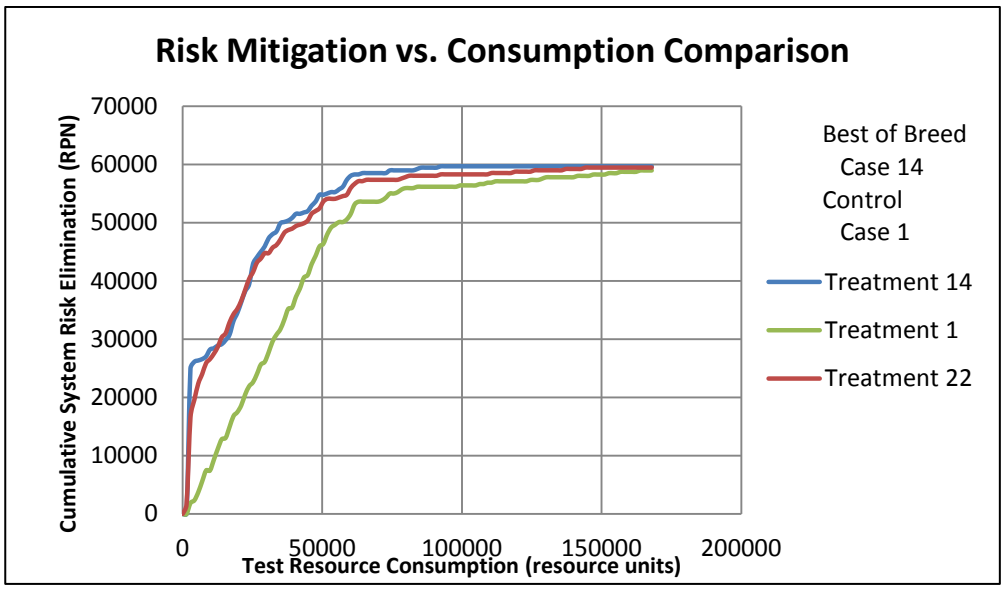


Figure A.37: Risk mitigation comparison, treatments 1, 14, and 22

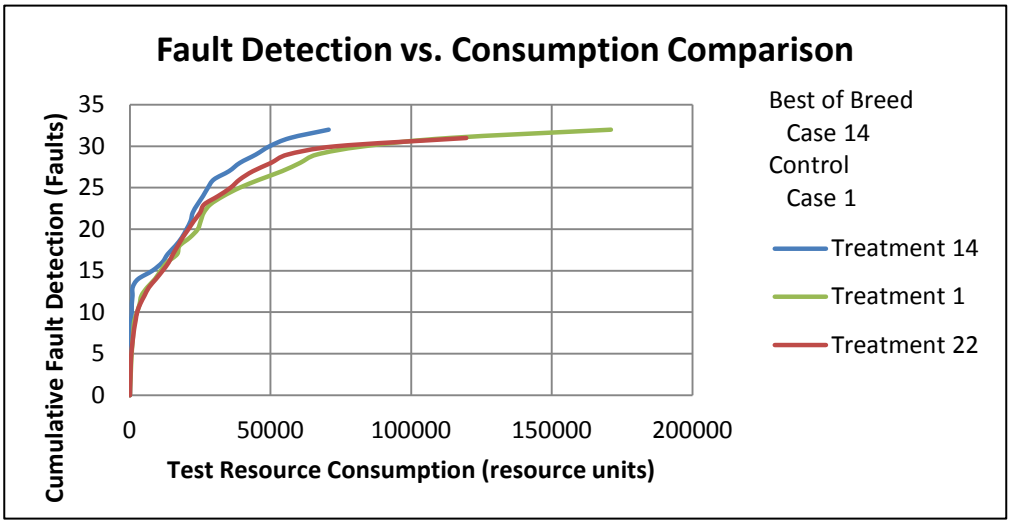


Figure A.38: Cumulative fault detection comparison, treatments 1, 14, and 22

Table A.19: Compiled DOE results for treatment number 22

Treatment # 22									
Treatment Order 10101	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	523	397	528	559	469	438	421	439	472
Average Resource Count	20899	16641	20896	22129	19070	17808	16919	18013	19047
Resource Count at Last Fault	141525	59544	116499	134513	78912	123716	78149	108010	105109
No. of undetected faults	0	0	0	1	0	0	0	0	
Total Resource Consumption				175066					
Total Test Case				4549					
Adjusted Ave TC Count	523	397	528	712	469	438	421	439	491
Adjusted Ave Res Count	20899	16641	20896	28002	19070	17808	16919	18013	19781
Adjusted Res Count Last Fault	141525	59544	116499	210079	78912	123716	78149	108010	114554

Summary

After repeating the treatment eight times a total of 1 fault was left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 114554 resource units out of a possible 175000. This treatment ranked 14th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 19781 resource units. This treatment ranked 11th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 491. This treatment ranked 12th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 13th out of 32.

Comparing treatments, combination number 13 shows quick risk mitigation growth very similar to treatment number 14, during the test runs, this combination became too greedy and left a fault undetected.

Treatment 24 (11101) Results

Treatment 24 is assigned the high signal values for all five variables in the designed experiment except the forth (Ind. sweep) position.

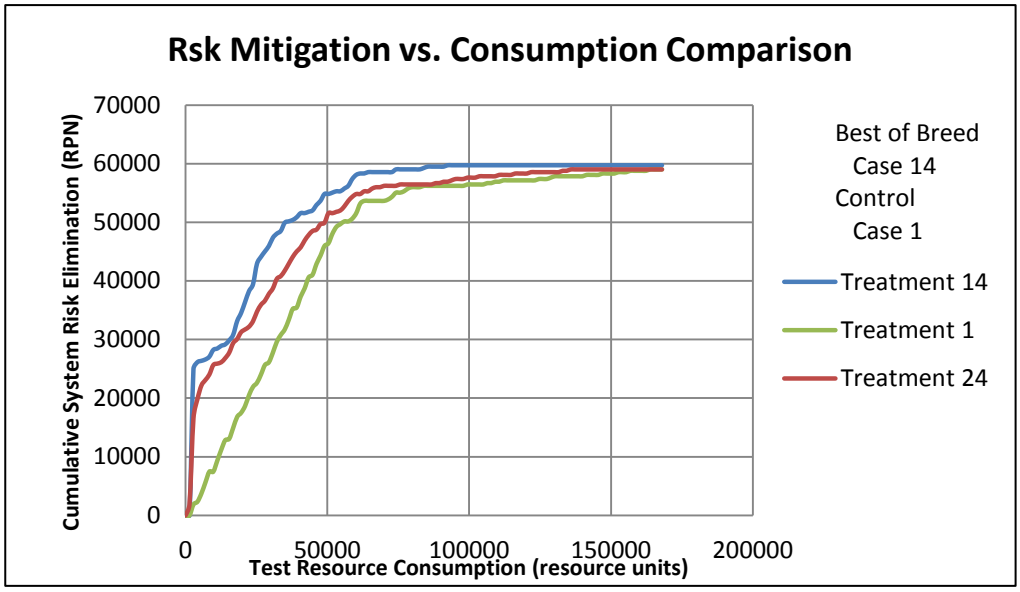


Figure A.39: Risk mitigation comparison, treatments number 1, 14, and 24

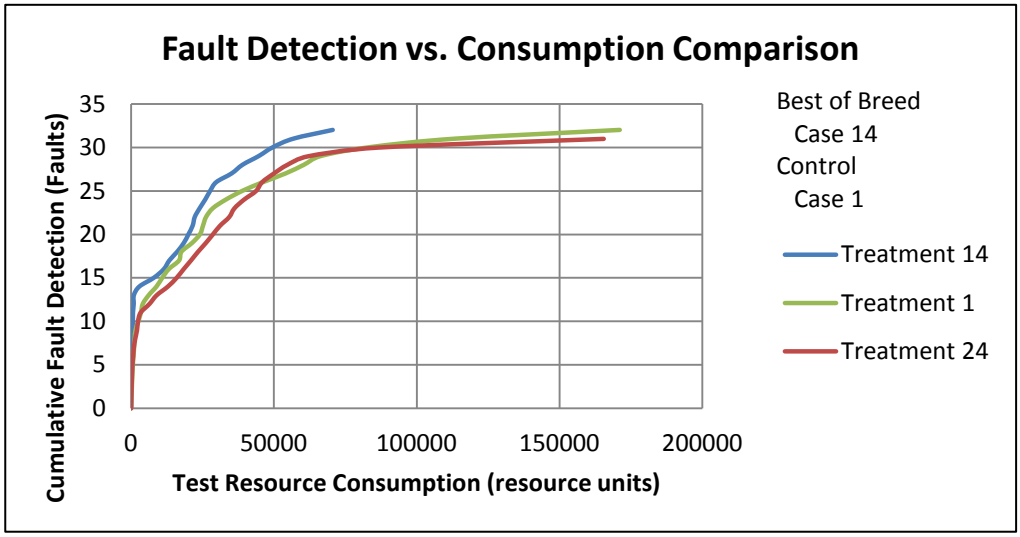


Figure A.40: Cumulative fault detection comparison, treatments 1, 14, and 24

Table A.20: Compiled DOE results for treatment number 24

Treatment # 24									
Treatment Order 11101	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	646	459	495	577	589	489	642	631	566
Average Resource Count	25660	18455	19613	22938	23255	19670	25248	24957	22475
Resource Count at Last Fault	133041	57631	93050	101787	114494	88459	131560	91288	101414
No. of undetected faults	0	1	0	0	0	1	0	1	
Total Resource Consumption		175010				175042		175084	
Total Test Case		4549				4533		4541	
Adjusted Ave TC Count	646	615	495	577	589	644	642	782	624
Adjusted Ave Res Count	25660	24441	19613	22938	23255	25619	25248	30743	24690
Adjusted Res Count Last Fault	133041	210012	93050	101787	114494	210050	131560	210101	150512

Summary

After repeating the treatment eight times a total of 3 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 150512 resource units out of a possible 175000. This treatment ranked 24th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 24690 resource units. This treatment ranked 24th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 624. This treatment ranked 26th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 24th out of 32. The variable combination in this treatment is similar to the effectiveness of treatment number 1.

