Automated Generation of Fault Scenarios to Assess Potential Human Errors and Functional Failures in Early Design Stages

Lukman Irshad
School of Mechanical,
Industrial and Manufacturing Engineering,
Oregon State University,
Corvallis, OR 97331
e-mail: mohammoh@oregonstate.edu

H. Onan Demirel[1]
Assistant Professor
School of Mechanical,
Industrial and Manufacturing Engineering,
Oregon State University,
Corvallis, OR 97331
e-mail: onan.demirel@oregonstate.edu

Irem Y. Tumer
Professor
School of Mechanical,
Industrial and Manufacturing Engineering,
Oregon State University,
Corvallis, OR 97331
e-mail: irem.tumer@oregonstate.edu

Human errors are attributed to a majority of accidents and malfunctions in complex engineered systems. The human error and functional failure reasoning (HEFFR) framework was developed to assess potential functional failures, human errors, and their propagation paths during early design stages so that more reliable systems with improved performance and safety can be designed. In order to perform a comprehensive analysis using this framework, a wide array of potential failure scenarios need to be tested. Coming up with such use cases that can cover a majority of faults can be challenging for engineers. This research aims overcome this limitation by creating a use case generation technique that covers both component- and human-related fault scenarios. The proposed technique is a time-based simulation that employs a modified depth first search (DFS) to simulate events as the event propagation is analyzed using HEFFR at each time-step. The results show that the proposed approach is capable of generating a wide variety of fault scenarios involving humans and components. Out of the 15.4 million scenarios that were found to violate the critical function, two had purely human-induced faults, 163,204 had purely non-human-induced faults, and the rest had a combination of both. The results also show that the framework was able to uncover hard-to-detect scenarios such as scenarios with human errors that do not propagate to affect the system. In fact, 86% of all human action combinations with nominal human-induced component behaviors had underlying human errors.
[DOI: 10.1115/1.4047557]

Keywords: computer-aided design, human computer interfaces/interactions, model-based systems engineering, complex engineering system design, conceptual design, human reliability assessment, failure assessment, risk assessment

1 Introduction

Human error has been attributed as a major cause of accidents and performance losses in complex engineered systems [1,2]. On average, 60–80% of all aviation accidents are caused by human error [3]. The National Safety Council reports that 90% of mobile crane crashes are results of operator error [4]. The U.S. Navy estimates that skill-based errors cost around $3.3 billion over a period of 4 years [3]. In addition, compensations due to musculoskeletal disorders associated with poor working conditions amount to $45–55 billion annually [3]. Similarly, the Institute of Medicine estimates that between 44,000 and 98,000 Americans die each year due to medical errors that can be prevented or mitigated, which costs around $17 billion [1]. Furthermore, human error is directly linked to the partial nuclear meltdown at Three Mile Island, reactor explosion at Chernobyl, and gas leak in Bhopal, India, which all took a heavy toll on both local and global economies, communities, and ecologies [5]. The above body of evidence shows that not only human errors are prevalent but also they are costly and at times fatal. If one looks further into these failures, the cause is often a combination of both technical and human elements acting upon each other over time rather than a single failure type [6,7]. Hence, it is important to analyze both component failures and human errors in combination from a risk mitigation standpoint rather than analyzing one or the other in isolation. Design changes made later in the design process after design decisions and resource allocations are committed are costly and time-consuming [8] leading to budget overruns and missed deadlines. Hence, it is not only important to analyze both component failures and human errors but also do it during early design stages.

Engineers have relied on component failure assessment techniques and human reliability assessment techniques to assess the risk of component failures and human errors, respectively. Techniques such as Failure Modes and Effects Analysis [9], Fault Tree Analysis [10], and Event Tree Analysis [11] have been traditionally used to assess component failures. They have also been used to assess human errors by switching the context to humans. However, this is usually performed by separate experts, meaning the combined effects of human errors and component failures cannot be assessed. They also require detailed component models, making them applicable only during later design stages. In a bid to move failure assessment to early design stages, methods such as Functional Failure Identification and Propagation [12], Conceptual Stress, and Conceptual Strength Interference Theory [13] were developed. These methods do not address human errors with enough detail. Human reliability assessment methods such as Systematic Human Error Reduction and Prediction Approach [14] and Technique for Human Error Rate Prediction [15], on the other hand, only assess human errors and are only applicable during later design stages.

As a result, recent research has yielded human error and component failure assessment techniques such as a joint analysis using Function Failure Design Method (FFDM) and Systematic Human Error Reduction and Prediction Approach (SHERPA) [16], Function Human Error Design Method (FHEDM) [17], and Human Error and Functional Failure Reasoning (HEFFR) [18] that are capable of analyzing both component failures and human errors in combination early in design, not just to mitigate potential failures but also to minimize cost and time to market. HEFFR differs from

[1]Corresponding author.

Manuscript received September 27, 2019; final manuscript received June 11, 2020; published online July 9, 2020. Assoc. Editor: Caterina Rizzi.

the aforementioned methods because it is capable of analyzing the propagation paths of the faults, giving additional insight into how the failures and human errors propagate to affect the overall system. It also can be coupled with other computational design tools such as Digital Human Modeling (DHM) to perform ergonomic evaluations early in design [19].

HEFFR uses critical event scenario inputs to produce potential functional failures, human errors, and their propagation paths as outputs. To perform a comprehensive analysis, HEFFR requires input scenarios that cover a majority of if not all component failures and human errors. A major shortcoming of HEFFR is that it relies on the designer(s) to come up with such use cases; thus making it highly subjective. Also, it is highly unlikely that anyone or a group can capture use cases exhaustively enough to cover a majority of the fault scenarios. The overall goal of this research is to overcome these limitations of HEFFR by coming up with a method to automatically generate use cases that can potentially cover a wide range of fault conditions involving both component failures and human errors. Automatically generating use cases will lessen subjectivity and allow designers to focus their time and efforts toward other important tasks rather than spending human resources generating use cases.

Functional Failure Identification and Propagation (FFIP) [12], a precursor of HEFFR, suffered from the same limitations as HEFFR when it comes to scenario generation. To address these limitations, previous research has introduced an automated scenario generation method that generates event trees for failure events for which the systems' safety functions fail to activate [20]. One shortcoming of this technique is that event trees related to only one triggering event can be studied at a time. Inherent Behavior of Functional Models (IBFM) [21] is another failure assessment technique that automatically generates fault scenarios by first simulating one fault at a time and eventually progressing to exhaustively simulate faults by incrementing the number of faults introduced at a time by one. IBFM uses functional behaviors instead of component behaviors to simulate faults which makes the fault generation method not applicable to HEFFR since HEFFR relies on component behavior models to simulate faults. There are several other automated event tree generation methods that either do not directly pertain to system design [22,23] or are only applicable at later design stages [24–26]. Additionally, none of the methods discussed generate use cases that include potential fault conditions relating to human errors. Hence, there is a need to explore other avenues that use model-based scenario generation to overcome the limitations of HEFFR.

In the software engineering field, model-based testing is a systematic method used early in the software development lifecycle to test software [27]. This includes automatically generating and evaluating test cases [27]. Unified Modeling Language (UML) is widely used in industry and academy to represent software systems in a graphical manner [28]. It compromises a collection of nine different types of graphs (e.g., activity and state) that can represent both structural and dynamic elements of a software system [28]. UML-based test case generation is one of the most commonly used test case generation methods in software engineering because of its universal nature and convenience of use [29]. It is also known to be highly effective at system-level software testing [29]. Since its graphical nature, similarities with the use of multiple graphs to represent the system in HEFFR, ease of implementation, and applicability during early design stages, we chose the UML-based automated test case generation methods as a basis to this research.

This paper uses the UML-based automated scenario generation techniques as a basis to introduce a method that can automatically generate use cases as inputs for HEFFR. It uses a modified Depth First Search (DFS) algorithm, component behavior models, and the human action classifications to achieve its intended purpose. A HEFFR analysis is conducted as the algorithm passes through each level in the search tree to identify the system status and depending on the functions affected (specified by the user), the algorithm passes on to the next level or moves on to a new branch. Overall, this research allows designers to conduct a risk assessment of component failures and human errors early in design using HEFFR. Rather than having to retrofit changes or find workarounds to mitigate potential risk, this approach will allow designers to come up with complex engineered systems with minimal potential for failures. Also, this paper demonstrates the capabilities and limitations of the proposed algorithm using a liquid tank design example. Note that this paper is based on a paper presented at the 2020 ASME Computers and Information Systems in Engineering Conference [30].

