Reasoning about System-Level Failure Behavior from Large Sets of Function-Based Simulations

Citation	Jensen, D. C., Bello, O., Hoyle, C., & Tumer, I. Y. (2014). Reasoning about system-level failure behavior from large sets of function-based simulations. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 28(04), 385-398. doi:10.1017/S0890060414000547
DOI	10.1017/S0890060414000547
Publisher	Cambridge University Press
Version	Accepted Manuscript
Terms of Use	http://cdss.library.oregonstate.edu/sa-termsofuse

Reasoning about System-Level Failure Behavior from Large Sets of Function-Based Simulations

David C. Jensen
Oladapo Bello
email:dcjensen@uark.edu
phone: 479-575-5679
Mechanical Engineering
University of Arkansas
Fayetteville, Arkansas, USA

Christopher Hoyle
Irem Y. Tumer
School of Mechanical, Industrial and Manufacturing Engineering
Oregon State University
Corvallis, Oregon 97331

Pages: 31
Figures: 4
Tables: 6

Reasoning about System-Level Failure Behavior from Large Sets of Function-Based Simulations

Abstract

This paper presents the use of data clustering methods applied to the analysis results of a design-stage, functional failure reasoning tool. A system simulation using qualitative descriptions of component behaviors and a functional reasoning tool are used to identify the functional impact of a large set of potential single and multiple fault scenarios. The impact of each scenario is collected as the set of categorical function "health" states for each component-level function in the system. This data represents the space of potential system states. The clustering and statistical tools presented in this paper are used to identify patterns in this system state space. These patterns reflect the underlying emergent failure behavior of the system. Specifically two data analysis tools are presented and compared. First, a modified k-means clustering algorithm is used with a distance metric of functional effect similarity. Second, a statistical approach known as Latent Class Analysis is used to find an underlying probability model of potential system failure states. These tools are used to reason about how the system responds to complex fault scenarios and assist in identifying potential design changes for fault mitigation. As computational power increases, the ability to reason with large sets of data becomes as critical as the analysis methods used to collect that data. The goal of this work is to provide complex system designers with a means of using early design simulation data to identify and mitigate potential emergent failure behavior.

Keywords: Fault Reasoning, Complex System Design, Clustering, Emergent Behavior, Big Data

1 Introduction

Unexpected and under analyzed interactions between components and subsystems is a causal factor for many catastrophic system failures. While individual component failure modes can be modeled and predicted, the system level effect of multiple faults across subsystem boundaries (software, hardware, etc) is challenging to identify. Methods of risk-based and safety-centric design have been developed to address this challenge and impact the design decision making process. A few methods use qualitative simulation of component behavior to provide an analysis of the system level impact failures in terms of lost functional capability. Reasoning on the functional effect of failures provides designers with the information needed to understand the potential impact of faults in a risk-informed approach to design.

The challenge of risk assessment at the design stage is the lack of refined system information. Traditional methods of failure and risk analysis rely on statistical failure data and apply methods where expert knowledge of the system is needed to know the impact and path of fault propagation. For this reason, risk assessment traditionally occurs at the validation stage of a well refined design, where specific component failure probabilities and likely fault propagation paths can be defined. However, to achieve the benefits of early risk-based decision making, several methods of failure analysis based on functional system descriptions have been developed. While some of these design-stage methods use historic failure rates associated with the component types or functions to identify risk (Stone et al., 2005; Grantham-Lough et al., 2009) others have used a behavioral approach to determine the potential impact of failures (Krus and Grantham Lough, 2007; Kurtoglu et al., 2008; Huang and Jin, 2008). By including component behavior information in the analysis, these latter approaches can simulate fault propagation and identify the effect of a fault within the context of the designed system.

Early design-stage failure analysis is a powerful decision making tool allowing designers to make changes to decrease the risk of single, multiple or cascading faults (Kurtoglu et al., 2010). However, the functional approach also enables a high degree of failure characterization. Understanding the ways in which a system design can fail provides designers with the information to develop more robust alternatives.

The Function Failure Identification and Propagation (FFIP) framework is one of the methods for assessing the functional impact of faults in the early design stage (Kurtoglu et al., 2008). The results of using an FFIP-based analysis of a design is an evaluation of the state of each function in the system in response to a simulated failure scenario. In previous work these results have been used to evaluate the consequence of different fault scenarios for a system design (Jensen et al., 2009b; Papakonstantinou et al., 2011; Sierla et al., 2012) and for making design decisions based on fault consequence (Kurtoglu et al., 2010). In this work we take a different approach. Instead of looking at the functional effect of a single fault scenario, we reason on the total set of effects from different scenarios to make design decisions.

Identifying the system-level functional effect is key to using potential failure behavior information for design decision making. For example, the decision between using two different technologies should be informed by how faults in those technologies and faults in the rest of the system interacting with that technology affect the mission objectives. For this reason, top-down safety based design methods such as STAMP (Dulac and Leveson, 2005; Leveson, 2011) use the control of undesired system functional states to develop the system architecture requirements. The challenge for top-down methods is providing assurance of completeness in capturing the potential low-level causes that might lead to the undesirable system state. Similarly, with bottom-up simulation methods like FFIP, interpreting the overall system-level functional effect from a composite set of component-level functional effects is challenging for complex systems. For example, an electrical short is simple to simulate and identify the functional effect to that component and other components on the same circuit. However, identifying system-level functions affected by this fault is challenging and usually relies on expert knowledge of the system. We see that there is a connection between the top-down and bottom-up approaches and this paper demonstrates that data analysis and clustering techniques can be used to identify classes of potential system behavior from the simulation at the component level.

To summarize, the overarching objective of this work is to develop a design-stage simulation and analysis tool set that uses simulation data to reason about the functional robustness of systems to potential component faults and fault propagation. This type of approach is intended to enable designers to compare potential system architectures, identify component and subsystem behaviors that leads to undesired system states, and assess the impact of complex fault scenarios. In order to achieve this high-level objective there are three specific objectives that this presented method addresses. These are:

1. Characterize the impacts of a large space of the potential complex failure scenarios. *(In what types of ways does the system fail?)*

2. Identify the system-level importance of the sets of potential system failures. *(What does each type of failure mean in terms of system functionality?)*

3. Determine how this analysis can be used to make system design decisions. *(Can we use this data for a systems view of functional robustness?)*

By addressing the first objective, this method moves beyond single scenarios analysis and begins to develop a system-level characterization based on simulation of component behavior. The result of completing the first objective is distinct types of system failure analogous to failure modes for the system. However, since these are identified through simulation and data analysis, the types of system failure must be related to the system-level functionality. In this way, objective two enables this method to link top-down and bottom-up analysis methods. Finally, the third objective begins to address how this approach can fit within the overall systems design processes.

1.1 Terminology

Due to the need to use terms that are found and defined differently in multiple disciplines the following definitions are intended for this paper.

Component: Any physical, software, or human element in a systems that has nominal and failure behavior.

Fault/Failure Mode: A discrete behavior of a component different from the nominal behaviors.

Fault Scenario: The set of nominal and faulty component modes provided to the system simulation. Each component is in exactly one state at one time.

Flow: The energy, material and signal that connects the functions of the system.

Function: The action a designer intends in the system that affects the flow of material, energy, or signal.

Function Health State: The evaluation of the relationship between a components behavior and its intend function. With the following categories:

Healthy: Function acts on flow as intended.

Degraded: Function acts on flow but not as intended.

Lost: Function does not act on flow.