Comparing treatments, combination number 24 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 25 (00011) Results

Treatment 25 is assigned the low signal values for all five variables in the designed experiment except the forth (Ind. sweep) and fifth (GA-probability) positions.

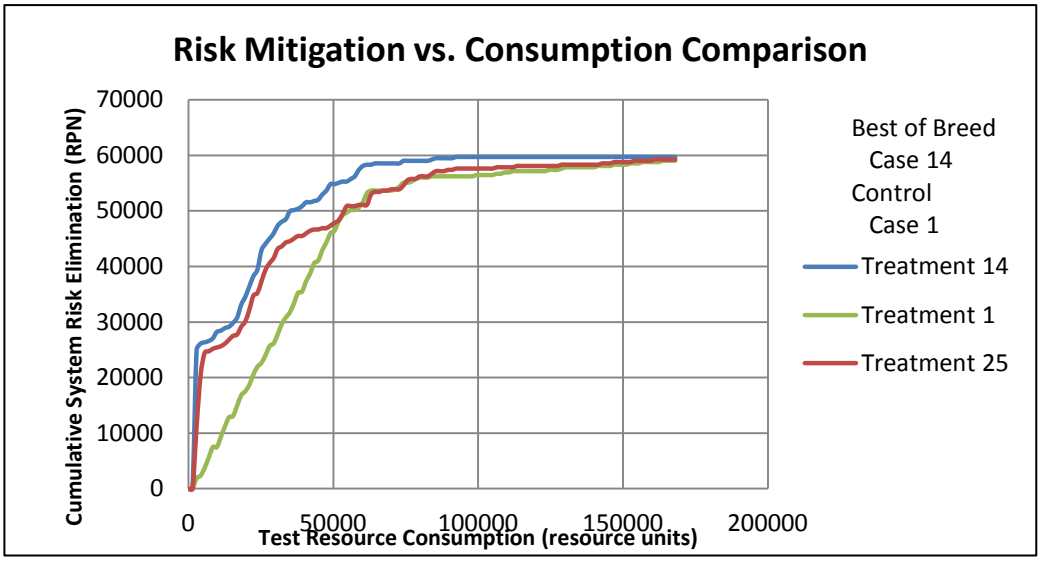


Figure A.41: Risk mitigation comparison, treatments 1, 14, and 25

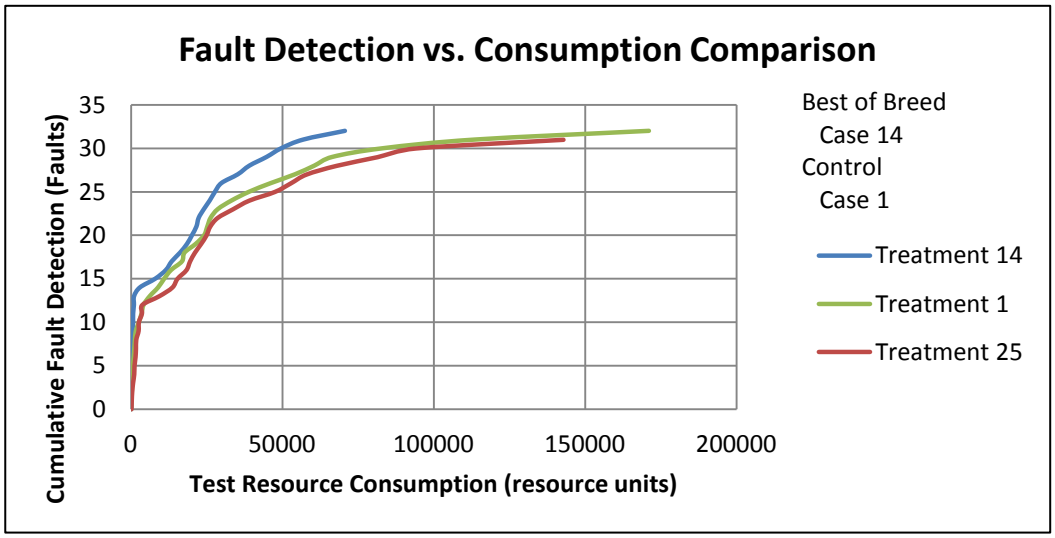


Figure A.42: Cumulative fault detection comparison, treatments 1, 14, and 25

Table A.21: Compiled DOE results for treatment number 25

Treatment # 25									
Treatment Order 00011	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	533	430	370	459	412	646	491	539	485
Average Resource Count	26763	21708	19036	23115	20966	32224	24579	27063	24432
Resource Count at Last Fault	145440	88128	72864	81552	72384	158784	126384	111888	107178
No. of undetected faults	0	1	0	0	0	0	1	0	
Total Resource Consumption		175056					175008		
Total Test Case		3571					3570		
Adjusted Ave TC Count	533	550	370	459	412	646	610	539	515
Adjusted Ave Res Count	26763	27594	19036	23115	20966	32224	30374	27063	25892
Adjusted Res Count Last Fault	145440	210067	72864	81552	72384	158784	210010	111888	132874

Summary

After repeating the treatment eight times a total of 2 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 132874 resource units out of a possible 175000. This treatment ranked 19th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 25892 resource units. This treatment ranked 20th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 515. This treatment ranked 20th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 19th out of 32.

Comparing treatments, combination number 25 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 26 (10011) Results

Treatment 26 is assigned the high signal values for all five variables in the designed experiment except the second (risk prioritization) and third (GA-crossover) positions.