2 Background

In this section, we go into detail about previous research that inspired this novel work. First, we discuss the Human Error and Functional Failure Reasoning (HEFFR) framework. Next, we explore existing failure assessment techniques that include automated fault generation and assess the potential of those methods to be utilized in automatic use case generation for HEFFR analysis. Finally, we look into automated test case generation methods used in software engineering, specifically UML-based test case generation methods, and explain the reasons behind choosing the approach to develop the use case generation method introduced in this research.

2.1 Human Error and Functional Failure Reasoning.

Human Error and Functional Failure Reasoning (HEFFR) [18] is an extension of Functional Failure Identification and Propagation (FFIP) [12] framework. It uses three graphical representations to build a system model and represent human interactions: a Functional Model, Configuration Flow Graph (CFG), and Action Sequence Graphs (ASG). A Functional Model systematically decomposes the intended function of the system into subfunctions and flows where nodes represent functions and arcs represent the flow of material, energy, or signal. CFGs represent the generic components that are needed to fulfill each function in the functional model as nodes and the flow of material, energy, or signal as arcs. An ASG is all actions a user needs to perform to interact with a specific component in CFG arranged in a sequence where nodes represent actions and arcs represent action outcomes and control signals. An ASG needs to be created for all components that interact with humans.

HEFFR uses component behavior models and action classifications to simulate failure and human errors. Component behavior models define all nominal and faulty states for each component in the CFG. Action classifications define all nominal and faulty action states for each action in the ASG. The critical event scenario (potential fault conditions) inputs are reasoned using the behavior simulation to identify the non-human-induced behavior state of each component in the CFG. The action simulation algorithm reasons the same input scenarios to determine the action classification of each action in ASG and then traces the action classifications using the ASG to identify the human-induced behavior states of components that interact with humans. These human-induced behavior states are then fed into the behavior simulation and subsequently into functional failure logic where the function component relationships and the functional model are used to classify the status of each function as "Lost," "Degraded," or "Nominal." Finally, functional failures (if the functional state is "Lost" or "Degraded"), human errors, and their propagation paths are produced as outputs. The framework can model both cognitive (e.g., failed detection) and non-cognitive (e.g., cannot reach an object, where an attempt to reach is made but cannot be completed due to physical limitations such as obstruction) human errors and how they affect the functional health of the system. The HEFFR simulation is time-based where each time-step is discrete events. The overall architecture of HEFFR is depicted in Fig. 1. More details on how to set up the modules of HEFFR framework can be found in Refs. [12,18,31].

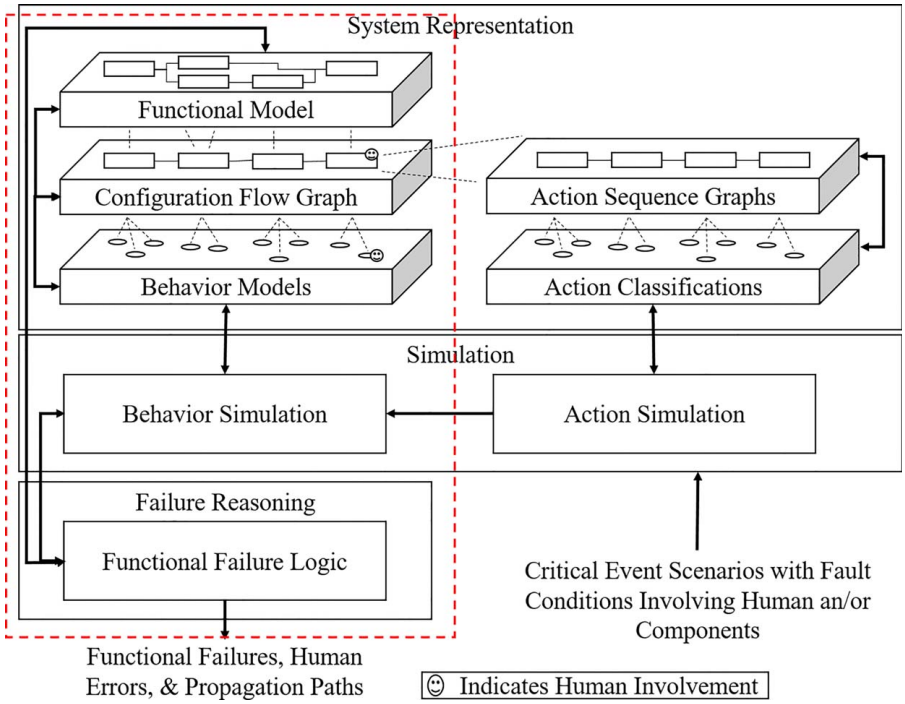

Fig. 1 The architecture of human error and functional failure reasoning (HEFFR) method, where the modules from FFIP are highlighted by the red box

HEFFR, when applied solely, is not capable of identifying any harm the human error can cause to the users or any human performance issues. However, when coupled with digital human modeling, it enables the evaluation of human performance and safety early in the design [19]. Another major limitation of HEFFR is that it relies on the designers to come up with potential fault conditions for inputs. It is important to have a wide range of fault scenarios that can cover most, if not all, potential fault conditions in order to fully understand the potential risk of failures and human errors. It is highly unlikely that any one person or a team can come up with such a range of use cases. In addition, the simulation only analyses one event scenario at a time. Hence, comparing and contrasting the impact of different input scenarios can be cumbersome. This paper addresses these issues by automating the scenario generation and enabling an exhaustive analysis of all generated scenarios to give designers more control over analyzing the results.

2.2 Automated Scenario Generation for Complex Engineered Systems Design and Failure Assessment. Previous research has attempted to automate scenario generation to conduct failure assessment and to validate system designs. One such attempt automatically generates test cases to validate system design and implementation against requirements using a four-part algorithm [32]. First, the algorithm uses the requirements model and the Greedy Search algorithm to identify base scenarios that have the potential to test all requirements. Next, incomplete base scenarios are identified and enhanced to make them complete. Finally, the base scenarios and the enhanced base scenarios are combined to create a comprehensive list of test cases. TestWeaver [33] is another automated test case generation tool used for systematic testing. It works like a game of chess where TestWeaver is playing against the system under test by making a series of moves with the goal of attaining goal states which force the system to violate requirements. This allows the testing of a wide range of alternative paths that can contribute to requirement violations. This tool was successfully used in the design of the crosswind stabilization function to the Active Body Control (ABC) suspension for the Mercedez-Benz 2009 S-Class [34]. Both of the test case generation methods discussed above do not explicitly search for failures instead they are intended toward identifying requirement violations.

There have also been attempts to automate failure assessment by automating the fault cause and effect scenarios. Prior research has implemented automated failure cause generation in Failure Modes and Effects Analysis (FMEA) for diagnostics and prognosis analysis. One such method induces various component failures to the system and compares results between nominal system behavior and faulty system behavior to understand the effect of a failed component [35]. The algorithm performs this analysis by exhaustively covering all possible component failures. Another diagnostic application of FMEA automatically generates diagnostics and fault analysis [36]. Another study looks into improving reliability by automatically generating fault trees. It uses a Finite State Machine (FSM)-based system model to generate a fault tree that consists of all possible failures [37]. Another method aimed at mitigating risk, SimpraPlan [38], uses functional requirements and the physical structure of the system to generate scenarios that test for system vulnerabilities. There are several other methods that automate event tree generation. However, these event tree generation methods either do not directly relate to system design [22,23] or require historical data or detailed system data [24–26] making them inapplicable during early design stages. All of the methods presented above either do not apply in a design context, do not generate human-related fault conditions, or require detailed system data. Hence, they are not effective when it comes to generating use cases for HEFFR analysis.