No Flow: There is no flow on which the function could act. (A type of *Lost*)

System State: The set of health states for all functions resulting from the simulation of one fault scenario.

2 Background

This section discusses the three technical areas used in this paper and presents some detail of the example system. First, we discuss the design-stage fault simulation tool (FFIP) which is the source of the data on which the analysis and clustering methods will be applied. Second, we provide background on the method of clustering data using a k-means algorithm. Finally, we present a categorical data clustering approach for identifying an underlying probabilistic model for the structure of the data, namely, Latent Class Analysis.

2.1 Function-Failure Analysis in Early Design

The Function Failure Identification and Propagation (FFIP) framework (Kurtoglu and Tumer, 2008; Jensen et al., 2008, 2009b; Kurtoglu et al., 2010; Jensen et al., 2009a; Tumer and

Smidts, 2010; Papakonstantinou et al., 2011; Coatanéa et al., 2011; Sierla et al., 2012) was introduced as a design-stage method for reasoning about failures based on the mapping between components, functions, and nominal and off-nominal behavior. The goal of the FFIP method is to identify failure propagation paths by mapping component behavior states to function 'health'. This approach uses simulation to determine fault propagation and fault effect, thus providing the designer with the possibility of analyzing component and interaction failures and reasoning about their effects on the rest of the system. The two main advantages of the FFIP method are: 1) a functional abstraction which allows it to be used in complex systems employing both software and physical components; and, 2) a simulation-based approach allowing analysis of multiple and cascading faults.

An FFIP analysis begins with a functional representation of a system and utilizes the mapping of functions to components in a component structural representation. A system simulation is built following the structural representation. The nominal and faulty behavior of generic components are stored as state machines in a component library. Each state represents a behavioral mode of the component where the qualitative intervals (high, low, etc.) of the input flow attributes are converted to output flow attributes. For example, in the nominal mode of a fuel line the input flow level of fuel is the same as the output. However, in the blockage fault mode, the output flow level is reduced to zero. Finally, the main contribution of the FFIP approach is the Function Failure Logic (FFL) reasoner which relates the input and output attributes of the component simulation to the expected change in the function mapped to those components. The result of the FFIP analysis is an evaluation of the health state of each function in the system. There are four potential health states for a function defined below. These states are based on the concept that a function is the expression of the designer's intent and describes the actions that affect the flows of energy, material and signal in the system.

1. Healthy: The function affects the flow as intended

2. Degraded: The function affects the flow different than intended

3. Lost: The function does not affect the flow
4. No Flow: There is no flow for the function to act on (usually due to an upstream failure)

In FFIP, a failure event is the triggering of one or more component transitions. Based on the behavioral simulation the functional impact is identified by the FFL reasoner. Each event scenario simulated produces one record. These records will be used by the clustering approaches demonstrated in this paper.

2.2 Data Clustering

Separating data into clusters or partitions has been a useful activity in the data mining community to elicit meaning from large data sets (Han et al., 2006). Starting with the classification of human traits and personality in the 1930-40s, clustering analysis continues to be an important tool to enable machine learning. Multiple methods and algorithms have been developed based on different perspectives on the meaning of a cluster (Estivill-Castro, 2002). There are three main approaches to clustering with multiple methods and algorithms supporting them.

Hierarchical clustering assumes that some category or classification captures all the data and that data points can further be sub-classified into more specific groups in a tree structure. In biology, the Linnaeus taxonomy of living things is an example of hierarchical clustering. Hierarchical methods often relate one or more data points by their similarity.

In contrast to hierarchical methods, partitioning methods separate the data space into different clusters without implying a higher level relationship between those clusters. Data points are related based on a measure of the distance between values. Algorithms that implement partitioning identify centroids of the clusters and then group all data points into a predetermined number of clusters based on their distance from that centroid. K-means clustering is one method of data partitioning that evaluates the Euclidean distance between data points (Lloyd, 1982).

Two significant issues of k-means clustering are that the number of clusters must be selected first and that data points may only have membership in one cluster. To address the first issue, heuristic rules such as choosing k based on the square root of half the data set size can provide an initial assessment (Mardia et al., 1980). Evaluation of the correctness of the value of k can be done through heuristic metrics as well. Variations of k-means known as soft or fuzzy clustering methods use a similar approach but instead provide membership percentages.

The third category of data clustering methods is model-based. These methods assume some structure to the data and try to find the correct statistical model to match that structure. Methods in this category use different means of estimating and finding the maximum likelihood of the data fitting the parameters of a statistical model (MacKay, 2003; Pearl, 2000). These methods assume that the reason some data points are related to other data points is due to some unobserved (or latent) variable. Unlike k-means, data points have a probability of being within a particular cluster based on their dependence to that unobserved variable. There are many variations of model-based clustering depending on the form of the data and the likely form of the clusters. For the analysis of function-based failure simulation data, the most appropriate model-based method is Latent Class Analysis. The details of this analysis and the justification for its use in this work are presented next.

2.3 Latent Class Analysis

Social scientists have used the concept of latent classes since the 1950s (Lazarsfeld and Koch, 1959). Manifest (or observed) variables are the data of empirical studies. A latent variable is one not directly tested but is nevertheless correlated to observations of the manifest variables. If the latent variable is continuous then methods such as factor analysis and multivariate mixture estimation can be used to find this structure. However, if the latent variables have discrete categories then the structure fits a latent class model (Vermunt and Magidson, 2004).

As an example, survey questions on personal views of several political topics can form the parameters of a statistical model. Latent class analysis (LCA) on the survey data could be used to identify subgroups into which the respondents are classified. Groups identified within this data would likely correspond to labels like “conservative”,“liberal”, etc. There are three main results from performing an LCA. First, each data point has a probabilistic membership to each class of the latent variable (e.g., the respondent’s likely political leaning). Secondly, each discrete variable state is correlated to a latent class (e.g., liberals have a high probability of answering affirmatively to question three.) The final component of the LCA output is class membership percentages for the entire data set (e.g., 40% conservative, etc.)

Formally, the latent class model is based on the concept that the probability of observing a specific pattern ($\mathbf{Y}$) of manifest variable states $\mathbf{y}$, denoted $P(\mathbf{Y} = \mathbf{y})$, is a weighted average of the C class-specific probabilities $P(\mathbf{Y} = \mathbf{y}|X = x)$, where X is a latent variable with C number of classes. Weighting with the proportion of that class to the latent variable $P(X = x)$ results in Equation 1.

$$P(\mathbf{Y} = \mathbf{y}) = \sum_{x=1}^{C} P(X = x)P(\mathbf{Y} = \mathbf{y}|X = x) \tag{1}$$

Further, the manifest variables within a class, Y_l are assumed to be locally independent. Therefore, Equation 2 defines the probability of observing a pattern in the L manifest variables within a class.

$$P(\mathbf{Y} = \mathbf{y}|X = x) = \prod_{l=1}^{L} P(Y_l = y_l|X = x) \tag{2}$$

Using the political example above, ($\mathbf{Y}$) is the pattern of answers associated with a political group answering the specific questions $\mathbf{y}$. This pattern is independent within each of the discrete political groups in X.