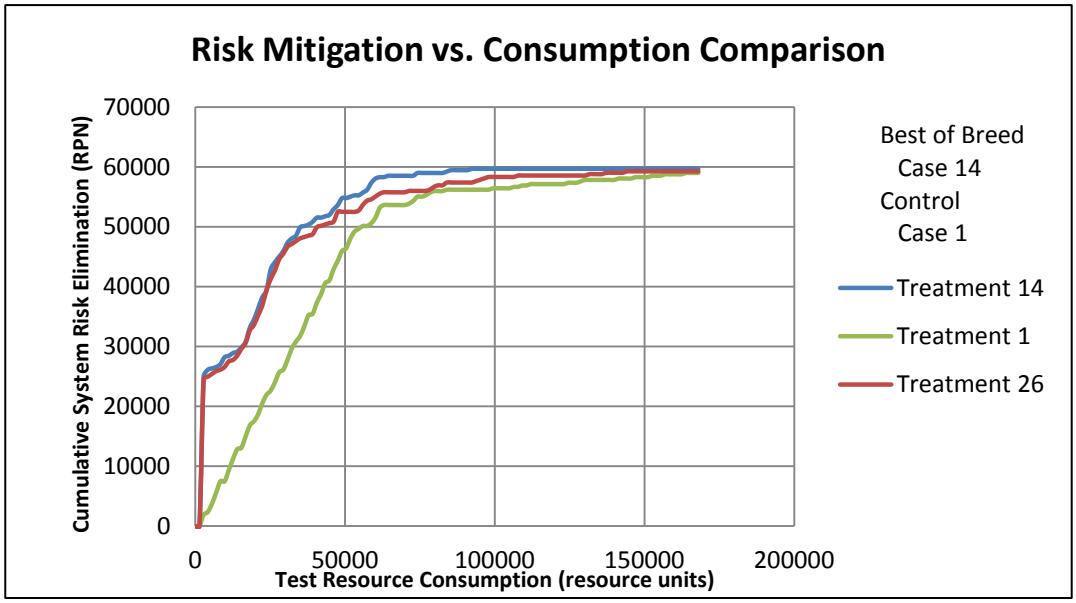


Figure A.43: Risk mitigation comparison, treatments 1, 14, and 26

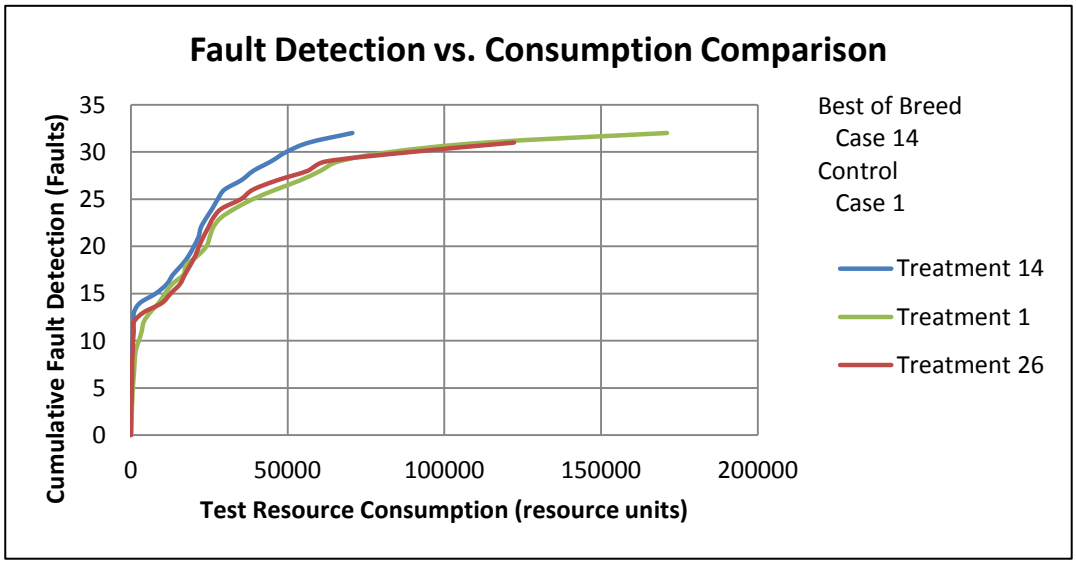


Figure A.44: Cumulative fault detection comparison, treatments 1, 14, and 26

Table A.22: Compiled DOE results for treatment number 26

Treatment # 26									
Treatment Order 10011	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	436	538	562	531	609	585	729	426	552
Average Resource Count	15920	19814	20814	19446	22643	21339	27194	15549	20340
Resource Count at Last Fault	54555	95053	95531	105296	129203	135165	169512	78480	107849
No. of undetected faults	0	0	0	0	0	1	0	0	
Total Resource Consumption						175065			
Total Test Case						4617			
Adjusted Ave TC Count	436	538	562	531	609	740	729	426	571
Adjusted Ave Res Count	15920	19814	20814	19446	22643	27237	27194	15549	21077
Adjusted Res Count Last Fault	54555	0	95531	105296	129203	210078	169512	78480	105332

Summary

After repeating the treatment eight times a total of 1 fault was left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 105332 resource units out of a possible 175000. This treatment ranked 12th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 21077 resource units. This treatment ranked 13th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 571. This treatment ranked 12th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 16th out of 32.

Comparing treatments, combination number 13 shows quick risk mitigation growth very similar to treatment number 14, during the test runs, this combination became too greedy and left a fault undetected.

Treatment 27 (01011) Results

Treatment 27 is assigned the high signal values for all five variables in the designed experiment except the first (cost) and third (GA-crossover) positions.

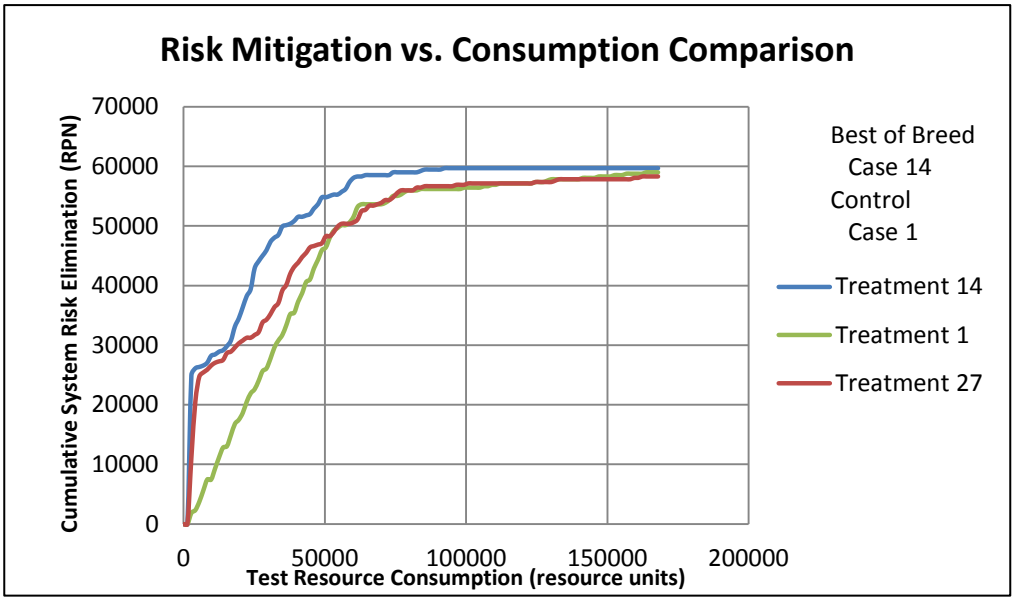


Figure A.45: Risk mitigation comparison, treatments 1, 14, and 27

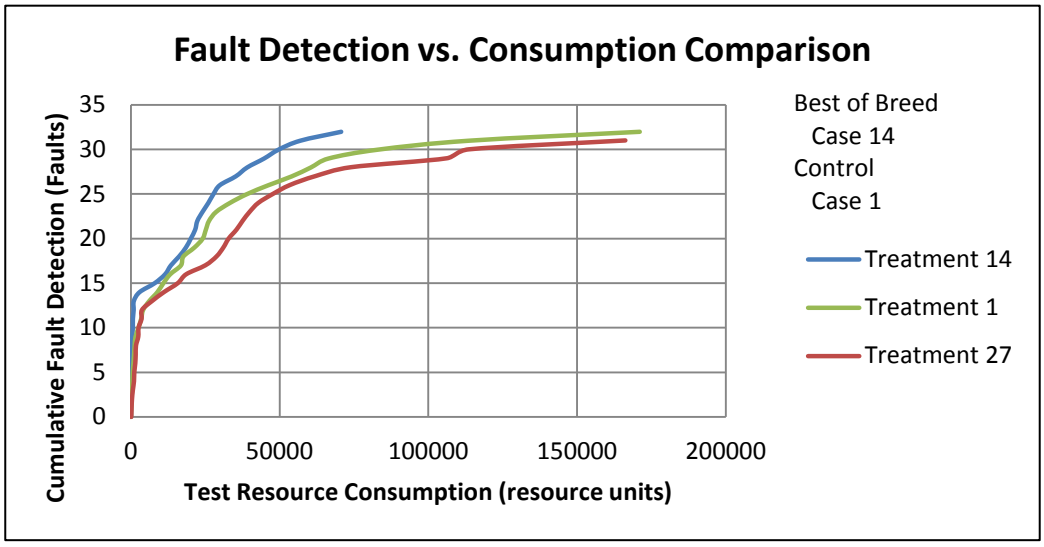


Figure A.46: Cumulative fault detection comparison, treatments 1, 14 and 27

Table A.23: Compiled DOE results for treatment number 27

Treatment # 27									
Treatment Order 01011	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	565	418	636	484	466	513	439	467	499
Average Resource Count	28394	21083	31581	24156	23778	25743	22420	23518	25084
Resource Count at Last Fault	131232	73152	159792	122736	80256	129888	72144	83280	106560
No. of undetected faults	0	1	1	3	0	0	0	1	
Total Resource Consumption		175104	175008	175104				175056	
Total Test Case		3572	3588	3593				3570	
Adjusted Ave TC Count	565	539	751	843	466	513	439	586	588
Adjusted Ave Res Count	28394	26991	37157	41591	23778	25743	22420	29348	29428
Adjusted Res Count Last Fault	131232	210125	210010	280166	80256	129888	72144	210067	165486

Summary

After repeating the treatment eight times a total of 6 faults were left undetected in the simulation. The treatment ranked last out of the 32 combinations. The search process can become too aggressive.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 165486 resource units out of a possible 175000. This treatment ranked 31st out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 29428 resource units. This treatment ranked 31st out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 588. This treatment ranked 32nd out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 31st out of 32. The variable combination in this treatment is less effective than treatment number 1.

Comparing treatments, combination number 27 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 28 (11011) Results

Treatment 28 is assigned the high signal values for all five variables in the designed experiment except the first (cost) and third (GA-crossover) positions.