IBFM [21] uses functional models and functional behavior models to automatically generate fault scenarios to assess potential failures and their system-level effects. The simulation starts with introducing one fault at a time and progresses by incrementing the number of faults introduced by one. It also allows for pseudo-time-based simulations. Even though IBFM can be applied early in design, since it does not consider component behavior models, the scenario generation technique cannot be applied to HEFFR. Another early design stage failure assessment framework,

Functional Failure Identification and Propagation (FFIP), has an extension in which automated scenario generation is present [20]. It generates event trees for triggering events that fail to activate a set of predefined safety functions. This method can only evaluate one triggering event and the corresponding event tree at a time; thus, making the overall analysis human resource intensive when a large number of triggering events need to be analyzed. Both of these methods are not capable of generating human-machine interaction related use cases with enough detail to be applied in an HEFFR analysis. As none of the techniques detailed above provide a sufficient solution to generate use cases for HEFFR automatically, we were prompted to broaden our research scope to other fields that utilized model-based system representations. A comparison between the proposed work and the existing risk assessment methods with automated fault scenario generation is provided in Table 1.

2.3 Automated Scenario Generation in Software Engineering. We have explored the Model-Based Testing (MBT) methods used in software engineering because they involve automated test case generations. The advantages of MBT are listed as follows [39]:

- It can reduce design cost.
- It provides the ability to identify issues with requirements.
- It allows testing early in the software design lifecycle.
- It allows for comprehensive tests that exhaustively cover all potential use cases.
- Fault Detection is more effective and efficient when compared with other types of software testing.

The advantages of MBT are characteristics that would be ideal in the use case generation method of HEFFR. Hence, we explored the test case generation methods utilized in MBT further. One such test case generation method uses environmental behavior for scenario generation [40]. It defines the behaviors of the system using event traces that are made of relations between precedence and inclusion. Event grammars, which specify the possible event traces, are traversed top-down and left-right to generate test cases and evaluate cyber-physical systems. Another test case generation framework uses high-level Petri Nets [41] to generate functional models, access control models, and potential threat models. Petri Nets is a systematic method to model and verify software systems [41]. The Petri Net models are then searched using Depth First Search (DFS) and Breadth First Search (BFS) to generate test cases automatically. A type of Petri Net model (namely Colored Petri Net model) that can be used to model distributed systems is used in another approach to generate test cases for distributed system protocols [42]. This approach takes a simulation-based approach to automatically generating test cases. In an attempt to overcome challenges relating to testing and verifying dynamic Simulink models, Matinnejad et al. proposed a meta-heuristic search-based test case generation method that covers both continuous and discrete behaviors [43]. The test case generation aims to increase the diversity in the output signals so that the chances of finding unexpected output signals are maximized. Finally, the generated test cases are prioritized based on their likelihood of identifying faults.

Unified Modified Language (UML)-based automated test generation methods are commonly used MBT types [29]. UML is a system modeling language that includes several diagrams to represent the architectural and behavioral aspects of a system [28]. Because of its wide use, usability, and effectiveness, test case generation methods based on UML are highly popular [29]. One such method [44] uses state charts to create FSMs using a tool called PerformCharts. Then, the FSMs are fed into Condado, a graph theory-based test case generation tool, to automatically generate test cases that cover all possible transitions. A similar approach converts state charts into an intermediate graph which is then traversed based on various coverage criteria to come up with test cases [45]. Another method [46] extracts data from class diagrams, sequence diagrams, and state diagram to automatically generate test cases. Swain et al. proposed a framework [47] that combines state models and activity models to create a state activity graph which is then searched using DFS to generate test cases.

Numerous UML-based test case generation methods use activity diagrams as the basis to generate test cases. For instance, one method uses an exhaustive search and a test queue prioritization technique to identify critical test cases [48]. Another method combines Tabu Search with test cases originating from activity diagrams to generate test cases [49]. Stallbaum et al. used risk-based prioritization to generate test cases [50]. The EasyTest method converts activity diagrams to activity dependency tables, and then into activity dependency graphs, and traverses using a DFS-based algorithm to come up with test paths and subsequently test cases [51]. Another approach to UML-based test case generation is to utilize use case diagrams as the basis to generate the test cases. For example, one technique utilizes use case simulations to build test objectives and sequence diagrams to generate test cases from test objectives [52]. Another technique uses case and sequence diagrams to create a system testing graph which is then traversed to generate test cases [53]. Raza et al. proposed a framework that uses the Interactive Overview Diagram and a series of matrix generations to generate test cases for a specific coverage criteria [54]. Prasanna et al. came up with a framework [55] that uses object diagrams and genetic algorithms' tree cross over technique to generate test cases exhaustively. Then, the DFS algorithm is used to extract the test paths.

In summary, the UML-based MBT methods either utilize a single diagram or multiple diagrams from the system model to generate test cases. The resulting test case trees are traversed using a search algorithm to identify the test paths. Some frameworks go a step further and prioritize test cases so that more emphasis can be given to most critical test cases. Similar to UML, the system representation in the HEFFR framework uses a combination of graphs to generate the system model. The similarity in the system representation and the benefits of MBT guided us toward using UML-based test case generation as a basis for this research. However, we cannot use them directly because the system representation in UML is different than in HEFFR. As a result, we have studied the process that the UML-based approaches have taken and applied it in this research to automatically generate fault scenarios for the HEFFR framework. We use the CFG, ASGs, the component behavior model, and the action classification to generate use cases that represent a majority of potential component- and human-related faults

Table 1 Comparison between risk assessment methods with automated scenario generation and the proposed work

	Ref. [35]	Ref. [36]	Ref. [37]	SimpraPlan [38]	IBFM [21]	Ref. [20]	HEFFR
Ability to generate scenarios relating to components	✓	✓	✓	✓	✓	✓	✓
Ability to generate scenarios relating to human	X	X	X	X	X	X	✓
Applicability for risk assessment	✓	X	✓	✓	✓	✓	✓
Applicability during early design stages	X	X	X	X	✓	✓	✓
Ability to generate a majority of fault scenarios involving both human and components	X	X	X	X	X	X	✓

that the system could experience during its lifecycle. A modified DFS algorithm is then used to search through the use cases and to evaluate their system-level impact and propagation paths. Section 3 describes the automated scenario generation method and its implementation with the HEFFR framework in detail.

3 Methodology

The objective of this research is to develop an automated scenario generation technique that covers a majority of the component- and human-related fault conditions so that a comprehensive failure analysis can be conducted using HEFFR. We use the behavior model and the action classifications to generate fault conditions using transition functions and a modified DFS. DFS is a tree or graph search algorithm that searches through a branch as far as possible before moving on to the next branch [56]. Each level of the branch is considered as a time-step and each branch as an input scenario for HEFFR. A HEFFR analysis is done at every time-step to check if the predefined critical functions are lost. If the functions are lost, the search stops and the path (branch) and the results (failures and propagation paths) are stored to a file and the search moves to the next branch. The search continues until all branches are evaluated.

3.1 Transition Function. The transition function is a set of rules that are used to create the child nodes for each mother node. For our application, these rules change the behavior mode (a state the component can be in, e.g. for a pipe, it can be leaking, clogged, or working as expected) of the components to induce faults. One of the rules makes no change to the mother node, creating a child node that is exactly the same as the mother. This is done to study the propagation of faulty behavior without introducing any further behavior modes. However, the number of time-steps or number of consecutive child nodes that the "no change rule" is applied is limited by a user-defined number. The rest of the rules start with changing the behavior state of one component at a time until all behavior states for each component are applied. Then, the behavior modes are changed for two components at a time until all component combinations are done. This process increments until the behavior modes of all components are changed to make sure that all possible combinations are executed. For example, if the behavior modes of components A, B, and C are A1, A2, B1, B2, and C1, C2 respectively, the rules applied and the resulting branches for mother node A1B1C1 are shown in Fig. 2. In order to avoid explosion of scenarios, once a faulty behavior mode is introduced for a component, reverting back to a nominal state is not allowed. However, the transition rules do not stop child nodes from going to a previously analyzed faulty behavior state (i.e., if a pipe is leaking, it is allowed to go to clogged and back to leaking in future time-steps. But it is not allowed to go back to nominal in the future). In reality, if a failure is present, usually, it does not go away unless it is repaired. However, one failure can propagate to another. In rare occasions, external influence can cause temporary faulty behaviors in components. Such malfunctions go away as the influencing factors resolve. For example, cold weather can cause fluid to freeze and clog fluid flow in a pipe until the weather improves. In such cases, designers are encouraged model those temporary behaviors as a subgroup of nominal behaviors to allow the algorithm to switch back and forth between the temporary and nominal behaviors.