As with k-means data clustering, algorithms for implementing LCA use expectation maximization for a predefined number of groups. Therefore, LCA must be executed iteratively in order to identify the correct number of classes for the latent variables. Identifying the goodness of fit of the latent class model is typically accomplished by examining either the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC). These are metrics to estimate the information entropy (information lost) when a statistical model is used to describe reality. The AIC formulation modifies the log-likelihood estimation by the number of parameters, punishing over-fitting models. The objective in checking goodness of fit with AIC is to find the minimum of Equation 3, where **K** is the number of parameters and **L** the likelihood function for the statistical model. The BIC formulation is similar but accounts for the sample data size.

$$AIC = 2\mathbf{K} - ln(\mathbf{L}) \tag{3}$$

LCA was chosen as a clustering method over other clustering methods because the manifest variables are the discrete health states of each function in the system. Additionally, the hypothesis of this work is that the failure behavior of a system is also categorical. This categorical system-level failure is the latent variable in our analysis. The discrete (and ordinal) nature of the variables rules out other multivariate mixture models.

2.4 Example System

To demonstrate the clustering approaches applied to function failure analysis results, we perform an FFIP analysis on a design concept of an electrical power system (EPS). This example system will be used to simulate numerous fault scenarios, identify the set of functional impacts for each scenario and apply the clustering algorithms to find patterns of system failure behavior. This EPS example is an early design-stage model that uses batteries to provide power for a set of AC and DC loads. This example is based on the design of the Advanced Diagnostic and Prognostic testbed located at the NASA Ames Research Center (Poll, 2007). In previous work, various potential design architectures were compared using a quantified interpretation of the FFIP results (Kurtoglu et al., 2010). The example used in this work expands upon a similar but less complex example (Jensen et al., 2012).

As seen in Figure 1, the concept for the EPS is a fault tolerant software controlled hardware system. At the system level, three operational states are recognized. Specifically, *Nominal*, when both load banks of AC and DC loads are operational; *Degraded*, when only one of the load banks is operational; and *Lost*, when neither load bank is operational. The purpose of the software control is to automatically maintain operation at a nominal state if possible and a degraded state otherwise. By evaluating the voltage levels in both the load banks and both battery banks the controller decides to open or close Relays 1 through 4. The first rule implemented in the software control is that no two batteries can be connected together. For example, Relays 1 and 4 cannot both be closed while there is power available from both batteries or an electrical over current will occur. After this rule, the controller observes the voltage and relay position sensor values to determine which relays to open or close to ensure continued operation. In a fault scenario, the controller can decide to swap power so that the first battery powers the second load and vice versa or simply to shut down one line and run at a degraded state. The control logic is implemented with a truth table where values of sensors correspond to specific relay positions. The control attempts to keep the system in the best operating state as described in Table 1. In this table the term "$Batt1 \rightarrow Load1$" indicates that Battery Bank 1 is powering Load Bank 1.

Table 1: Operational States the Software Control Attempts to Maintain

Nominal		**Degraded**		**Lost**
State 1	State 2	State 3	State 4	State 7
$Batt1 \rightarrow Load1$ $Batt2 \rightarrow Load2$	$Batt1 \rightarrow Load2$ $Batt2 \rightarrow Load1$	$Batt1 \rightarrow Load1$	$Batt2 \rightarrow Load2$	No Action
		State 5	State 6	
		$Batt1 \rightarrow Load2$	$Batt2 \rightarrow Load1$	

This fault tolerant example system enables the identification of high-level system goals such as maintain load operation and illustrates fault propagation over both software and hardware components. This example system is complicated enough to demonstrate the clustering methods yet still provides clarity in the impact of complex faults. The FFIP analysis has also been demonstrated on a more complicated system (nuclear power generation (Papakonstantinou et al., 2011; Sierla et al., 2012)).

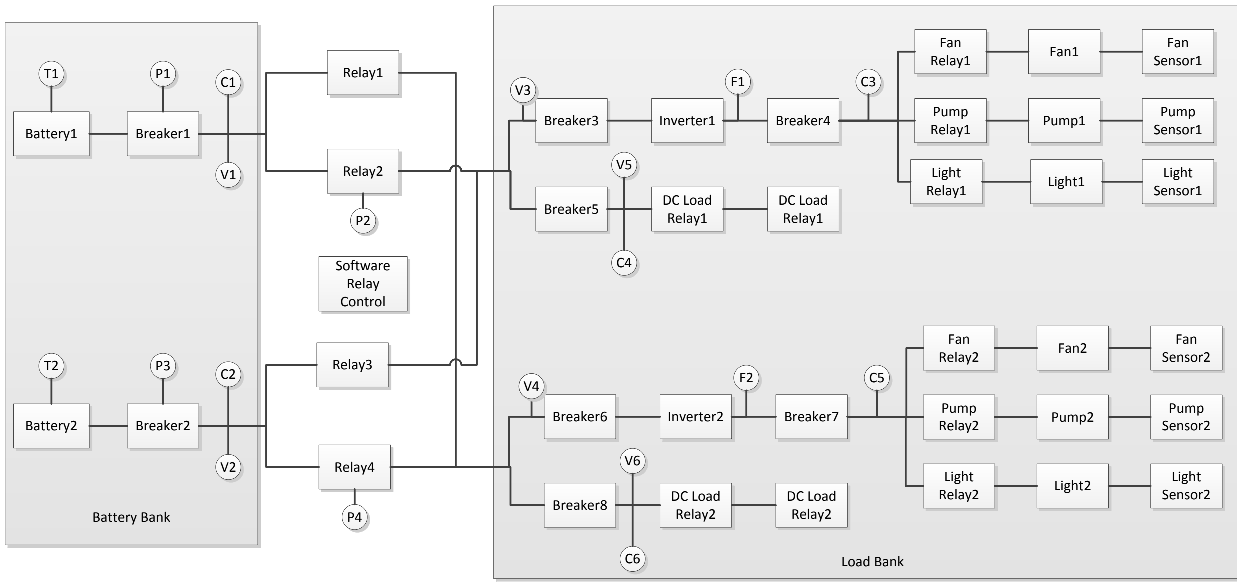


Figure 1: Architecture of the Electrical Power System (EPS) used for function-based failure analysis and reaults clustering.

3 Methods

The development and justification of the functional effect analysis using the FFIP methodology is documented in previous work (Kurtoglu and Tumer, 2008; Jensen et al., 2009a; Kurtoglu et al., 2010; Papakonstantinou et al., 2011; Sierla et al., 2012) and will not be repeated here. Because the motivation of this work is to use data analysis techniques to identify underlying system behavior, we begin with collecting the analysis results from the FFIP-based simulation. Other methods of design analysis and simulation could be used instead. The two things that are needed to apply these techniques is a large number of behaviors to simulate (many scenarios) and multiple data points to describe each scenario. FFIP provides this by the ability to simulate single and multiple fault scenarios as well as variations in flow parameters. Further, for each scenario simulated, the result is the health state of each component-level function in the system. These function health states are the variables that describe the system state in response to the simulated scenario. In the following sections we discuss the simulation and collection of functional effect failure data and the application of the similarity clustering and probabilistic latent class analysis.