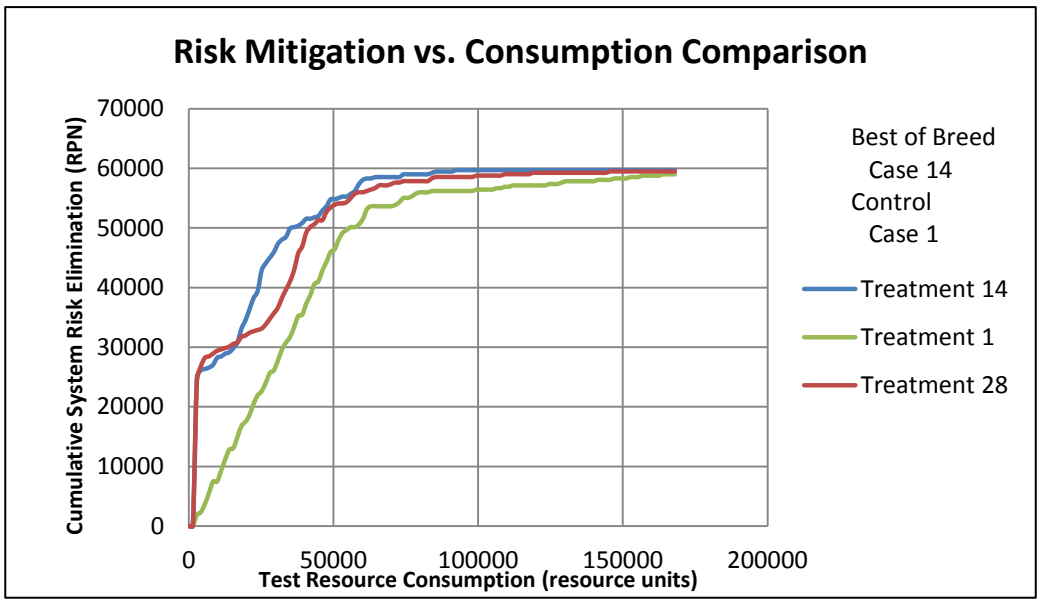


Figure A.47: Risk mitigation comparison, treatments 1, 14, and 28

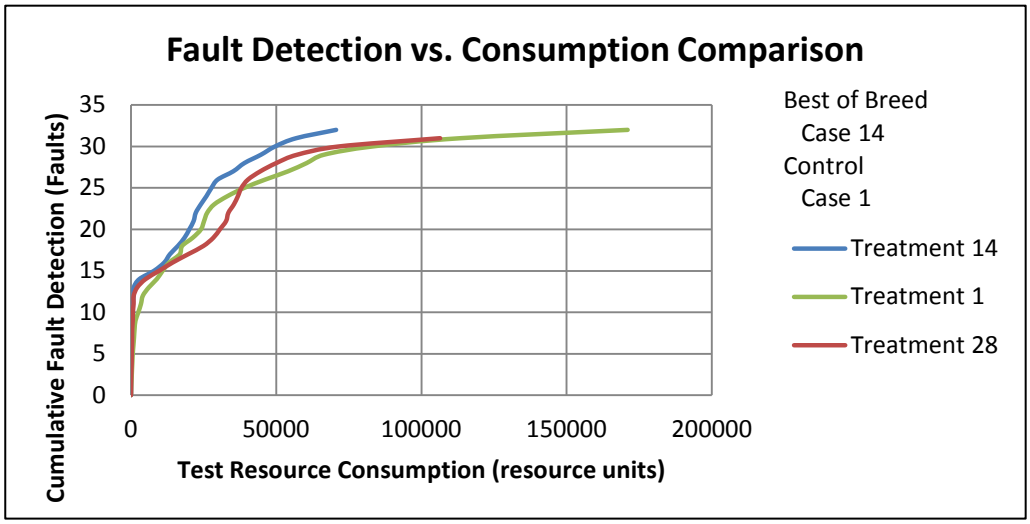


Figure A.48: Cumulative fault detection comparison, treatments, 1, 14 and 28

Table A.24: Compiled DOE results for treatment number 28

Treatment # 28									
Treatment Order 11011	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	513	641	572	535	501	546	678	475	558
Average Resource Count	18755	23638	21077	19747	18598	20120	24974	17610	20565
Resource Count at Last Fault	84019	143008	69111	55284	54662	82305	106527	62064	82123
No. of undetected faults	0	0	0	1	0	0	0	0	
Total Resource Consumption				175046					
Total Test Case				4628					
Adjusted Ave TC Count	513	641	572	692	501	546	678	475	577
Adjusted Ave Res Count	18755	23638	21077	25694	18598	20120	24974	17610	21308
Adjusted Res Count Last Fault	84019	143008	69111	210055	54662	82305	106527	62064	101469

Summary

After repeating the treatment eight times a total of 1 fault was left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 101469 resource units out of a possible 175000. This treatment ranked 11th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 21308 resource units. This treatment ranked 14th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 577. This treatment ranked 17th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 11th out of 32.

Comparing treatments, combination number 28 shows quick risk mitigation growth very similar to treatment number 14, during the test runs, this combination became too greedy and left a fault undetected.

Treatment 29 (00111) Results

Treatment 29 is assigned the high signal values for all five variables in the designed experiment except the first (cost) and second (risk prioritization) positions.

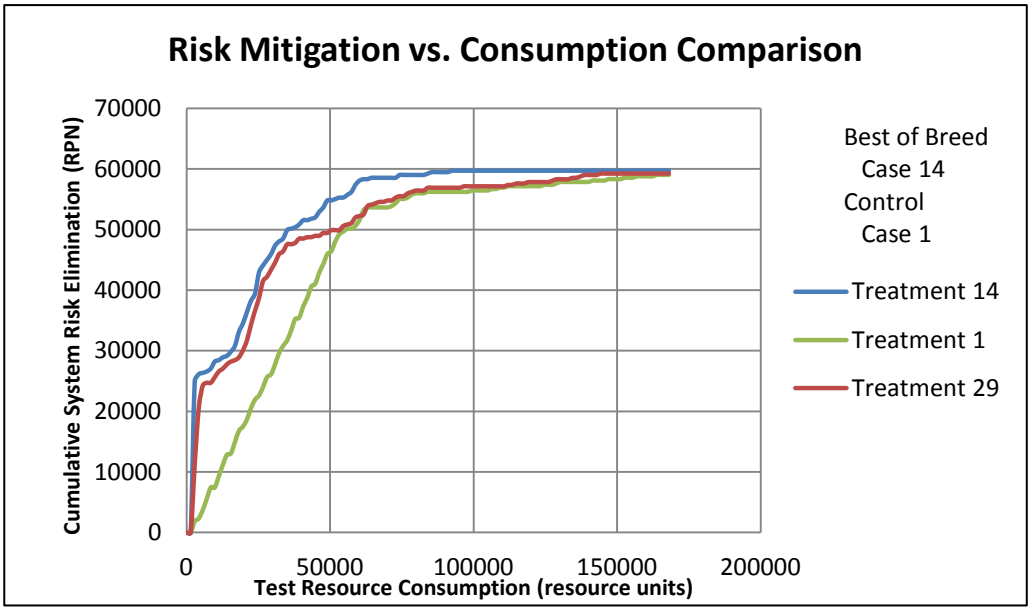


Figure A.49: Risk mitigation comparison, treatments 1, 14, and 29

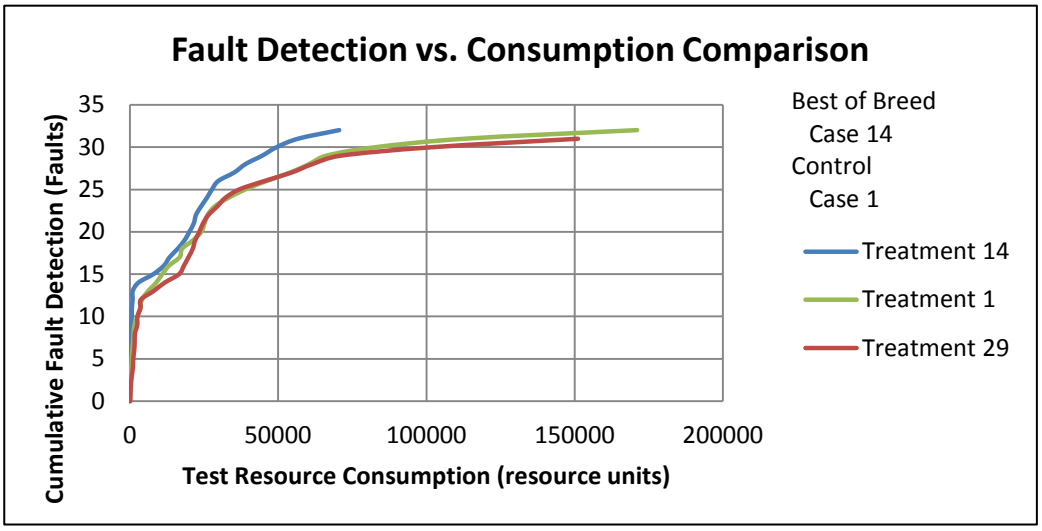


Figure A.50: Cumulative fault detection comparison, treatments, 1, 14, and 29

Table A.25: Compiled DOE results for treatment number 29

Treatment # 29									
Treatment Order 00111	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	453	387	469	480	510	403	496	460	457
Average Resource Count	22904	19812	23610	24156	25712	20448	25054	23313	23126
Resource Count at Last Fault	134832	78288	137184	142080	125088	93936	131952	126432	121224
No. of undetected faults	1	0	0	0	0	0	0	1	
Total Resource Consumption	175056							175008	
Total Test Case	3571							3570	
Adjusted Ave TC Count	573	387	469	480	510	403	496	580	487
Adjusted Ave Res Count	28753	19812	23610	24156	25712	20448	25054	29147	24587
Adjusted Res Count Last Fault	210067	78288	137184	142080	125088	93936	131952	210010	141076

Summary

After repeating the treatment eight times a total of 2 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 141076 resource units out of a possible 175000. This treatment ranked 22nd out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 24587 resource units. This treatment ranked 19th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 487. This treatment ranked 21st out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 19th out of 32.