When there are components that interact with the human, the rules are slightly modified. The behavior mode generation for the component still stays the same. However, the child nodes are allowed to go back and forth between nominal and faulty behavior for human-induced behavior modes, because these behaviors do not involve mechanical failures and they only depend on the human actions. When a non-human-induced behavior is present, child nodes are not allowed to go to human-induced behaviors anymore. When a non-human-induced fault such as a mechanical failure is present in a component, a humans' interaction with that component will not alter the mechanical failure or the system unless that component is repaired. Hence, considering human actions for such behavior modes does not add any value. Note that the action classifications (results of human actions) are not used to generate scenarios. Instead, the resulting component behavior modes are used to generate scenarios. The HEFFR framework assumes that the human can only interact with the system through its components. Thus, the human-induced behavior modes of the components cover the system-level effects of human error. Hence, we have chosen not to consider action classifications during scenario generation because this will only increase the possible combinations and not add any value in terms of understanding the system-level effects of human error.

Action classifications provide valuable information on how human-induced behaviors are produced. In order to give designers more details on what specific human actions contribute to human-induced behaviors, the algorithm does the following. When a human-induced behavior mode is present, all possible combinations of action classifications that can result in that specific behavior mode are generated using the action simulation and presented with the results of the overall simulation. Even when the behavior mode is nominal, combinations of action classifications are provided to the users to make sure that any human errors that fail to propagate to affect the component behavior do not go unnoticed. When generating the action classification combinations, the algorithm checks to see if the action classification combination is viable. For instance, one cannot see an invisible object. The algorithm checks for these relationships using the ASG and if combinations that violate these relationships are present, they are omitted. These steps make sure that the combinations presented to the user are as realistic as possible.

3.2 Critical Event Scenario Generation and Evaluation. The overall process follows the following steps. Note that the modules from HEFFR is not explained here since they have already been well documented in previous research [18,19,57].

3.2.1 Initialization. The number of consecutive time-steps the "no change rule" can be applied is retrieved from the user. The users are advised to consider the maximum number of time-steps required to enforce the loss of the critical functions when choosing the number of consecutive time-steps. For instance, let us assume that the designers have modeled a hydraulic braking system in a

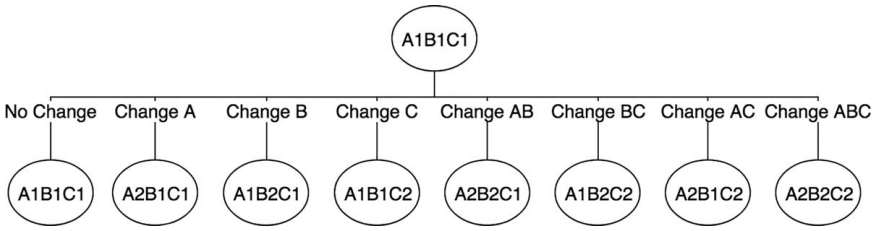

Fig. 2 An example of an application of the transition function

way where a leak in the line will cause the fluid to empty in five time-steps and the brake pad will wear down by 10% on each time-step if there is a faulty behavior in the caliber. The number of time-steps the "no change rule" should be applied should be ten instead of five because it is the maximum number of time-steps it will take for the braking function to fail. Also, the maximum number of time-steps that need to be simulated or the lowest level a branch can go to is also read from the user. These inputs make sure that the algorithm does not get stuck in a branch indefinitely. Next, the critical functions are read from the users. The behavior modes of all components and the action classifications of all actions are initialized to a nominal state. This will serve as the mother seed for the DFS. Also, the time-step is initialized to zero.

3.2.2 Goal State. The algorithm continues through a branch until a goal state is achieved. The goal state in this case is the failure of all critical functions. When a goal state is achieved, the algorithm writes the path (critical event scenario input to HEFFR) and the results form HEFFR (functional failures, human errors, and their propagation paths) to an output file.

3.2.3 Execution of the Transition Function. Two markers are used to track the application of the rules. They track the rules applied down and across a branch. When the transition rule is applied, these markers are updated. If the transition rule is the "no change rule," the algorithm checks if it has been applied for the maximum allowable consecutive applications. If it has, it moves on to the next rule. When a transition rule is applied, HEFFR analysis is conducted to check if the critical functions have failed. If the critical functions have failed, the path of the behavior modes (i.e., branch) and the time-steps are written to the output file. The corresponding HEFFR results are also written to the output file. Then, the search moves to the previous level and the next transition rule is applied. The above process is iterated until there are no more rules to be applied in level 1. Every time the search moves to a new branch, the last time-step from that branch is picked up and incremented for the new child nodes.

If at least one of the critical functions has not failed, the child node becomes the new mother node and it is stored with its corresponding time-step. Then, transition rules are applied to the new mother seed and the process above is repeated. If there are no rules to be applied at the current level, the search moves back one level and follows the above steps. Also, if the number of time-steps reaches the maximum allowable time-steps, the search returns to the previous level and the above steps are followed. This process is shown in a high level flowchart in Fig. 3.

3.2.4 Understanding the Results. The output file contains the fault scenarios with corresponding time-steps and the failures, human errors, and their propagation paths. We have chosen to leave the data in its raw form because it will give the designers the flexibility to analyze for what they are looking for specifically. For instance, if one is using this framework to identify the behavior modes that are most vulnerable in terms of having an effect on certain functions, they can define those functions as critical functions and look at what behavior modes were involved in the shortest paths that caused those functions to fail. Similarly, if one wants to compare alternative designs, they can compare the data from the runs for the alternatives to see which designs had the longest paths to failure on average. With the emergence of big data and advances in data science, there are a wide variety of tools to extract information. Hence, we present as much data as possible to designers so they can use such tools to extract information that is tailored to their needs.

4 Working Example

We have chosen a liquid tank design problem to demonstrate the application and explore the capabilities and limitations of the proposed method. Different versions of this problem have appeared as case studies in various research studies [12,57–61]. The