3.1 Identifying the Functional Impact of Component Faults and Interactions

The impact of different component fault modes is identified for the EPS using a simulation of the system built by connecting component models created with the Stateflow toolbox in Matlab Simulink. A scenario is simulated where one or more faults is triggered and the resulting changes in system dynamics are allowed to propagate. The output of each simulation is the function health state of each component-level function. For example, one scenario includes triggering the failure behavior for both batteries. To simulate this scenario, the system simulation begins with all components operating nominally. Then after 25 time steps the first battery's operating mode is changed to "Failed-Disconnected." The effects of this change is the loss of current and voltage from that component. After 50 time steps the second battery's operating state is changed in the same way. The effect of these changes is allowed to propagate through the system. In this example, the software controller attempts to switch between sources by changing which relays are closed. Finding no solution that provided power to the loads the software controller defaults to opening all relays as a failure safety measure. After 100 time steps the simulation is ended and the final function health states for each component-level function is recorded as the result for that scenario. The injection of failures at 25 and 50 time steps is arbitrary. Through analysis of numerous simulation it was found that the state machines used need four to eight time steps to reach a steady state. Further, reducing the time between failure mode insertion resulted in no change to the final system state. However, the order of the fault mode changes did affect the final system state results for many scenarios (excluding the one above). Therefore, every order of faults is also simulated. Because this system has 58 component-level functions, the result of simulating a scenario is a vector where each element corresponds to the health state of each of the 58 functions. These function health states are recorded as integers from 1 to 4 to ease data handling.

Using a Matlab script, a large set of scenario results is generated; first simulating each component fault mode as a single fault scenario and then two fault combinations. Three or more fault scenarios can also be generated in the same manner. While simulating three or more scenarios is possible, for this example system the limited number of components resulted in few unique system states for more than two failure scenarios. For this system, simulating every possible combination of two faults is not computationally expensive. However, for more complex systems there are three possible ways for guiding the scenario selection and simulation process. First, expert knowledge can provide direction on the components that are likely to negatively interact and have known fault causation or simply using proximity. An alternative to this approach is simulating fault modes based on the relationship between causes and symptoms of faults (Jensen et al., 2009b). This latter approach is based on triggering failure modes in components with fault symptoms (e.g. leaking) which are of the same type as fault causes (e.g. exposure to liquid). Finally, the clusters generated using the approach may provide guidance in identifying fault modes that should be simulated together in an iterative approach.

Function failure analysis results are collected from each scenario in a matrix where each row is a separate scenario and the columns correspond to the resulting identified health state of the component functions. For the clustering analysis, three sets of scenarios were generated. The first set of results tested each failure mode of each component resulting in 193 simulations. The second and third set of scenarios tested two fault scenarios. The difference between these last two sets was a reversal of the order in which the faults where tested (e.g., battery fault then relay fault, and reversed order in the third set). For the three sets this generated 37,299 fault simulation records. Both fault orderings were included because it is possible that the order of faults may change the system level effects.

3.2 Pre-processing to Enhance Clustering Effectiveness

The clustering methods demonstrated in this work are applied to find similarities and structure between different fault scenarios. However, the first level of grouping is to identify which fault scenarios resulted in identical functional results. These represent scenarios that cannot be functionally distinguished from each other. For example, faults in two loads that both cause high current draw can trip a breaker. The large number of combinations of two load faults results in a large set of identical faults, that is, they all result in the same tripped breaker and subsequent loss of power. This grouping is accomplished through a simple sorting algorithm which groups identical scenario results into bins. Selecting one scenario result from each bin represents the set of unique system states. When applied to the EPS example system the 37,299 total scenarios were sorted and 3,509 unique system states were identified. The significant reduction reflects a large number of identical functional impacts. Many of these identical impacts are related to faults in the sensors which all had five failure modes but resulted in little effect to the system because the controllers that use those sensors were not simulated. The exception to this was failures in the sensors used by the controller, where faults did result in a change in the behavior of the system. The unique system states represent one or more failure scenario results and are the data provided to the clustering methods.

3.3 Clustering of Results Based on Functional Similarity

The motivation for implementing similarity clustering is to identify groupings of failure scenarios and aid designers in creating robust mitigation methods. For example, if a system designer knows of a particular undesirable system state, then finding all scenarios that lead to a similar functional state can identify if adequate control methods have been implemented. In order to identify the relationship between two system states we must develop a metric of distance between function health states. In data clustering methods, the distance between variables can be determined based on the Euclidean distance between the variable values ($Distance = \sqrt{a^2 + b^2}$). However, the values chosen to represent health states are categorical numbers not nominal numbers, which violates an underlying assumption in the Euclidean formulation. Therefore, we introduce a functional distance metric based on functional impact. A relational table (Table 2) is generated to define the similarity between function health states. For this analysis, we identify "Lost" and "No Flow" as having no significant functional difference to the system. Here, designers could choose to increase the distance of off-nominal states to effectively punish and group those scenarios as being worse. Since a low system-knowledge approach is being used for this example, all states have a single unit of difference. For example we can consider a system with two functions and compare the similarity of two fault scenarios. If the resulting system state from scenario 1 is $\{Healthy, Lost\}$ and the system state from scenario 2 is $\{Degraded, NoFlow\}$, then the Euclidean distance between these two using the relation matrix in Table 2 is $\sqrt{1^2 + 0^2}$ or 1.

Table 2: Relational Matrix for Identifying the Distance between Function Health States.

State	Healthy	Degraded	Lost	No Flow
Healthy	0	1	2	2
Degraded	1	0	1	1
Lost	2	1	0	0
No Flow	2	1	0	0

Table 2 is one way to quantify the qualitative distance between functional health states. The k-means clustering algorithm was also applied to the same simulation results using different distance values and where "No Flow" and "Lost" were not equivalent. The cluster centroid and and distances between centroids changes when this scale is changed. However, when comparing the population of scenarios between clusters using different distance matrices, the average error is about .5%. This is within the normal variation of the algorithm when repeated with the same relational matrix. As a result of this finding, it is clear that the concept of functional similarity is strongly dependent on the scale used in this relational matrix. However, population of the clusters and the resulting meaning of those clusters is consistent across scales.

3.3.1 Results of Similarity Clustering

The total distance is calculated by summing over the distance for each function health state. A weighting for functional importance could be incorporated into this step. However, for this analysis each function is given equal importance. This algorithm identifies the functional similarity using Table 2 for each low-level function. Since there is no way to know a priori how many clusters to expect, we repeatedly call the k-means algorithm to cluster the data using 1-10 clusters. Additionally, the algorithm is replicated 100 times for each clustering to avoid local minimums. There are several recognized methods of identifying the appropriate number of clusters. The first approach implemented is the "knee method" (MacKay, 2003), where the within cluster sum of square (WCSS) distance to the cluster centroid is plotted. When additional clusters do not substantially change the WCSS there is no need to further cluster the data. Using the EPS example data, the inflection point appears between 5 and 7 clusters (see Figure 2a). This ambiguity results in the need for a second cluster validation method. By comparison of the dispersion of the scenario similarities within a cluster and the dispersion of the impacts of those scenarios it is possible to identify the appropriateness of the clustering groups. For this work a plot is developed where cluster centroids are plotted against the sum of their function health states normalized by the total number of functions. That is, a vertical value of 1 indicates that all functions are at the healthy state (a nominal scenario). If all component functions in the system had failed in a scenario then the normalized impact would be 4. Vertical position gives an estimate of the scope of the system affected by the fault. Each scenario in a cluster is then plotted based on a horizontal position representing the distance of that scenario to the cluster centroid and a vertical position based on the normalized sum of function health states. Selecting to use five clusters for the k-means algorithm, the plot shown in Figure 2b illustrates the variance of the distances from the cluster centroid in the horizontal direction and the variance of the

Table 3: Evaluating Cluster Distance and Impact Mean and Coefficient of Variation

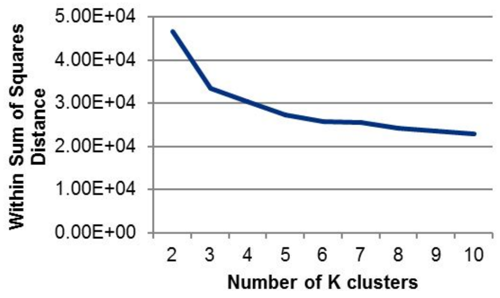
Metric	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Distance Centroid	Mean =11.99 CV= 0.39	Mean =7.54 CV = 0.3	Mean =7.48 CV = 0.90	Mean =8.03 CV =0.26	Mean =5.73 CV = 0.92
Normalized Scenario Impact	Mean = 1.44 CV =0.10	Mean =1.11 CV =0.02	Mean =1.28 CV =0.18	Mean =1.08 CV =0.02	Mean = 1.31 CV =0.04

scenario impacts in the vertical direction.