Comparing treatments, combination number 29 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

Treatment 30 (10111) Results

Treatment 30 is assigned the high signal values for all five variables in the designed experiment except the second (risk prioritization) position.

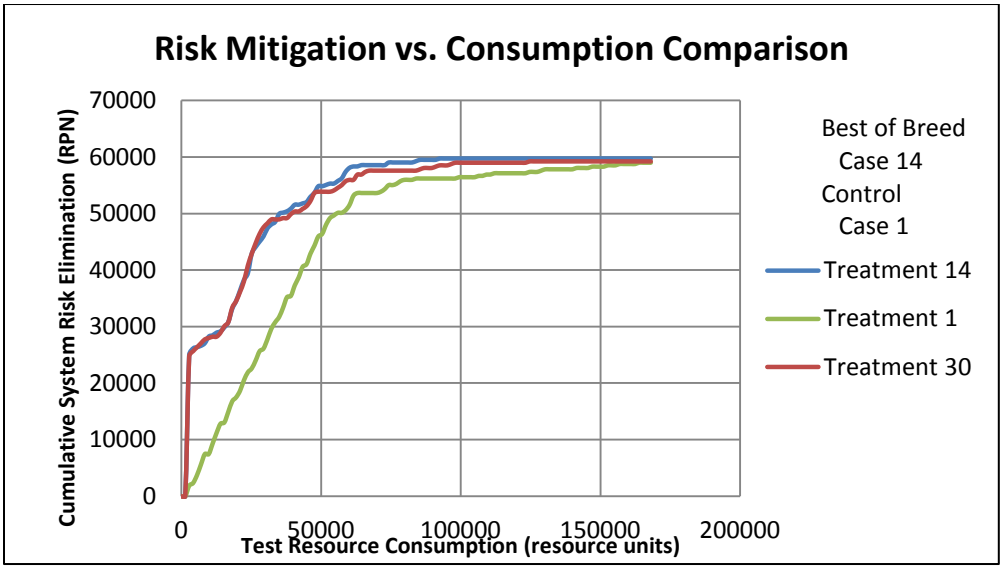


Figure A.51: Risk mitigation comparison, treatments 1, 14, and 30

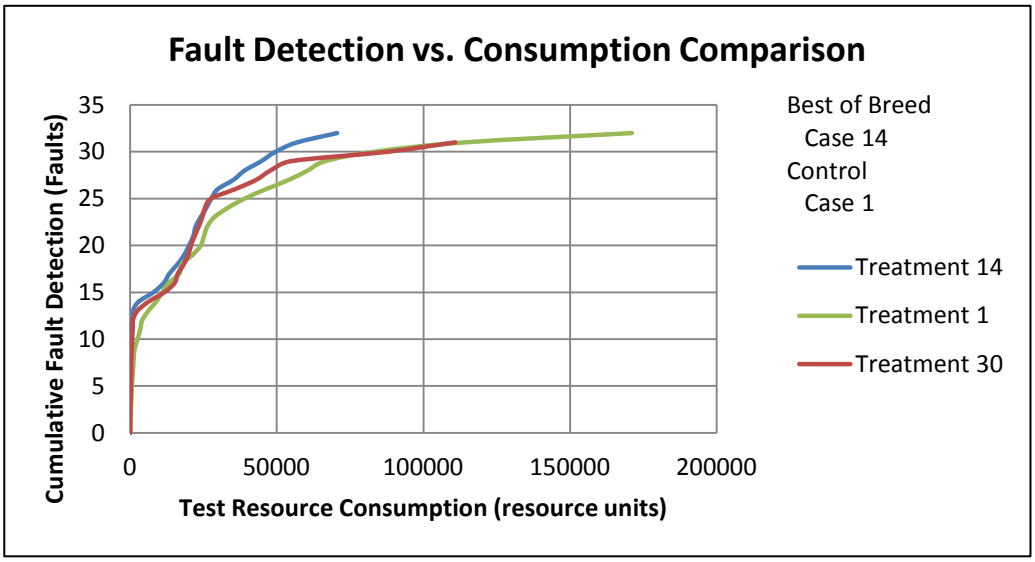


Figure A.52: Cumulative fault detection comparison, treatments 1, 14, and 30

Table A.26: Compiled DOE results for treatment number 30

Treatment # 30									
Treatment Order 10111	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	473	490	495	541	501	524	439	568	504
Average Resource Count	17082	17552	17780	19698	18077	19027	15836	15111	17520
Resource Count at Last Fault	89921	83302	88750	92276	94594	122565	60922	61484	86727
No. of undetected faults	0	0	0	0	0	0	0	2	
Total Resource Consumption	0							175063	
Total Test Case	0							4609	
Adjusted Ave TC Count	473	490	495	541	501	524	439	878	543
Adjusted Ave Res Count	17082	17552	17780	19698	18077	19027	15836	27296	19044
Adjusted Res Count Last Fault	89921	83302	88750	92276	94594	122565	60922	245088	109677

Summary

After repeating the treatment eight times a total of 2 faults were left undetected in the simulation.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 109677 resource units out of a possible 175000. This treatment ranked 18th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 19044 resource units. This treatment ranked 18th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 543. This treatment ranked 22nd out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 18th out of 32.

Comparing treatments, combination number 30 shows quick risk mitigation growth, ahead of treatment number 1 but simply takes an average of 65% more resource per run to detect the last fault. One caution in interpreting the results of this treatment focuses on the two missed faults occurred in the same run.

Treatment 31 (01111) Results

Treatment 31 is assigned the high signal values for all five variables in the designed experiment except the first (cost) position.