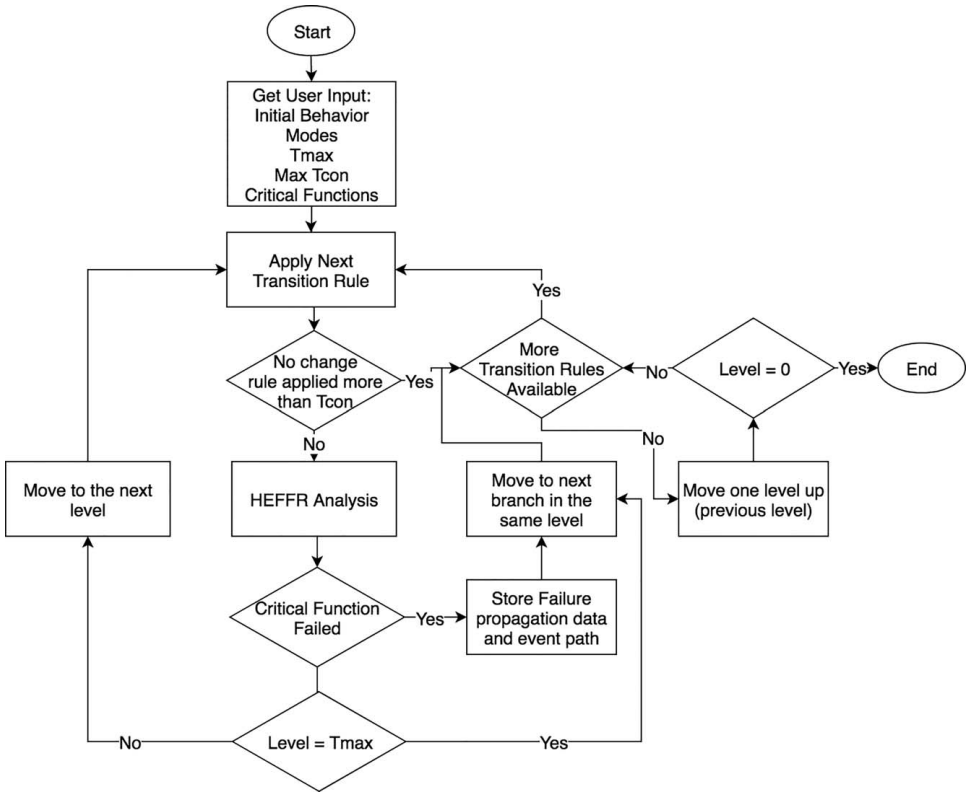

Fig. 3 A high-level flowchart of the proposed approach

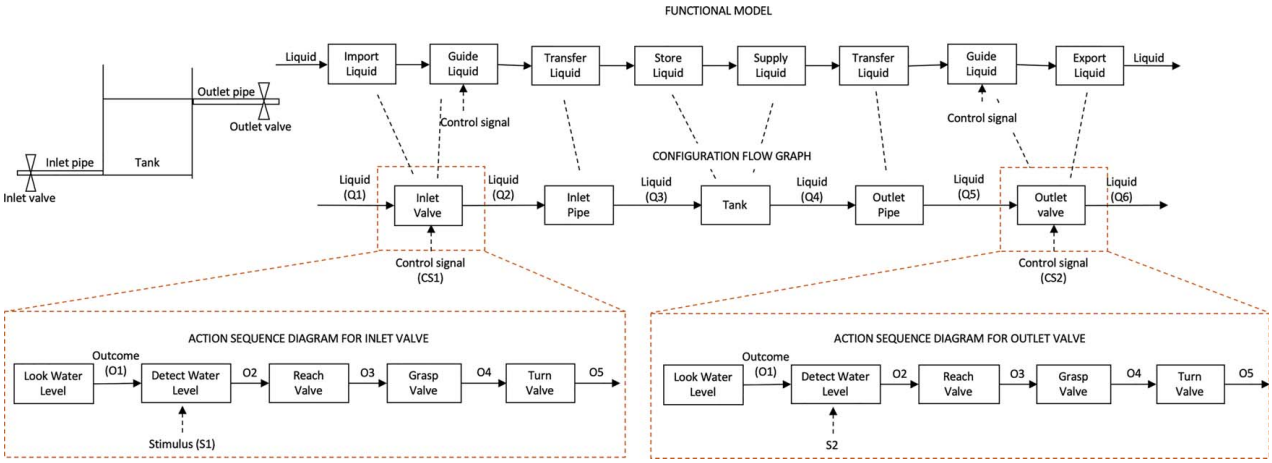

Fig. 4 The system representation of the liquid tank, where the red boxes represent the human involvement [57]

problem is to design a liquid tank that holds its water level from either dropping below a minimum threshold or going above a maximum threshold. First, a system model is created to represent the conceptual design of the liquid tank. The functional model, CFG, ASGs, and a schematic of the system are presented in Fig. 4. The system consists of two valves, two pipes, and a tank. An operator is expected to shut off the inlet valve if the water level is above the maximum threshold and shut off the outlet valve if the water level is below the minimum threshold. Often, in complex engineered systems such as aircrafts and nuclear power plants, operators are expected to monitor displays and gauges, and operate/interact with the system based on the information that is present [62–64]. This case study was chosen as a simplified version of such scenarios. The functions, corresponding components, and their behavior modes are shown in Table 2. The human-induced behavior modes are highlighted using bold text. For components that require human interactions (inlet and outlet valve), the actions from ASGs and their classifications are given in Table 3.

For this study, we chose the number of time-steps a scenario should be allowed to propagate (number of times the "no change rule" should be applied) to two. This is because the behavior of the tank was set such that it would take two time-steps to overflow or dry out depending on the flow of the liquid. Having more than two consecutive time-steps is redundant since if the function store liquid were to fail due to a specific fault, it would in two time-steps. Hence, analyzing the propagation of the same failure anymore does not add any further value. We chose the maximum number of time-steps as five. Since the simulation continuously introduces faults at each time-step, no single fault scenario can be repeated for more than two consecutive time-steps, and the tank behavior drives the failure of function store liquid within two time-steps, ideally, having up to four time-steps would have revealed a majority of worse case fault scenario combinations. We chose five time-steps so that we analyze a step further to uncover any unforeseen fault conditions. Note that in order to analyze five time-steps (until $t = 5$), six time-steps need to be analyzed in total because the simulation starts at $t = 0$. Analyzing any further will introduce repetition of faults from previous sets of time-steps. For instance, a scenario set that was present between time-steps one and three may be repeated from time-steps five to seven. Since the simulation is time-based, the number of potential scenarios is infinite if such repetitions are allowed. Hence, to avoid the explosion of scenarios, it is up to the user to choose the number of total time-steps and number of time-steps, the "no change rule" can be applied wisely by considering critical functions and the behavior modes of the related components.

Table 2 Functions, corresponding components, and their behavior modes

Function	Component	Behavior states
Import Liquid Guide Liquid Export Liquid	Valve	**Nominal Open, Nominal Close, Failed Open, Failed Close**, Stuck Open, Stuck Close
Transfer Liquid Supply Liquid	Pipe	Nominal, Clogged, Leaking
Store Liquid	Tank	Nominal, Leaking

Table 3 Action classifications of all involved actions

Actions	Classifications			
Look	Visible	Not visible		
Detect	Detected—Nominal	Not detected—Nominal	Detected—Failed	Not detected—Failed
Reach	Reached—Nominal	Reached—Failed	Cannot reach	No action
Grasp	Grasped	Cannot grasp	No action	
Turn	Turn to close	Turn to open	Cannot turn	No action

When creating faults for time-steps, no temporary component malfunctions were considered (i.e., once a non-human-induced faulty behavior mode was introduced for a component, it was not allowed to go back to a nominal behavior mode. However, human-induced behaviors were allowed to go back and forth between nominal and faulty behaviors). When generating action classification combinations that can result in human-induced behaviors the following rules were included to keep the scenarios realistic.

- The operators cannot detect a signal without being able to see it. Hence, When the action classification of the action see water level is "Not Visible," combinations with action detect not equal to "Not Detected—Failed" were omitted.
- One cannot grasp an object without reaching it first. So, when the action classification for the action reach meant that the operator did not reach the valve, combinations with action grasp not equal to "No Action" were omitted.
- One cannot turn a valve without grasping it. Hence, when the action classification of the action grasp meant that the operator did not grasp the valve, the combinations with action turn not equal to "No Action" were omitted.

By executing this case study, we intend to explore if the algorithm is capable of creating effective use cases that cover a wide range of fault scenarios that involve component failures and

human errors. Then, we study the results to see if the use cases were useful in identifying potential human errors, component failures, and their propagation paths by answering questions such as what are the fault scenarios that affect the critical functions of the system the fastest? In this case, we have chosen the *Store Liquid* function as the critical function because its loss means that the water level is either too low or too high. While the loss of other functions may result in the tank not maintaining its liquid level, the loss by themselves does not mean that the liquid level is not at a desired level. We also intend to identify the behavior modes that have the highest chance of affecting the critical function. We do this by calculating the percentage of scenarios (of all scenarios identified to cause the critical function to fail) with each type of faulty behavior modes. Overall, we try to understand if the automated scenario generation method helps overcome the previously mentioned shortcomings of HEFFR. Section 5 details the results from the analysis.