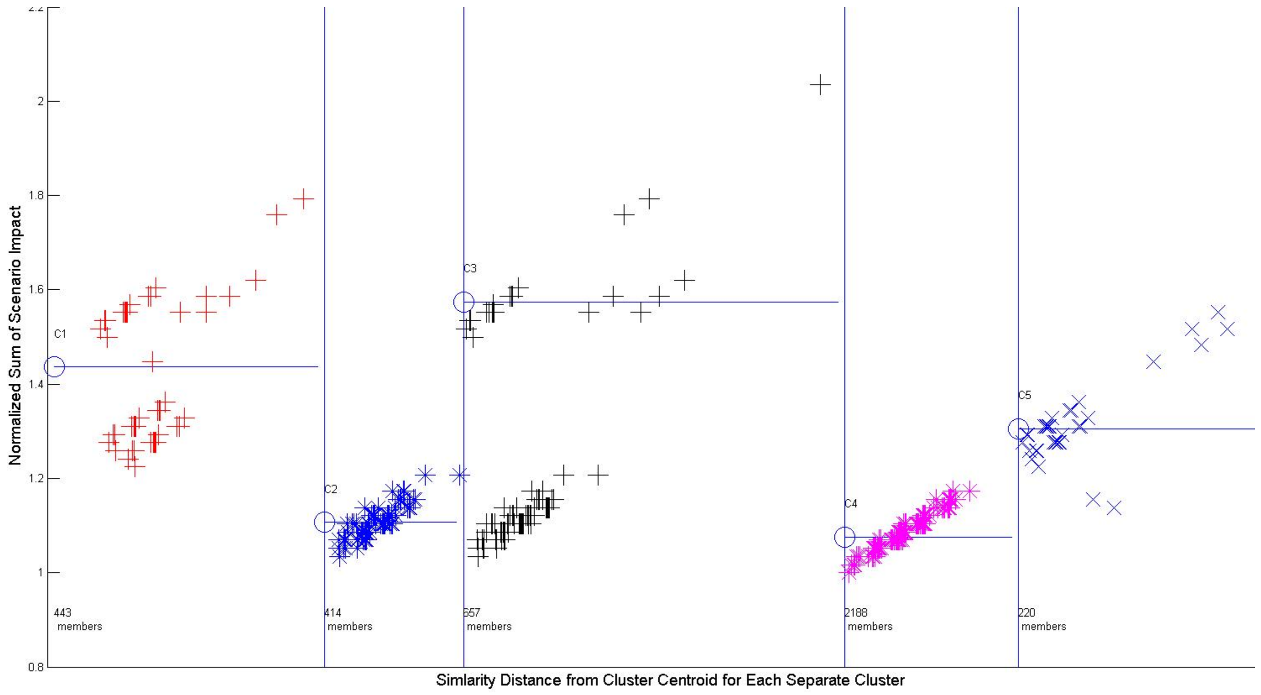
For this example system, Table 3 records the the mean and coefficient of variation for each cluster for the distance from the centroids and the normalized impact of the scenario. The coefficient of variation (CV) is the ratio of the standard deviation and the mean of a population where larger numbers indicate greater dispersion of the data. For the distance metric, the CV indicates how similar the scenarios in the cluster are to each other. For the impact metric, the CV shows the variation in the impact for scenarios in that cluster. Based on this data that the scenarios with the least similarity are in clusters 3 and 5. Similarly, the most diverse set of impacts is in found in clusters 1 and 3. Based on this analysis cluster 3 has the potential to have very dissimilar scenarios with somewhat significant differences in total functional impact. Since there was ambiguity in the the correct number of clusters between 5 and 7 and the potential for cluster 3 to be subdivided, 6 clusters where selected for the analysis of scenario similarity.

3.4 The Latent Class Analysis Method

The second method of grouping the failure results is focused on identifying patterns of failure behavior. For this method a Latent Class Analysis (LCA) is performed on the 3,509 unique fault simulation results using the package poLCA (Linzer and Lewis, 2011b,a) for the statistical software tool R (R Development Core Team, 2011). The poLCA package treats the manifest variables as categorical. The manifest variables in this analysis are the function health states and the latent variable describes the system failure behavior. Similar to the k-means clustering, the number of latent variable classes must be specified prior to the analysis. Therefore, an iterative approach is also taken to fit multiple latent class models



(a) The sum of the within cluster square distance of scenarios to the centroid of their respective cluster.



(b) Graphing clusters based on their distance from centroid and total scenario impact with 5 clusters. Cluster 3 has both vertical (impact) variance and horizontal (similarity) variance and could be separated into two clusters.

Figure 2: Summary of results for applying a modified k-means clustering to the unique system failure states.

with different numbers of classes. In order to avoid local maxima, the poLCA classification algorithm is executed 10 times for each specified number of classes. The correct number of classes is identified as the LC model with the lowest Akaike Information Criterion (AIC) and lowest Baysian Information Cirterion (BIC).

Once the correct latent class model is identified, there are three desired outputs from the

LCA. The first output is a set of conditional probability tables for each manifest variable. These tables identify the probability of finding a manifest variable at a specific state for each category of the latent variable. In the context of this analysis, this indicates that if a failure event is of a particular class of system failure then the function is likely to be in a specific state (healthy, degraded, etc.) The second output uses these probability tables to identify the posterior probability of a scenario belonging to each class of the latent variable. This is the output used for the probabilistic classification of the failure events. Finally, the proportion of each classification is reported. This leads to the identification of the class with the largest membership of failure events.

3.4.1 Results of Model-Based Clustering

The AIC and BIC tend to flatten when evaluating latent models with more classes. Implementing a LCA on the example system data set, minima of AIC and BIC can be seen at 5 classes and 8 classes. Unlike k-means clustering, LCA can identify probabilistic membership of scenarios into each class. Due to the low level of emergent behavior in this system, scenario were classified into each class with very high confidence. The classification of individual scenarios in both 5 or 8 latent classes was compared and 5 classes was selected due to the tendency to split 100% confident classification in the 5 class model into two or more groups with partial classification in the 8 class model.

The meaning of the different classes is not directly found but must be inferred from the resulting groups. That is, if the system is found to have 5 different classes of failure, providing a description of those failure classes cannot be generated from the analysis but requires expert knowledge. The normal approach in an LCA is to compare the probabilities of observing a particular variable (function) state within a class to develop descriptions for that class. However, given 58 function variables that each have 4 different states, this task can be very challenging and is not scalable to large systems. Instead, by comparing the classification provided by LCA to the clustering found through the modified k-means, these groups can be readily identified. This will be discussed in the next section.