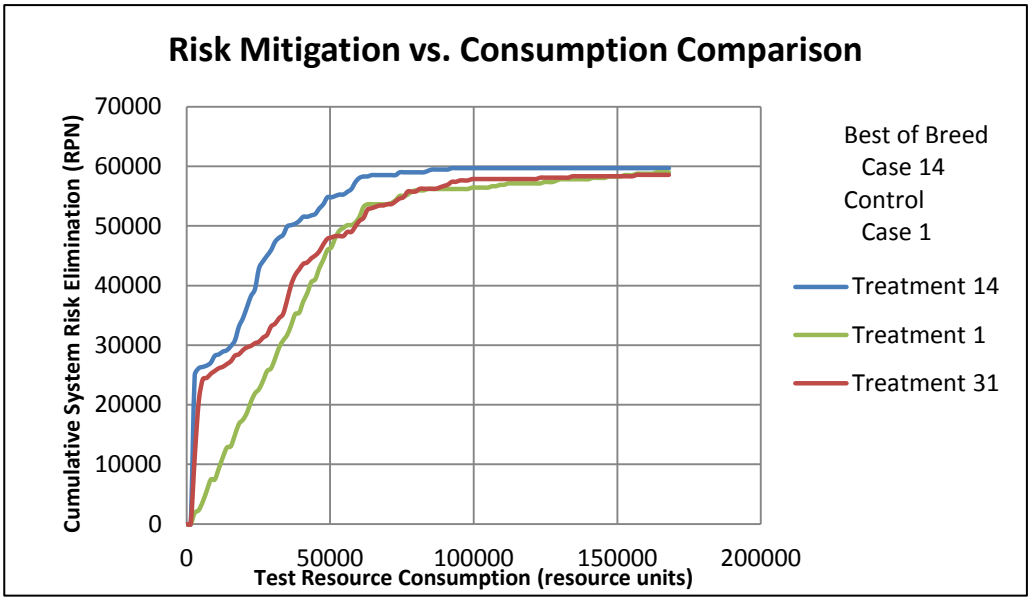


Figure A.53: Risk mitigation comparison, treatments 1, 14, and 31

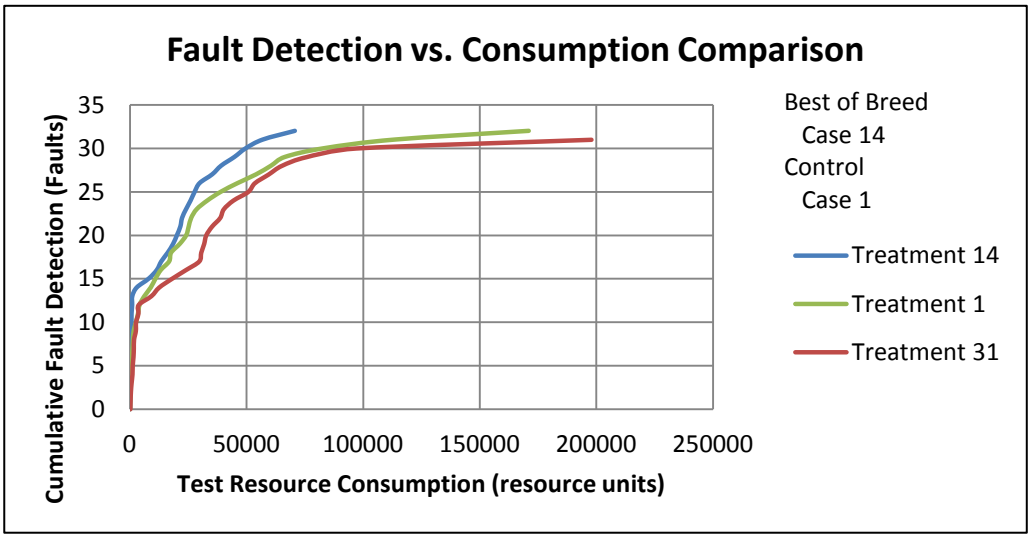


Figure A.54: Cumulative fault detection comparison, treatments 1, 14, and 2

Table A.27: Compiled DOE results for treatment number 31

Treatment # 31									
Treatment Order 01111	Run #								
	1	2	3	4	5	6	7	8	Average
Average Test Case Count	494	535	475	452	605	479	521	548	514
Average Resource Count	24920	26913	23980	22743	30159	24051	26067	27341	25772
Resource Count at Last Fault	89184	88176	75648	90432	155232	93744	97344	121008	101346
No. of undetected faults	0	0	1	1	0	1	1	1	
Total Resource Consumption			175104	175104		175056	171360	175104	
Total Test Case			3572	3572		3571	3570	3571	
Adjusted Ave TC Count	494	535	594	572	605	598	639	665	588
Adjusted Ave Res Count	24920	26913	29797	28599	30159	29864	31678	33053	29373
Adjusted Res Count Last Fault	89184	88176	210125	210125	155232	210067	205632	210125	172333

Summary

After repeating the treatment eight times a total of 5 faults were left undetected in the simulation. The search process can become too aggressive.

The average amount of resources consumed to detect the last fault in the complex system simulator after eight treatment runs was 172333 resource units out of a possible 175000. This treatment ranked 30th out of the 32 treatments in this category.

The average of the amount of resources consumed at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 9373 resource units. This treatment ranked 30th out of the 32 treatments in this category.

The average of the amount of test cases required at the point of their discovery, for all of the faults in the complex system simulator after eight treatment runs was 588. This treatment ranked 30th out of the 32 treatments in this category.

The overall ranking of this treatment relative to the weighted average of the three priorities is 30th out of 32. The variable combination in this treatment is less effective than treatment number 1.

Comparing treatments, combination number 31 shows quick risk mitigation growth, ahead of treatment number 1 but soon falls behind due to the slow process of detecting faults in the early stages.

References

Abbott, T. (1988). *"Improving the Product Development Process" Case Studies of Engineering and Manufacturing for Superior Results.* ASME short course material.

Akao, Y. (2004). *Quality function deployment: integrating customer requirements into product design*. Productivity Press.

Albrecht, A. J. (1979, October). Measuring application development productivity. In *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium,* 10: 83-92.

Amaral, L. A. N., & Uzzi, B. (2007). Complex systems-a new paradigm for the integrative study of management, physical, and technological systems. *Management Science*, *53*(7): 1033-1035.

Anderson, D. A. (2013). *Environmental Economics & Natural Resource Management 4th Edition*. Routledge.

Arcuri, A. (2009). *Automatic software generation and improvement through search based techniques.* Doctoral dissertation, University of Birmingham.

Arcuri, A., Iqbal, M. Z., & Briand, L. (2010). Black-box system testing of real-time embedded systems using random and search-based testing. In *Testing Software and Systems*. Springer Berlin Heidelberg: 95-110.

Aström, K. J., & Murray, R. M. (2010). *Feedback systems: an introduction for scientists and engineers.* Princeton University Press.

Autodesk (2009). Autodesk/ASME Sustainability Survey Results, Available online: http://images.autodesk.com/adsk/files/asme_surveyresults2.pdf (accessed on 10 March 2013).

Bace, J., & Rozwell, C. (2006). Understanding the components of compliance. *Gartner Research Paper 145.*

Bach, J. (2004). Exploratory Testing, In *The Testing Practitioner*, 2nd Edition. Veenendaal, Bosch, UTN Publishers: 253-265.

Badurdeen, F., Iyengar, D., Goldsby, T. J., Metta, H., Gupta, S., & Jawahir, I. S. (2009). Extending total life-cycle thinking to sustainable supply chain design. *International Journal of Product Lifecycle Management*, *4*(1): 49-67.

Bahill, A. T., & Gissing, B. (1998). Re-evaluating systems engineering concepts using systems thinking. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *28*(4): 516-527.

Battiti, R., & Tecchiolli, G. (1994). The reactive tabu search. *ORSA Journal on Computing,* 6(2): 126-140.

Baudry, B., Fleurey, F., Jézéquel, J. M., & Le Traon, Y. (2005). From genetic to bacteriological algorithms for mutation-based testing. *Software Testing, Verification and Reliability,* 15(2): 73-96.

Berger, P. D., & Nasr, N. I. (1998). Customer lifetime value: marketing models and applications. *Journal of Interactive Marketing*, *12*(1): 17-30.

Blanco, R., Tuya, J., & Adenso-Díaz, B. (2009). Automated test data generation using a scatter search approach*. Information and Software Technology*, 51(4): 708-720.

Braun, M., & Schweidel, D. A. (2011). Modeling customer lifetimes with multiple causes of churn. *Marketing Science,* 30(5): 881-902.

Brown, C., & Miller, S. (2008). The impacts of local markets: a review of research on farmers markets and community supported agriculture (CSA). *American Journal of Agricultural Economics,* 90(5): 1298-1302.

Brown, R., Webber, C., & Koomey, J. G. (2002). Status and future directions of the ENERGY STAR program. *Energy,* 27(5): 505-520.

Brownlie, R., Prowse, J., & Phadke, M. S. (1992). Robust Testing of AT&T PMX/StarMAIL Using Oats*. AT&T Technical Journal*, 71(3): 41-47.

Bunkley, N. (2011). *Ford's High Tech Control System Hurts Ranking for Quality* . NY Times . Available online: http://www.nytimes.com/2011/06/24/business/24ford.html?_r=0 (accessed on 11 November 2012).

Burton, I. (1987). Report on Reports: Our Common Future: The World Commission on Environment and Development. *Environment: Science and Policy for Sustainable Development*, *29*(5): 25-29.

Cameron, K. S. (2006). *Competing values leadership: Creating value in organizations*. Edward Elgar Publishing.

Carrubba, E. R., & Gordon, D. R (1988). *Product assurance principles: integrating design assurance and quality assurance.* McGraw-Hill.

Chan, L. K., & Wu, M. L. (2002). Quality function deployment: a literature review. *European Journal of Operational Research*, 143(3): 463-497.

Chen, A., Dietrich, K. N., Huo, X., & Ho, S. M. (2010). Developmental neurotoxicants in e-waste: an emerging health concern. *Environmental Health Perspectives*, *119*(4): 431-438.

Chen, Y., Probert, R. L., & Sims, D. P. (2002, September). Specification-based regression test selection with risk analysis*. In Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research*. IBM Press.

Clausing, J. R. & Clausing, D. (1988). The House of Quality. *Harvard Business Review*, May-June: 63-73.

Cohen, G. (2010). *Agile Excellence for Product Managers: A Guide to Creating Winning Products with Agile Development Teams*. Happy About Publishing.

Cohen, J., Ferguson, R., & Hayes, W. (2013). *White Paper: A Defect Prioritization Method Based on the Risk Priority Number*. Carnegie-Mellon University Pittsburg PA Software Engineering Institute.

Cohen, M. B., Gibbons, P. B., Mugridge, W. B., & Colbourn, C. J. (2003, May). Constructing test suites for interaction testing. In *Proceedings of 25th International Conference on Software Engineering, 2003.* IEEE: 38-48

Colton, C. C. (1824). *Lacon, Or, Many Things in a Few Words: Addressed to Those who Think*, new edition. Longman, Orme, Brown, Green & Longmans. Paterson-Row, London: 113.

Crosson, S., Needles, B. E., (2008). *Managerial Accounting (8th Ed*). Boston: Houghton Mifflin Company.

Crow, L. H. (1977). Confidence interval procedures for reliability growth analysis. *Army Material Systems Analysis Activity.* Aberdeen Proving Ground MD.

Crow, L. H. (1982). Confidence interval procedures for the Weibull process with applications to reliability growth. *Technometrics,* 24(1): 67-72.

Crowe, D., & Feinberg, A. (1998). Stage-gating accelerated reliability growth in an industrial environment. In *Annual Technical Meeting-Institute of Environmental Sciences and Technology*, 44: 246-254.