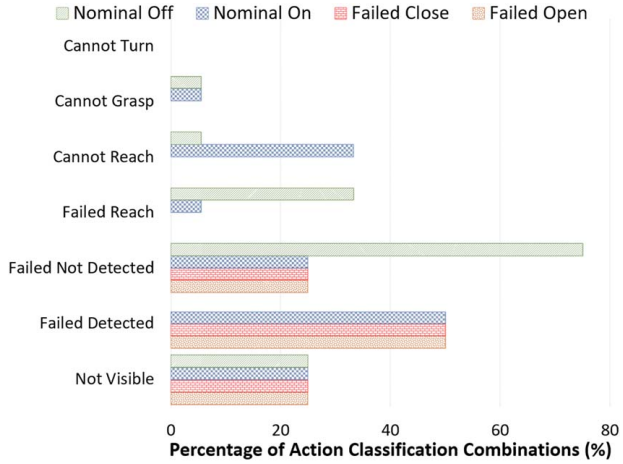

Fig. 5 The percentage of action classification combinations with each type of human error

5 Results

The simulation begins by taking the critical function as an input. Next, the behavior modes of all components are set to nominal—*Nominal Open* for valves, *Nominal* for pipes, and *Nominal* for the tank. The initial flow and the water level of the tank are then set to nominal. Then, the number of time-steps a new node is allowed to propagate ("no change rule" is applied) and the total number of time-steps are set. Once these inputs are read, the algorithm begins to evaluate critical event scenarios automatically. The execution took around 28 min on a personal laptop (IntelCore i5, 2.9 GHz speed, and 16 GB RAM) which is reasonable considering the amount of information that can be extracted. In total, around 15 million event scenarios resulted in the function store liquid failing in $t = 5$. Only two scenarios were found to have purely human-induced faulty behaviors whereas 163,204 scenarios had purely non-human-induced behaviors. The rest of the scenarios had a combination of both non-human and human-induced faulty behaviors.

Out of the 1824 possible action classification combinations, only 152 were generated. The rest were omitted (based on the rules defined in Sec. 4) because they were not realistic. Out of the generated action classification combinations, four each resulted in human-induced behaviors: *Failed Open* and *Failed Close*. There were 72 combinations each for *Nominal On* and *Nominal Off*. Additionally, 86% of the action classification combinations that contributed toward *Nominal On* and *Nominal Off* had underlying human errors (actions in faulty classifications, for example, cannot turn—when an attempt to turn is made and cannot be physically achieved, but not due to a component failure) for at least one action meaning that these errors did not propagate to affect the system. However, they must be considered when making design decisions because they may affect the system as the design evolves. Seventy-five percent of all combinations that led to *Nominal Off* had the human error *Failed Not Detected*. Similarly, detect-related human errors were prevalent across all behavior modes (50% *Failed Detection* in *Failed Open*, *Failed Close*, and *Nominal On*) followed by reach, grasp, and vision-related errors. The details of this analysis are shown in Fig. 5, where the percentages are calculated by considering the number of times an action classification was present in the action classification combinations that could result in a specific human-induced behavior.

The shortest path for failure was at four time-steps ($t = 3$). There were 10,459 event scenarios that led to the failure of the store liquid function in four time-steps. All of the event scenarios had the failure type *Leak* for the component tank at least once. This is expected because the function *Store Liquid* is directly fulfilled by the tank. A leaking outlet pipe and a clogged inlet pipe were other commonly occurring failures (79% and 83%, respectively). *Failed open* was most common for the outlet valve (78%) and *Failed Close* was most common for the inlet valve (78%) among human-induced behavior modes. The prevalence of the behaviors *Leak* tank, *Clogged* inlet pipe, and *Failed Open* outlet valve indicate that a majority, if not all of these event scenarios resulted in a tank dry out. The detailed analysis of the presence of each faulty behavior mode in the event scenarios with the shortest path to critical function failure are shown in Fig. 6.

Using this data, designers may make design decisions to avoid the leak in the tank and the outlet pipe by trying different materials, adjusting wall thickness, or recommending additional testing of these components. Similarly, they may choose to add sensors to detect clogs in the outlet pipe or check the quality of the liquid to make sure that there is no residue build-up. For the human-induced behaviors, detect-related errors are most common followed by reach-, grasp-, and vision-related errors. To prevent detect-related errors, designers may choose to make changes to the system by adding redundant signals or making signals more salient. On the other hand, they may suggest training to improve the operators ability to detect signals. Designers may choose to conduct DHM analyses such as reach analysis and percent vision obscuration to identify ways to mitigate non-cognitive human errors. Depending on the workspace design, they may also choose to perform ergonomic assessments such as comfort, lower back compression force, and biomechanics. They may apply human factors engineering guidelines, suggest training, or device operational procedures to mitigate all types of human errors. In contrast, they may not resort to design decision yet and further analyze the data. They may choose to find out how the behavior of other functions contribute to the failure of the tank or repeat the execution with different critical functions with different input conditions. Either way, when design changes are made, they may update the system model and iterate through this process until a satisfactory design is derived. Note the HEFFR framework, being an early design stage tool, can only model changes the system functions, components, and human action sequences. It cannot represent changes to component parameters (such as thickness and material properties) and human action parameters (change in anthropocentric and environmental conditions, etc.), which usually come to light during later design stages. Hence, the analysis should only be iterated when such changes are made to the design.

6 Discussion

The working example presented earlier shows how the automated scenario generation can be used with HEFFR to perform a comprehensive analysis to identify potential functional failures, human errors, and their propagation paths early in design. The presence of both human-induced and non-human-induced behaviors in the

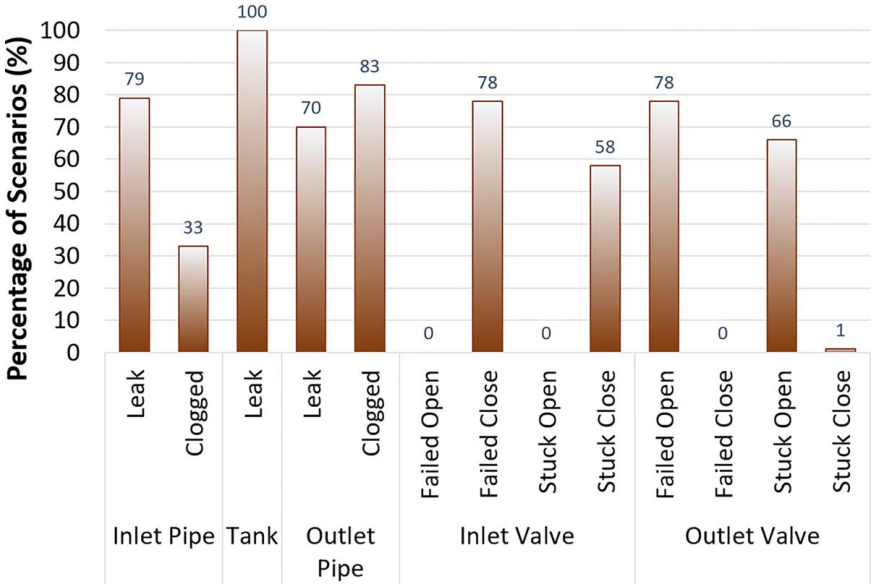

Fig. 6 The percentage of total event scenarios with each type of faulty behavior mode

scenarios indicates that the automatically generated scenarios included fault conditions involving both humans and components. The results showed the importance of avoiding behavior modes such as *Leak* tank and *Clogged* input valve to mitigate the potential loss of the function *Store Liquid*. Among human actions, detection-related human errors were most prevalent stressing the need for mitigating such errors. Additionally, human errors were present even when the components were in nominal behavior modes. Even though these human errors did not propagate to affect the system under the circumstances the system was analyzed, it is important to consider them when making design decisions because they may be exposed in different circumstances.

The above results show the ability of the algorithm to generate a broad spectrum of fault scenarios involving both components and humans that can violate the critical function of the system. Only a small percentage of generated scenarios had purely human-induced behaviors or non-human-induced behaviors, meaning that if human errors or component failures were evaluated in isolation, a majority of potential fault inducing conditions would have been missed. This shows the importance of assessing the effects of both component failures and human errors acting in combination to better understand potential risk and promote appropriate mitigation strategies. The data analysis presented in this paper is minimal when compared with the information that can be extracted from the data. Only, $t = 3$ was analyzed in this example. Further analysis can be done at $t = 4$ or $t = 5$ to identify how the system behaves when worse case scenarios are not present. Such an analysis can give important information on how the system will operate under more regular and less severe fault conditions. The ability to identify human errors that do not propagate to affect the system is another plus since most system failures occur through failed to detect, compounding failures. Overall, the proposed approach allows for a comprehensive failure reasoning through input scenarios involving the fallibilities of both humans and components. The results can be used to conduct various types of analyses, depending on the needs and requirements of the designers.