3.5 Comparing and Validating Clustering Methods

The modified k-means clustering partitioned all of the unique scenario result states into 6 clusters. Each scenario result then has two properties: 1) The normalized total impact of that scenario; and 2)The distance of that scenario from the theoretical centroid of the cluster in which it belongs. This distance is a measure of functional similarity over the identified 58 functions in the space. Scenarios very near the centroid are the “typical” scenarios for that cluster.

The result of the LCA model is a predictive description of the latent failure behavior and the probabilities of observing a particular function’s state. Comparing this model-based approach to the k-means approach has two benefits. First, LCA provides a mathematical validation of the partitioning of the k-means method when the two clustering methods agree. Second, the centroid of the k-means cluster can be used to identify the meaning of the matching LCA cluster.

In Figure 3, the k-means clusters are plotted based on total normalized impact and their distance from the cluster centroid. The classification of scenarios by the LCA and the modified k-means was inconsistent for 26 of the 3509 unique scenarios. The scenarios that where classified differently by the two methods are noted with diamonds in Figure 3. Because this plot compares similarity and normalized impact, some of the markers overlap. This means that these scenarios are equally different from the cluster centroid and have affect the same number of functions. It does not mean that the final system state of these scenarios is identical.

There are two metrics for evaluating the consistency of the clusters found by the two algorithms. In Table 4 both metrics are shown for the 5 class LCA results and the 6 clusters from the k-means algorithm. First, to compare if the scenario populations are consistent, the union of cluster membership is evaluated. In Table 4, the number below each cluster

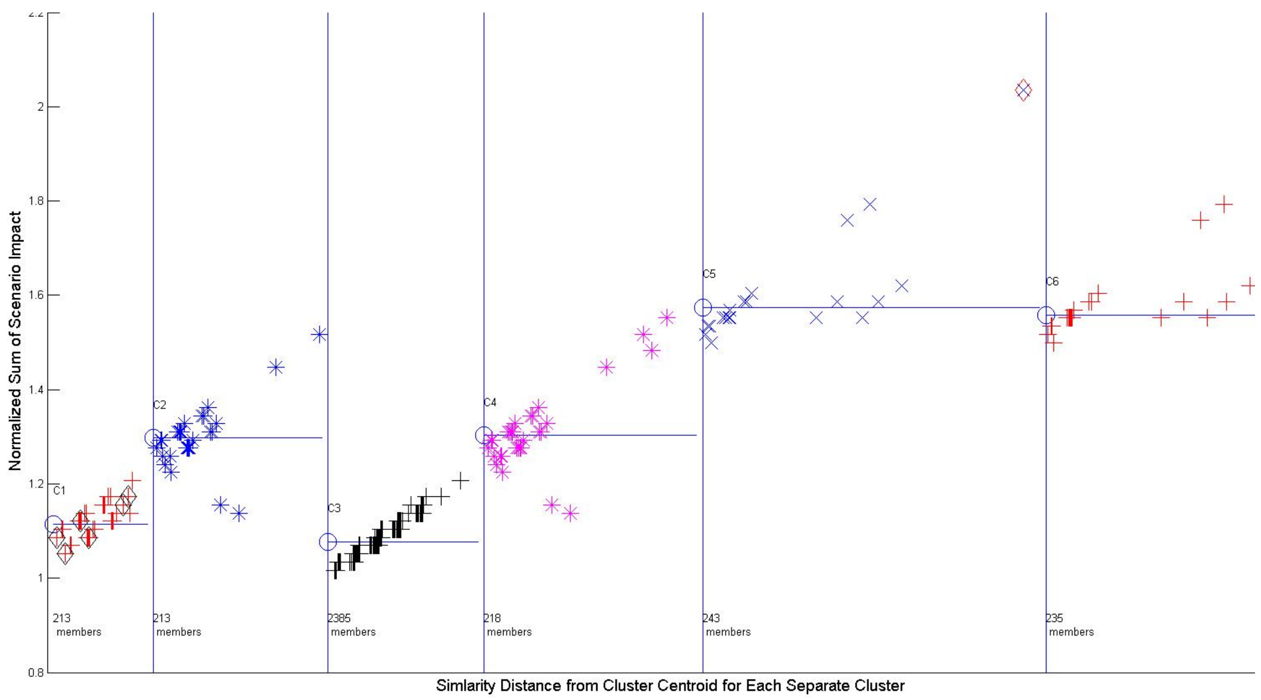


Figure 3: Comparing the clustering found through the k-means and LCA method. Discrepancies are marked with diamonds. Note that some markers overlap.

name is the total number of scenarios classified into that cluster or class. The integers within the table show the membership union. For example, 2 of the scenarios found in the third LCA class are also found in second cluster from the k-means algorithm. The numerical order provided by the algorithm is random. The second metric for comparing clusters is the distance between centroids. Since the LCA gives a probability distribution of health states for each function as the centroid, it cannot be directly compared to the single value centroids from the k-means algorithm. Instead, the centroid of the resulting classification from the LCA is used. That is, if a class contained scenarios 1-3, then the centroid is based on the centroid of those three scenario results and not the probabilistic centroid of the model of that class which the LCA algorithm used to fit scenarios 1-3. In Table 4, the centroid to centroid distance is reported for each cluster and class as a real number in units of the distance between functional states. From Table 4, both metrics identify the same overlap in the k-means clusters and LCA classes (as indicated in the colored cells). Using this example it is clear that the fourth LCA class is the combination of first and third k-means cluster.

Table 4: Comparing the Centroid to Centroid Cluster Distance and Scenario Membership Overlap

Clusters	**LCA Classes**				
# of scenarios	LCA 1 243	LCA2 209	LCA3 245	LCA4 2605	LCA5 206
K1 213	10.19 0	7.95 0	11.41 0	2.33 213	8.25 0
K2 213	12.07 0	9.27 2	815 2	5.17 3	2.60 206
K3 2385	10.99 0	7.79 0	11.05 0	0.30 2385	7.75 0
K4 218	7.94 8	2.60 207	12.02 0	5.17 3	9.35 0
K5 243	3.51 235	5.77 0	12.73 8	7.40 0	10.64 0
K6 235	12.99 0	10.62 0	3.67 235	7.39 0	5.76 0

3.6 Relating Clusters to System-Level Functionality

The centroid of each cluster found through the modified k-means analysis represents a point in the functional state space defined by 58 functions. In this space each function may have the value between 1-3 representing *Nominal*, *Degraded*, and *Lost or Now Flow* respectively. By observing what scenario is closest to the cluster centroid and what functional dimensions have the largest impact for a cluster, the meanings of the clusters become apparent. In Table 5 the k-means clusters are sorted so that the highest functional impacts are grouped together. All component functions that do not appear in Table 5 have values near 1 and are considered predominately *Nominal* for the scenarios in that cluster. Additionally, the representative scenario for that cluster is also listed in the second row. The non-nominal functions are listed for each cluster along with the centroid's location along that functional axis. By looking at these characteristic functions and the health states for each cluster centroid, the clusters can be described in terms of their dominant system level effects. Thus each cluster is defined by a set of functions in some off-nominal health state. While the clustering algorithm identifies that there are dependencies between these functions (and thus clusters them together), it can not directly reveal causality. For this reason we take the component-level functions identified in each cluster and use the model to organize the connectivity of the graph shown in Figure 4. Care should be taken not to interpret this as the direction of fault propagation. Instead Figure 4 shows the relationship between the functional dependencies in the clusters and the physical system architecture. Finally, as can be seen in Table 5, the K3 cluster centroid does not have any characteristic functions in the degraded or lost state. This means that scenarios within this group have few failures that affect multiple functions and there is minimal dependencies between the faulty states of functions. Since the degraded and lost state functions are used to characterize the clusters, the K3 cluster is not included in Figure 4.