Crowe, D., & Feinberg, A. (Eds.). (2001). *Design for Reliability*. CRC Press.

Department of Defense, (1949). *Procedures for performing a failure mode effect and critical analysis.* US Department of Defense: MIL-P-1629.

Devanathan, S., Ramanujan, D., Bernstein, W. Z., Zhao, F., & Ramani, K. (2010). Integration of sustainability into early design through the function impact matrix. *Journal of Mechanical Design,* 132(8): 1-8.

Duane, J. T. (1964). Learning curve approach to reliability monitoring. In *IEEE Transactions on Aerospace,* 2(2): 563-566.

Dupont, S. (2010). Digital diplomacy. *Foreign Policy.* Retrieved from: http://www.nytimes.com/2011/06/24/business/24ford.html?_r=0 (accessed on 11 November 2012).

Eccles, R. G., Ioannou, I., & Serafeim, G. (2012). *The impact of a corporate culture of sustainability on corporate behavior and performance.* National Bureau of Economic Research.

Elkington, J. (2004). Enter the triple bottom line. In Adrian Henriques and Julie Richardsons (eds) The Tripple Bottom Line: Does It All Add Up. *Earthscan*: 1-16.

Ellram, L. (1993). Total cost of ownership: elements and implementation. *Journal of Supply Chain Management,* 29(4): 2-11.

ElMaraghy, W., ElMaraghy, H., Tomiyama, T., & Monostori, L. (2012). Complexity in engineering design and manufacturing. *CIRP Annals-Manufacturing Technology*, *61*(2): 793-814.

Eppinger, S. D., & Browning, T. R. (2012). *Design structure matrix methods and applications.* MIT Press.

Erat, S., & Kavadias, S. (2008). Sequential testing of product designs: Implications for learning. *Management Science*, 54(5): 956-968.

Fader, P. S., & Hardie, B. G. (2007). How to project customer retention. *Journal of Interactive Marketing,* 21(1): 76-90.

Fader, P. S., & Hardie, B. G. (2009). Probability models for customer-base analysis. *Journal of Interactive Marketing,* 23(1): 61-69.

Fenton, N., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause, P., & Mishra, R. (2007). Predicting software defects in varying development lifecycles using Bayesian nets. *Information and Software Technology*, 49(1): 32-43.

Fiksel, J. (2006). Sustainability and resilience: toward a systems approach. *Sustainability: Science Practice and Policy*, *2*(2): 14-21.

Forrester, J. W. (1971). *World Dynamics*. Wright-Allen Press Inc., Cambridge.

Friedman, M. (2007). *The social responsibility of business is to increase its profits*. Springer, Berlin: 173-178.

Friedman, T. L. (2005). *The World is Flat A Brief History of the Twenty-First Century*. New York: Farrar, Straus and Friedman.

Gehin, A., Zwolinski, P., & Brissaud, D. (2008). A tool to implement sustainable end-of-life strategies in the product development phase. *Journal of Cleaner Production*, 16(5): 566-576.

Glover, F. (1990). Tabu Search—part II. *ORSA Journal on Computing*, 2(1): 4-32.

Godefroid, P., & Khurshid, S. (2002). Exploring very large state spaces using genetic algorithms. In *Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg: 266-280.

Godefroid, P., Klarlund, N., & Sen, K. (2005, June). DART: directed automated random testing. In *ACM Sigplan Notices.* ACM, 40(6): 213-223.

Gorzelany. (2012, December 29). Biggest Auto Recalls Of 2012 (And Why They Haven't Affected New-Car. *Forbes Magazine*, *LIFESTYLE*. Retrieved from: http://www.forbes.com/sites/jimgorzelany/2012/12/29/biggest-auto-recalls-of-2012/ (accessed on 10 March 2013).

Hauser, J. R. (1993). How Puritan-Bennett used the house of quality. *Sloan Management Review*, 34(3): 61-70.

Hemmelskamp, J., & Brockmann, K. L. (1997). Environmental labels—the German 'Blue Angel'. *Futures*, 29(1): 67-76.

House, C. H., & Price, R. L. (1990). The return map: tracking product teams. *Harvard Business Review*, 69(1): 92-100.

IEEE. (2002). *Software Quality Assurance Planning - Standard 730-2002*. IEEE.

Jawahir, I. S., & Dillon, O. W. (2007, October). Sustainable manufacturing processes: new challenges for developing predictive models and optimization techniques. In

Proceedings of the First International Conference on Sustainable Manufacturing. Montreal, Canada.

Jawahir, I. S., Rouch, K. E., Dillon, O. W., Holloway, L., & Hall, A. (2007). Design for sustainability (DFS): new challenges in developing and implementing a curriculum for next generation design and manufacturing engineers*. International Journal of Engineering Education*, 23(6): 1053-1064.

Jawahir, I. S., & Jayal, A. D. (2011). Product and Process Innovation for Modeling of Sustainable Machining Processes*. In Advances in Sustainable Manufacturing*. Springer Berlin Heidelberg: 301-307.

Jayal, A. D., Badurdeen, F., Dillon Jr, O. W., & Jawahir, I. S. (2010). Sustainable manufacturing: Modeling and optimization challenges at the product, process and system levels*. CIRP Journal of Manufacturing Science and Technology*, 2(3): 144-152.

Jensen, M. C., & Meckling, W. H. (1976). Theory of the firm: Managerial behavior, agency costs and ownership structure*. Journal of Financial Economics*, 3(4): 305-360.

Jervan, G., Eles, P., & Peng, Z. (1999, May). A Uniform Test Generation Technique for Hardware/Software Systems. In *IEEE European Test Workshop (ETW99)*.

Jones, C. (2008). Measuring defect potentials and defect removal efficiency. *CrossTalk The Journal of Defense Software Engineering*, 21(6): 11-13.

Joshi, K., Venkatachalam, A., & Jawahir, I. S. (2006, October). A new methodology for transforming 3R concept into 6R concept for improved product sustainability. In *IV Global Conference on Sustainable Product Development and Life Cycle Engineering:* 3-6.

Kaner, C., Bach, J., & Pettichord, B. (2008*). Lessons learned in software testing*. John Wiley & Sons.

Kennedy, M. N., & Ward, A. (2003). *Product development for the lean enterprise: why Toyota's system is four times more productive and how you can implement it*. Oaklea Press.

Kerscher, W. J. I. I. I. (1993, January). System assurance at AC Rochester. In *Proceedings of* Annual *Reliability and Maintainability Symposium, 1993.* IEEE: 71-76

Kirk, D. E. (2012). *Optimal control theory: an introduction*. Dover Publications.

Kiron, D., Kruschwitz, N., Reeves, M., & Goh, E. (2013). The benefits of sustainability-driven innovation. *MIT Sloan Management Review*, *54*(2): 69-73.

Kosorukoff, A. (2001). Human based genetic algorithm. In *2001 IEEE International Conference on Systems, Man, and Cybernetics:* 3464-3469.

Krishnan, R., Krishna, S. M., & Nandhan, P. S. (2007). Combinatorial testing: learnings from our experience. *ACM SIGSOFT Software Engineering Notes*, 32(3): 1-8.

Kuhn, D. R., & Reilly, M. J. (2002, December). An investigation of the applicability of design of experiments to software testing*. In Proceedings of 27th Annual NASA Goddard Software Engineering Workshop, 2002.* IEEE: 91-95.

Kuhn, D. R., Wallace, D. R., & AM Gallo, J. (2004). Software fault interactions and implications for software testing. In *IEEE Transactions on Software Engineering,* 30(6): 418-421.

Laszlo, C. (2008). *Sustainable value: How the world's leading companies are doing well by doing good*. Stanford University Press.

Lazic, L., & Mastorakis, N. (2008). Orthogonal array application for optimal combination of software defect detection techniques choices. *WSEAS Transactions on Computers*, 7(8): 1319-1336.

Lefticaru, R., & Ipate, F. (2008, April). Functional search-based testing from state machines. In *2008 1st International Conference on Software Testing, Verification, and Validation.* IEEE: 525-528.

Lessig, L. (2002). *The future of ideas: The fate of the commons in a connected world*. Random House LLC.

Linnenluecke, M. K., & Griffiths, A. (2010). Corporate sustainability and organizational culture*. Journal of World Business*, 45(4): 357-366.

Little, T. A. (2011). *Robust Optimization & Tolerance Design*. Design for Six Sigma Training Material. Thomas A. Little Consulting, TLC.

Ma, Y. S., Chen, G., & Thimm, G. (2008). Paradigm shift: unified and associative feature-based concurrent and collaborative engineering. *Journal of Intelligent Manufacturing*, 19(6): 625-641.

Madachy, R., Boehm, B., Richardson, J., Feather, M., & Menzies, T. (2007). Value-Based Design of Software V&V Processes for NASA Flight Projects*. In Proceedings from AIAA Space 2007 Conference.*

Malik, K. (2013). Human Development Report 2013. The rise of the South: Human progress in a diverse world. Available at: http://ssrn.com/abstract=2294673.