The performance-related measures such as the execution time and the number of scenarios that caused the critical function to fail of this case study do not necessarily reflect the overall performance of the algorithm. Considering the amount of information that can be extracted and how useful they can be to design a safer, more reliable system, the execution time for this problem was acceptable. However, the execution times surrounding more complex engineered systems and if they outweigh the benefit of the information received is yet to be seen. A large number of event sequences were identified to violate the critical function. We may never know if these numbers mean that the scenarios included all of the possible combinations of human errors and component failures since there is no such data to compare the results with. However, one can be assured that a majority of such fallibilities were covered because the algorithm evaluated all possible combinations of behavior modes at least once and the large volume of results mean that different combination of these behavior mode combinations was evaluated.

The number of total time-steps and the number of time-steps where the same scenario can be executed consecutively play a significant role in the total number of scenarios evaluated. In fact, the number of scenarios increases exponentially with each time-step. Since the simulation is time-based, there is nothing that limits the number of scenarios except for the time. While having a lot of data is important to be able to extract a wide variety of information, repetitive data in large volumes can make this process slow, resource intensive, and at worse impossible. Designers are encouraged to consider the critical functions, the component, and the behavior modes that can induce a failure carefully to minimize the time-related variables. For example, in the liquid tank case study, the behavior mode of the tank dictated that it will take two time-steps for the tank to dry out or overflow from a nominal level. That determined the total number of consecutive steps as two and total time-steps as five. If the behavior mode of the valve only required one time-step for a failure to occur and a designer was evaluating the function related to the valve as the critical function, he or she may choose one or two for consecutive time-steps and two or three for total time-steps.

Another major contributor to the breadth of the search tree is the behavior modes of individual components. As the behavior modes increase, the number of combinations of potential scenarios increases exponentially, which in turn increases the total number of available scenarios. Hence, users are encouraged to carefully consider behavior modes and avoid any redundancy. Based on the cases presented above, one can argue that the proposed scenario generation method is vulnerable to data explosion (i.e., too many scenarios being created). However, these vulnerabilities encourage designers to think about the behaviors of the components that will be part of the system they are designing early in the design process, which can lead to well thought out designs. Additionally,

the explosion of the scenarios can be avoided if the contributing variables are handled carefully.

Another way to avoid data explosion is to take a systems-of-systems approach into the analysis. Complex systems can be broken down to less complex subsystems. The proposed approach can be applied to these subsystems to identify the potential vulnerabilities of them. The resulting data can be used to construct the whole complex engineered system in which the black box functions of each subsystem will build the functional model, the subsystems themselves will be components in the CFG, and the subsystems that interact with human will have ASGs. Then, the proposed automated scenario generation-based component failure and human error reasoning framework can be applied to the new system model. Such an approach will help designers pay attention to individual components in more detail while making sure that the overall system vulnerabilities and human fallibilities are addressed. Overall, the proposed automatic test case generation technique overcomes some of the shortcomings of the HEFFR framework. The scenarios generated, cover a wide variety of failure conditions involving both humans and components. This approach also allows the analysis of a large number of scenarios at the same time. The proposed automated scenario generation technique may be prone to data explosion. However, data explosion can be avoided and the path to mitigation will help designers come up with more thought out concepts.

7 Conclusions and Future Work

This paper introduced a novel approach to automatically generate critical event scenarios that cover a majority of component and human fallibilities present within a system to identify potential component failures, human errors, and their propagation paths during early design stages. The automatic scenario generation is achieved by searching through the behavior model and the action classifications using a depth first search. The Human Error and Functional Failure Reasoning (HEFFR) framework is used to identify if the critical functions of the systems will be lost due to the generated event scenarios, and such scenarios and their failure and human error propagation paths are presented through an output file. The designers are encouraged to use data mining techniques to extract information that is required to fulfill their specific needs. The proposed approach is demonstrated using a liquid tank study. Finally, the results from the liquid tank study are analyzed to understand the capabilities and limitations of the proposed approach.

This study only applies the proposed approach once to check if one critical function is failing in a simple problem. Hence, the reported execution time and the failure scenarios do not particularly shed light on the overall performance algorithm. As future work, we will apply this approach to a more complex system with multiple executions and use the results to improve performance such as execution time and the number of scenarios executed. The study presented in this paper does not verify the validity of the proposed work. The performance of the approach presented in this paper will be compared with existing methods in the future to study the validity and general usability of the automated scenario generation technique. Another area of future work will study results from multiple executions to look into ways to streamline the transition rules so that the number of scenarios executed is minimized while optimizing scenario coverage to include a majority of component failures and human errors. Streamlining the transition rules will help with improving the overall performance of the automated scenario generation approach introduced in this paper.

Another area of future work is aimed at introducing a quantitative measure to the analysis to aid risk-based decision-making. Previous research has introduced Functional Failure Reasoning [65] which improves Functional Failure Identification and Propagation by quantifying risk. Introducing such risk quantification to the proposed approach will aid designers to understand the most vulnerable functions, and what type of events contribute to such vulnerabilities and help them make decisions to mitigate the risk of such vulnerabilities by preventing the event scenarios that contribute toward them from happening. Additionally, quantitative scores can be used to compare alternative designs and make trade-offs. The ultimate goal is to provide designers with a tool that would aid them design more reliable systems with improved performance and safety.

Acknowledgment

This research is supported by The National Aeronautics and Space Administration (NASA) award number 80NSSC18K0940. 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.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The datasets generated and supporting the findings of this article are obtainable from the corresponding author upon reasonable request.

References

[1] Kohn, L. T., Corrigan, J. M., and Donaldson, M. S., 2000, *To Err is Human: Building a Safer Health System*, Vol. 6, National Academies Press, Washington, DC.

[2] Högberg, L., 2013, "Root Causes and Impacts of Severe Accidents At Large Nuclear Power Plants," Ambio, **42**(3), pp. 267–284.

[3] Wiegmann, D. A., and Shappell, S. A., 2001, "Human Error Analysis of Commercial Aviation Accidents: Application of the Human Factors Analysis and Classification System (HFACS)," Aviat. Space Environ. Med., **72**(11), pp. 1006–1016.

[4] Neitzel, R. L., Seixas, N. S., and Ren, K. K., 2001, "A Review of Crane Safety in the Construction Industry," Appl. Occup. Environ. Hyg., **16**(12), pp. 1106–1117.

[5] Meshkati, N., 1991, "Human Factors in Large-Scale Technological Systems' Accidents: Three Mile Island, Bhopal, Chernobyl," Ind. Crisis Q., **5**(2), pp. 133–154.

[6] Demirel, H. O., 2015, "Modular Human-in-the-Loop Design Framework Based on Human Factors," PhD thesis, Purdue University, West Lafayette, IN.

[7] Norman, D., 2013, *The Design of Everyday Things: Revised and Expanded Edition*, Constellation, New York.

[8] Ullman, D. G., 2010, *The Mechanical Design Process: Part 1*, 2nd ed., McGraw-Hill, New York.

[9] Mil-Std-1629A, 1980, Technical Report, Department of Defense, Washington DC.

[10] Vesely, W. E., Goldberg, F. F., Roberts, N. H., and Haasl, D. F., 1981, *Fault Tree Handbook*. Technical Report, Nuclear Regulatory Commission, Washington DC.

[11] Ericson, C. A., 2005, "Event Tree Analysis," *Hazard Analysis Techniques for System Safety*, John Wiley Sons, Hoboken, NJ, pp. 223–234.

[12] Kurtoglu, T., and Tumer, I. Y., 2008, "A Graph-Based Fault Identification and Propagation Framework for Functional Design of Complex Systems," ASME J. Mech. Des., **130**(5), p. 051401.

[13] Huang, Z., and Jin, Y., 2008, "Conceptual Stress and Conceptual Strength for Functional Design-for-Reliability," ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Brooklyn, NY, Aug. 3–6, American Society of Mechanical Engineers, pp. 437–447.

[14] Embrey, D., 1986, "Sherpa: A Systematic Human Error Reduction and Prediction Approach," Proceedings of the International Topical Meeting on Advances in Human Factors in Nuclear Power Systems, Knoxville, TN, Apr. 21–24, pp. 184–193.

[15] Swain, A., 1964, "Therp Technique for Human Error Rate Prediction," Proceedings of the Symposium on Quantification of Human Performance, Albuquerque, NM, Aug. 17–19.