Table 5: Off-Nominal Functional Impact for Each Cluster and Representative Scenario

K1		**K2**		**K3**		**K4**		**K5**		**K6**	
Breaker 5 Open		Breaker 6 Open		Fan 2 Failed Off		Breaker 3 Open		Breaker 1 Open		Battery 2 Disconnected	
DC 1	3	Fan Relay 2	2.99	Fan 2	1.28	Fan Relay 1	2.99	Breaker 3	3	Breaker 6	3
DC 1 Relay	2.33	Pump Relay 2	2.99	Fan 1	1.19	Pump Relay 1	2.99	Inverter 1	3	Inverter 2	3
Fan 1	2.06	Pump 2	2.99	DC 2	1.19	Pump 1	2.99	Breaker 4	3	Breaker 7	3
Fan 2	1.14	Light 2 Relay	2.99	Light 2	1.14	Light 1	2.99	Fan Relay 1	3	Fan Relay 2	3
		Light 2	2.99	Light 1	1.14	Light 1	2.99	Pump Relay 1	3	Pump Relay 2	3
		Inverter 2	2.97			Inverter 1	2.97	Pump 1	3	Pump 2	3
		Fan 2	2.97			Fan 1	2.97	Light Relay 1	3	Light Relay 2	3
		Breaker 7	2.34			Breaker 4	2.35	Light 1	3	Light 2	3
								Breaker 5	3	Breaker 8	3
								DC Relay 1	3	DC Relay 2	3
								DC1	3	DC2	3
								Fan 1	2.99	Fan 2	2.99
								Relay 2	2.93	Relay 4	2.93
								Battery 1	2.91	Battery 2	2.91
								Breaker 1	2.91	Breaker 2	2.91

4 Results

In this section we will present how the results of conducting the clustering approach address the three objectives of: 1) characterizing the impacts of a large number of failure scenarios; 2) Identifying the system-level meaning of those characterizations; and 3) determining how this analysis can be used to make system design decisions. The first objective of characterization is accomplished through identifying an underlying pattern of failure behavior exhibited in the system states that result from numerous fault simulations. This underlying pattern of behavior is found through applying the Latent Class Analysis (LCA) to the set of unique systems states. The result of applying the LCA to the 3,509 unique systems states that

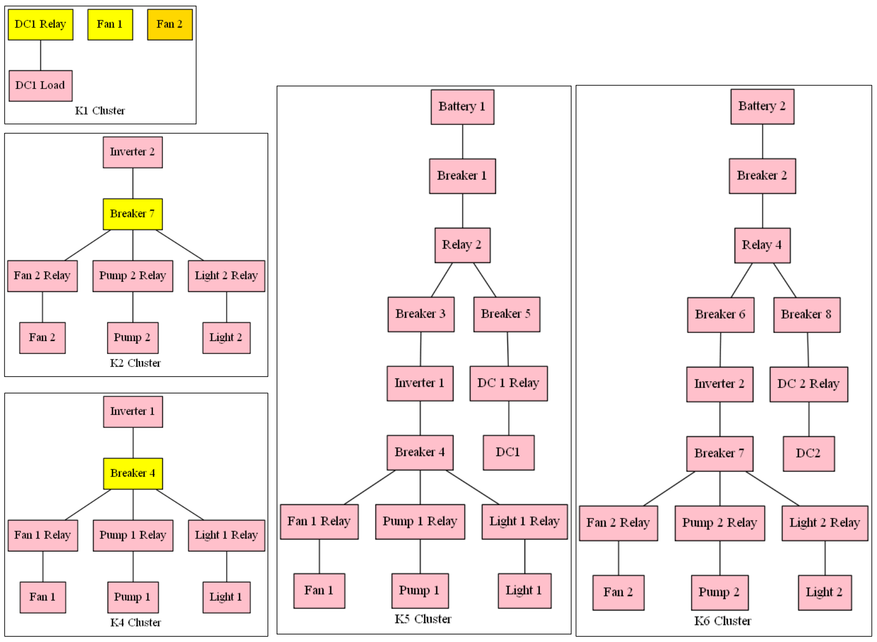


Figure 4: The clusters identified through the modified k-means and LCA are mapped to the system model.

result from fault scenario simulation for the example system best fit a model with 5 discrete classes of system failure. Further, the probability of scenarios fitting exactly one of the five classes is very high (most are 100%). This confirms that five different patterns of system failure emerge from the simulation of combinations of component fault behavior. Because the LCA approach fits a structure to the data, each class is fully defined by the probability of a function being at a health state. The health state of a function as a result of simulating a scenario is deterministic and has a known value after simulation. However, the class of system failure is a model where each function has a probability of being at each health state. The system-level failure behavior classes are the result of the interactions of component behaviors. For this reason the five classes represent emergent failure behavior observed at the system level in the scenarios simulated. This does not represent all potential emergent behaviors of the system. The clustering algorithm uses the simulation dataand thus if the behavior is not present in the simulation it will not be identified by the algorithm. However, due to the large number of scenarios that form the data for each class model, this approach does provide some confidence that this system will not experience significantly different behavior. While the LCA-based clustering was able to address the first objective by finding underlying classes of system behavior to characterize scenarios, those classes must also be related to the system level functions of interest.

The second objective, to identify the system-level meaning (for designers) of the classes of behavior, is accomplished using a k-means clustering on scenario impact similarity. By using the cluster centroids, each cluster is described with a set of functions and their health state. Limiting the focus to degraded and lost functionality provided five of six clusters that can be used to relate the system functionality to the scenario clusters. Figure 4 shows the characteristic functions and their health states for each cluster and uses the system model to identify physical connections. The third cluster centroid did not exhibit consistently degraded or lost functionality and is not included. By comparing the system model to the cluster's representative functions, the relation to system-level functions begins to emerge. For example, the scenarios classified in Cluster 4 are predominately scenarios affecting the first load bank. When certain fault scenarios result in loss of power to that load bank the function of those components is lost or degraded. For this simple system this demonstrates that, without a priori knowledge of component connectivity, the clustering approaches identified behavior-based connections. For more complex systems with emergent behavior, these connections could be identified in components in different subsystems where interactions may be harder for designers to predict.

The third objective of this work was to determine whether the discrete failure behavior of the system identified through the clustering analysis could be used for system-level design decision making. As described in Section 2.4, the example system is designed to be fault tolerant where the software control attempts to operate as many of the loads as possible. The software control was designed to recognize and operate the system at the best available of the 7 potential states identified in Table 1. Comparing these 7 control action states to the clusters provides an assessment of the effectiveness of the system architecture and control. Table 6 shows how the degraded control states address faults from certain clusters. One example of a design decision that could be made after application of this analysis is to redesign the architecture and control to address the individual load faults that are seen in Cluster 3. The application of this approach has shown that the current control method addresses four of the fundamental failure behaviors of the system, but has no specific action states to address the other two.

Table 6: Relation of Degraded Software Control States to Scenario Clusters

State 3 $Batt1 \rightarrow Load1$	State 4 $Batt2 \rightarrow Load2$	State 5 $Batt1 \rightarrow Load2$	State 6 $Batt2 \rightarrow Load1$
Cluster 6 Cluster 2	Cluster 5 Cluster 4	Cluster 4	Cluster 2

Finally, the small set of scenarios that k-means classified in Cluster 5 and that the LCA grouped in cluster 6 (see Figure 3), correspond to scenarios where both battery banks could provide no power. These special scenarios that are hard to cluster indicate important scenarios for the system designer to investigate. For this system, scenarios where both batteries are disconnected (and other similar scenarios) are unrecoverable by the software control. Based on the probability, and the consequence of those faults, designers may want to redesign the system redundancies.