Martin, J. N. (2000). Processes for engineering a system: an overview of the ANSI/EIA 632 standard and its heritage. *Systems Engineering*, *3*(1): 1-26.

Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*. Prentice Hall PTR.

McDonough, W., & Braungart, M. (2002). Design for the triple top line: new tools for sustainable commerce. *Corporate Environmental Strategy*, *9*(3): 251-258.

McMinn, P. (2004). Search-based software test data generation: a survey. *Software Testing, Verification and Reliability*, *14*(2): 105-156.

Meadows, D. H., L., M. D., Randers, J., & Behrens, W. W. (1972). *The Limits to Growth*. Universe Books.

Metta, H., (2011). *A multi-stage decision support model for coordinated sustainable product and supply chain design.* Lexington KY: University of Kentucky Doctoral Dissertation.

Moore, G. E. (2006). Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp. 114 ff. *Solid-State Circuits Society Newsletter,* IEEE, 11(5): 33-35.

Morgan, J. M., & Liker, J. K. (2006). *The Toyota product development system*. New York.

Mulgan, G. (2011). *Connexity: How to live in a connected world.* Random House.

Navarro, P. (1988). Why do corporations give to charity?. *Journal of Business*, 61: 65-93.

Nguyen, C. D., Perini, A., Tonella, P., & Kessler, F. B. (2007, December). Automated continuous testing of multi-agent systems. In *The Fifth European Workshop on Multi-Agent Systems*.

NIST/SEMATECH. (2013) e-Handbook of Statistical Methods. Retrieved from: http://www.itl.nist.gov/div898/ (accessed on 10 March 2013).

Pargas, R. P., Harrold, M. J., & Peck, R. R. (1999). Test-data generation using genetic algorithms. *Software Testing Verification and Reliability*, 9(4): 263-282.

Peccei. (1981). *One Hundred Pages for the Future: Reflections of the President of the Club of Rome.* Pergamon Press. New York.

Perrow, C. (1999). Organizing to reduce the vulnerabilities of complexity. *Journal of Contingencies and Crisis Management*, *7*(3): 150-155.

Prasad, B. (1998). Review of QFD and related deployment techniques. *Journal of Manufacturing Systems*, *17*(3): 221-234.

Pugh, S., & Clausing, D. (1996). *Creating Innovtive Products Using Total Design: The Living Legacy of Stuart Pugh*. Addison-Wesley Longman Publishing Co., Inc..

Ramani, K., Ramanujan, D., Bernstein, W. Z., Zhao, F., Sutherland, J., Handwerker, C., & Thurston, D. (2010). Integrated sustainable life cycle design: a review. *Journal of Mechanical Design*, *132*(9): 1-15.

Reinartz, W. J., & Kumar, V. (2003). The impact of customer relationship characteristics on profitable lifetime duration. *Journal of Marketing*, *67*(1): 77-99.

Rosen, M. A. (2013). Engineering and Sustainability: Attitudes and Actions. *Sustainability*, *5*(1): 372-386.

Rosset, S., & Neumann, E. (2003, November). Integrating Customer Value Considerations into Predictive Modeling. In *Proceedings from the Third IEEE International Conference on Data Mining:* 283-290.

Schmalensee, R. (2004). Sunk costs and antitrust barriers to entry. *American Economic Review Papers and Proceedings*, 69(2): 471-475.

Schwartz, B. (2009). *The paradox of choice*. HarperCollins. New York.

Schweidel, D. A., Fader, P. S., & Bradlow, E. T. (2008). Understanding service retention within and across cohorts using limited information. *Journal of Marketing*, 72(1): 82-94.

Seevers, K. D., Badurdeen, F., & Jawahir, I. S. (2013). Sustainable value creation through innovative product design. In *Proceedings of 11th Global Conference on Sustainable Manufacturing, CIRP.* Berlin.

Shapiro, B. (2002). Want a happy customer? Coordinate sales and marketing. Boston: *Harvard Business School.* Available online: http://hbswk.hbs.edu/item/3154.html (accessed on 10 March 2013).

Sharma, A., Jadhav, A., Srivastava, P. R., & Goyal, R. (2010). Test cost optimization using tabu search. *Journal of Software Engineering and Applications*, *3*(05): 477-486.

Smith, P. G. & Reinertsen, D. G. (1991). *Developing products in half the time*. New York: Van Nostrand Reinhold.

Srikanth, H., Banerjee, S., Williams, L., & Osborne, J. (2014). Towards the prioritization of system test cases. *Software Testing, Verification and Reliability*, *24*(4): 320-337.

Sthamer, H. H. (1995). *The automatic generation of software test data using genetic algorithms.* Doctoral dissertation, University of Glamorgan.

Strauch, B. (2004). Investigating Human Error: Incidents, Accidents, and Complex Systems. *Aviation, Space, and Environmental Medicine*, *75*(4): 372-372.

Tolio, T., Ceglarek, D., ElMaraghy, H. A., Fischer, A., Hu, S. J., Laperrière, L., ... & Váncza, J. (2010). SPECIES—Co-evolution of products, processes and production systems. *CIRP Annals-Manufacturing Technology*, 59(2): 672-693.

Taguchi, G., Clausing, D., & Watanabe, L. T. (1987). *System of experimental design: engineering methods to optimize quality and minimize costs*. White Plains, NY: UNIPUB/Kraus International Publications.

Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing*. National Institute of Standards and Technology*, RTI Project: 7007(011).

Thakor, A. V., DeGraff, J., & Quinn, R. (2000). Creating sustained shareholder value—and dispelling some myths. *Financial Times, Mastering strategy: The Complete MBA Companion in Strategy*, Pearson Education, Harlow.

Ueda, K., Takenaka, T., Váncza, J., & Monostori, L. (2009). Value creation and decision-making in sustainable society. *CIRP Annals-Manufacturing Technology*, 58(2): 681-700.

Vágási, M., Szalkai, Z., & Jankó, Á. (2003). Sustainable customer relationship. In *Proceedings of 5th International Summer Academy on Technology Studies Corporate Sustainability,* Deutschlandsberg*, Austri*a: 289-302.

Valdes-Dapena, P. (2011). Honda recalling 1.5 million cars. CNN Money. Retrieved from: http://money.cnn.com/2011/08/05/autos/honda_recall (accessed on 10 November 2012).

Watkins, A., Berndt, D., Aebischer, K., Fisher, J., & Johnson, L. (2004, January). Breeding software test cases for complex systems. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences.*

Weisstein, E. W. (2014). "n-Tuple.". From MathWorld--A Wolfram Web Resource. Retrieved from: http://mathworld.wolfram.com/n-Tuple.html (accessed on 10 March 2014).

Whitehead, J. C., & Haab T. C., ECON 101: Negative Externality. *Environmental Economics Blog:* Retrieved from: http://www.env-econ.net/negative-externality.html (accessed on 10 March 2014).

Widmer, R., Oswald-Krapf, H., Sinha-Khetriwal, D., Schnellmann, M., & Böni, H. (2005). Global perspectives on e-waste. *Environmental Impact Assessment Review*, *25*(5): 436-458.

Williams, E. D., Ayres, R. U., & Heller, M. (2002). The 1.7 kilogram microchip: Energy and material use in the production of semiconductor devices. *Environmental Science & Technology*, 36(24): 5504-5510.

Wilson, C. C., Kennedy, M. E., & Trammell, C. J. (1996). *Superior product development: managing the process for innovative products*. Blackwell.

Wolff, M. F. (1996). 'Green Wall' hurts environment management. *Research-Technology Management*, *39*(2): 5-6.

Womack, J. P., Jones, D. T., & Roos, D. (2007). *The machine that changed the world: The story of lean production--Toyota's secret weapon in the global car wars that is now revolutionizing world industry*. Simon and Schuster.

Yamaji, M., & Amasaka, K. (2011). New Japan quality management model: implementation of new JIT for strategic management technology. *Journal of International Business & Economics Research*, 7(3): 107-114.

VITA

Name

K. Daniel Seevers

Place of Birth

St. Louis, Missouri

Education

B.S. Mechanical Engineering, Missouri University of Science & Technology (formally University of Missouri - Rolla)

M.A. Patterson School of Diplomacy and International Commerce, University of Kentucky

M. S. Manufacturing Technology, Eastern Kentucky University

Professional Positions

Development Engineer, IBM

Sr. Manager, Product Development, Lexmark International Inc.

Director of Operations, Lexmark International Inc.

Director of Product and Process Assurance, Lexmark International Inc.

Director of Development Operations, Lexmark International Inc.

Honors/Awards

Eagle Scout

National Distinguish Service Award, Order of the Arrow, Boy Scouts of America

William T. Hornaday Metal, for distinguished service to natural resource conservation

Publications & Patents

U.S. Patent No. 5215012 A: Ribbon Cartridge for Printers

U.S. Patent No. 5926673 A: Driving mechanism for photosensitive image bearing drum in electrophotographic machines

U.S. Patent No. 6397026 B1: Apparatus and methods for increasing bias force on opposing photosensitive member and developing means