5 Conclusions

This paper proposed two different approaches for clustering the results of a function-based failure analysis method in the early design stage. In contrast to others methods which focus on single faults or single failure scenarios, the goal of this work is to characterize a design's overall failure behavior. The results of implementing these clustering approaches on an example fault tolerant, software-controlled electrical power system (EPS) demonstrates the ability to both identify system-level failure behavior and use the classification of that behavior for design decision making.

The first clustering approach was a modified k-means algorithm where the distance between failure scenarios was determined based on the functional similarity of the impact of those scenarios. This method partitions the fault scenarios into discrete clusters. Each cluster has a centroid which is the representative set of functions and their health states for that cluster. The second clustering approach was a model-based method that used Latent Class Analysis (LCA) to identify a latent variable with a set of discrete classes. The latent variable is a single unmeasurable variable that describes the system's failure state or failure modes. The LCA provides a probabilistic model that is used to characterize the system behavior. By comparing these methods the k-means clustering was mathematically validated when the scenario groupings agreed with the LCA classifications. Further, the challenge of describing the system failure modes found through LCA is addressed by using the centroids of the corresponding k-means clusters.

The example EPS describes how the designed control addressed some but not all of the system failure behavior modes. When informed by other variables such as cost, this could be used in a multi-objective decision making process. A future challenge that this work can address is that large-scale system modeling may be impossible at the component fidelity level. However, the LCA classes are models of the system state and could be used as abstractions for the component details. For example, the EPS can be described as having a few nominal modes and the identified five failure modes. This simplified model can then be incorporated into a larger model without the need to specify low-level component behavior.

Additionally, more work is needed in applying the presented methodology to complex systems to develop a relationship between the completeness of the analysis and the number and types of failures to simulate.

The objective of this work is to aid designers in identifying the potential system-level failure behaviors and use the classification of those behaviors to improve system design. By using data analysis techniques on large sets of design-stage analysis data, designers can make better risk-informed decisions and provide stake-holders with safer systems.

Acknowledgment

This research was supported by NASA through the Systems Engineering Consortium sub contracted through the University of Alabama-Huntsville (SUB2013059). Any opinions or findings of this work are the responsibility of the authors, and do not necessarily reflect the views of the sponsors or collaborators.

References

Coatanéa, E., Nonsiri, S., Ritola, T., Tumer, I., and Jensen, D. (2011). A framework for building dimensionless behavioral models to aid in function-based failure propagation analysis. *Journal of Mechanical Design*, 133:121001.

Dulac, N. and Leveson, N. (2005). Incorporating safety into early system architecture trade studies. In *Proceedings of the International conference of the system safety society*.

Estivill-Castro, V. (2002). Why so many clustering algorithms: a position paper. *ACM SIGKDD Explorations Newsletter*, 4(1):65–75.

Grantham-Lough, K., Stone, R. B., and Tumer, I. Y. (2009). The risk in early design method. *Journal of Engineering Design*, 20(2):144–173.

Han, J., Kamber, M., and Pei, J. (2006). *Data mining: concepts and techniques*. Morgan kaufmann.

Huang, Z. and Jin, Y. (2008). Conceptual Stress and Conceptual Strength for Functional Design-for-Reliability. In *Proceedings of the ASME Design Engineering Technical Conferences; International Design Theory and Methodology Conference*.

Jensen, D., Hoyle, C., and Tumer, I. (2012). Clustering function-based failure analysis results to evaluate and reduce system-level risks. In *ASME 2012 International Design Engineering Technical Conference and Computers and Information in Engineering Conference*.

Jensen, D., Tumer, I. Y., and Kurtoglu, T. (2008). Modeling the propagation of failures in software-driven hardware systems to enable risk-informed design. In *Proceedings of the ASME International Mechanical Engineering Congress and Exposition*.

Jensen, D., Tumer, I. Y., and Kurtoglu, T. (2009a). Design of an Electrical Power System using a Functional Failure and Flow State Logic Reasoning Methodology. In *Proceedings of the Prognostics and Health Management Society Conference*.

Jensen, D., Tumer, I. Y., and Kurtoglu, T. (2009b). Flow State Logic (FSL) for analysis of failure propagation in early design. In *Proceedings of the ASME Design Engineering Technical Conferences; International Design Theory and Methodology Conference*.

Krus, D. and Grantham Lough, K. (2007). Applying function-based failure propagation in conceptual design. In *Proceedings of the ASME Design Engineering Technical Conferences; International Design Theory and Methodology Conference*.

Kurtoglu, T., Johnson, S., Barszcz, E., Johnson, J., and Robinson, P. (2008). Integrating system health management into early design of aerospace systems using functional fault analysis. In *Proc. of the International Conference on Prognostics and Heath Management, PHM'08*.

Kurtoglu, T. and Tumer, I. Y. (2008). A graph-based fault identification and propagation framework for functional design of complex systems. *Journal of Mechanical Design*, 130(5).

Kurtoglu, T., Tumer, I. Y., and Jensen, D. (2010). A Functional Failure Reasoning Methodology for Evaluation of Conceptual System Architectures. *Research in Engineering Design*, 21(4):209.

Lazarsfeld, P. F. and Koch, S. (1959). *"Latent Structure Analysis" in Psychology: A Study of a Science*, volume 3. New York: McGraw-Hill.

Leveson, N. (2011). Engineering a safer world. *MIT Press*.

Linzer, D. A. and Lewis, J. (2011a). polca: an r package for polytomous variable latent class analysis. *Journal of Statistical Software*, 42(10):1–29.

Linzer, D. A. and Lewis, J., editors (2011b). *poLCA: Polytomous Variable Latent Class Analysis*. R package version 1.3.1. http://userwww.service.emory.edu/ dlinzer/poLCA.

Lloyd, S. (1982). Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Mardia, K. V., Kent, J. T., and Bibby, J. M. (1980). *Multivariate analysis*. Academic press.

Papakonstantinou, N., Jensen, D., Sierla, S., and Tumer, I. Y. (2011). Capturing interactions and emergent failure behavior in complex engineered systems and multiple scales. In *Proceedings of the ASME Design Engineering Technical Conferences; Computers in Engineering Conference*.

Pearl, J. (2000). *Causality: models, reasoning and inference*. Cambridge Univ Press.

Poll, S. (2007). Advanced diagnostics and prognostics testbed. In *18th International Workshop on Principles of Diagnosis*.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Sierla, S., Tumer, I., Papakonstantinou, N., Koskinen, K., and Jensen, D. (2012). Early integration of safety to the mechatronic system design process by the functional failure identification and propagation framework. *Mechatronics*, page doi:10.1016/j.mechatronics.2012.01.003.

Stone, R. B., Tumer, I. Y., and VanWie, M. (2005). The Function Failure Design Method. *Journal of Mechanical Design*, 14:25–33.

Tumer, I. and Smidts, C. (2010). Integrated design and analysis of software-driven hardware systems. *IEEE Transactions on Computers*, 60:1072–1084.

Vermunt, J. and Magidson, J. (2004). *"Latent Class Analysis" in The Sage encyclopedia of social science research methods*, volume 1. Sage Publications, Inc.