[16] Ahmed, S., Demirel, H. O., Tumer, I. Y., and Stone, R. B., 2018, "Towards Human-Induced Failure Assessment During Early Design," Tools and Methods of Competitive Engineering (TMCE 2018), Las Palmas de Gran Canaria, Spain, May 7–11, Delft University, pp. 507–520.

[17] Zurita, N. F. S., Stone, R. B., Demirel, O., and Tumer, I. Y., 2018, "The Function-Human Error Design Method (FHEDM)," ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Quebec City, Quebec, Canada, Aug. 26–29, American Society of Mechanical Engineers, p. V007T06A058.

[18] Irshad, L., Ahmed, S., Demirel, H. O., and Tumer, I., 2019, "Computational Functional Failure Analysis to Identify Human Errors During Early Design Stages," ASME J. Comput. Inf. Sci. Eng., **19**(3), p. 031005.
[19] Irshad, L., Ahmed, S., Demirel, O., and Tumer, I. Y., 2019, "Coupling Digital Human Modeling with Early Design Stage Human Error Analysis to Assess Ergonomic Vulnerabilities," AIAA Scitech 2019 Forum, San Diego, CA, Jan. 7–11, p. 2349.
[20] Papakonstantinou, N., Sierla, S., O'Halloran, B., and Tumer, I. Y., 2013, "A Simulation Based Approach to Automate Event Tree Generation for Early Complex System Designs," ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Portland, OR, Aug. 4–7, American Society of Mechanical Engineers, p. V02BT02A008.
[21] McIntire, M. G., Keshavarzi, E., Tumer, I. Y., and Hoyle, C., 2016, "Functional Models With Inherent Behavior: Towards a Framework for Safety Analysis Early in the Design of Complex Systems," ASME 2016 International Mechanical Engineering Congress and Exposition, Phoenix, AZ, Nov. 11–17, American Society of Mechanical Engineers, p. V011T15A035.
[22] Mercurio, D., Podofillini, L., Zio, E., and Dang, V., 2009, "Identification and Classification of Dynamic Event Tree Scenarios Via Possibilistic Clustering: Application to a Steam Generator Tube Rupture Event," Accid. Anal. Prev., **41**(6), pp. 1180–1191.
[23] Papazoglou, I. A., 1998, "Functional Block Diagrams and Automated Construction of Event Trees," Reliab. Eng. Syst. Saf., **61**(3), pp. 185–214.
[24] Sen, D. K., Banks, J. C., Maggio, G., and Railsback, J., 2006, "Rapid Development of An Event Tree Modeling Tool Using Cots Software," 2006 IEEE Aerospace Conference, IEEE, Big Sky, MT, Mar. 4–11, p. 8.
[25] Smith, C., Knudsen, J., Kvarfordt, K., and Wood, T., 2008, "Key Attributes of the Saphire Risk and Reliability Analysis Software for Risk-Informed Probabilistic Applications," Reliab. Eng. Syst. Saf., **93**(8), pp. 1151–1164.
[26] Rutt, B., Catalyurek, U., Hakobyan, A., Metzroth, K., Aldemir, T., Denning, R., Dunagan, S., and Kunsman, D., 2006, "Distributed Dynamic Event Tree Generation for Reliability and Risk Assessment," 2006 IEEE Challenges of Large Applications in Distributed Environments, Paris, France, June 19, IEEE, pp. 61–70.
[27] Farooq, U., Lam, C. P., and Li, H., 2008, "Towards Automated Test Sequence Generation," Software Engineering. 2008, 19th Australian Conference on ASWEC 2008, Perth, WA, Australia, Mar. 26–28, IEEE, pp. 441–450.
[28] Chevalley, P., and Thévenod-Fosse, P., 2001, "Automated Generation of Statistical Test Cases From Uml State Diagrams," Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International, Chicago, IL, Oct. 8–12, IEEE, pp. 205–214.
[29] Offutt, J., and Abdurazik, A., 1999, "Generating Tests From Uml Specifications," International Conference on the Unified Modeling Language, Fort Collins, CO, Oct. 28–30, Springer, pp. 416–429.
[30] Irshad, L., Demirel, H. O., and Tumer, I. Y., 2019, "Using Automated Use Case Generation for Early Design Stage Functional Failure and Human Error Analysis," ASME Paper No. DETC2019-98466.
[31] Irshad, L., Onan Demirel, H., Tumer, I. Y., and Brat, G., 2020, "Using Rio-Paris Flight 447 Crash to Assess Human Error and Failure Propagation Analysis Early in Design," ASCE-ASME J. Risk Uncert Eng. Syst. Part B Mech. Eng., **6**(1), p. 011008.
[32] Cunning, S. J., and Rozenblit, J. W., 2005, "Automating Test Generation for Discrete Event Oriented Embedded Systems," J. Intell. Rob. Syst., **41**(2–3), pp. 87–112.
[33] Junghanns, A., Mauss, J., and Tatar, M., 2008, "Tatar: Testweaver – A Tool for Simulation-Based Test of Mechatronic Designs," 6th International Modelica Conference, Bielefeld, Mar. 3, Citeseer, Qtronic Gmbh, Alt-moabit D, D- Berlin.
[34] Hilf, K.-D., Matheis, I., Mauss, J., and Rauh, J., 2010, "Automated Simulation of Scenarios to Guide the Development of a Crosswind Stabilization Function," IFAC Proc. Volumes, **43**(7), pp. 768–772.
[35] Snooke, N. A., Price, C., Downes, C., and Aspey, C., 2015, "Automated Failure Effect Analysis for PHM of UAV," Proceedings of the International System Safety Regional Conference (ISSRC 2008), R. J. Simmons, ed., Singapore, Apr. 23–25, International System Safety Society, pp. 28–37.
[36] Struss, P., 2006, "A Model-Based Methodology for the Integration of Diagnosis and Fault Analysis During the Entire Life Cycle," IFAC Proc. Volumes, **39**(13), pp. 1157–1162.
[37] Liggesmeyer, P., and Rothfelder, M., 1998, "Improving System Reliability with Automatic Fault Tree Generation," Digest of Papers. Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing (Cat. No. 98CB36224), Munich, Germany, June 23–25, IEEE, pp. 90–99.
[38] Nejad, H., and Mosleh, A., 2005, "Automated Risk Scenario Generation Using System Functional and Structural Knowledge," ASME 2005 International Mechanical Engineering Congress and Exposition, Orlando, FL, Nov. 5–11, American Society of Mechanical Engineers, pp. 85–89.
[39] Blackburn, M., Busser, R., and Nauman, A., 2004, "Why Model-Based Test Automation is Different and what You Should Know to Get Started," International Conference on Practical Software Quality and Testing, Washington, DC, Mar. 22–26, pp. 212–232.
[40] Auguston, M., Michael, J. B., and Shing, M.-T., 2005, "Environment Behavior Models for Scenario Generation and Testing Automation," ACM SIGSOFT Software Engineering Notes, **30**(4), pp. 1–6.
[41] Xu, D., Xu, W., Kent, M., Thomas, L., and Wang, L., 2015, "An Automated Test Generation Technique for Software Quality Assurance," IEEE Trans. Reliab., **64**(1), pp. 247–268.
[42] Wang, R., Kristensen, L. M., Meling, H., and Stolz, V., 2019, "Automated Test Case Generation for the Paxos Single-Decree Protocol Using a Coloured Petri Net Model," J. Logical Algebraic Methods Program., **104**, pp. 254–273.
[43] Matinnejad, R., Nejati, S., Briand, L., and Bruckmann, T., 2018, "Test Generation and Test Prioritization for Simulink Models With Dynamic Behavior," IEEE Trans. Software Eng., **45**(9), pp. 919–944.
[44] Santiago, V., Do Amaral, A. S. M., Vijaykumar, N. L., Mattiello-Francisco, M. d. F., Martins, E., and Lopes, O. C., 2006, "A Practical Approach for Automated Test Case Generation Using Statecharts," 30th Annual International Computer Software and Applications Conference (COMPSAC'06), Chicago, IL, Sept. 17–21, Vol. 2, IEEE, pp. 183–188.
[45] Pradhan, S., Ray, M., and Swain, S. K., 2019, "Transition Coverage Based Test Case Generation From State Chart Diagram," J. King Saud Univ.-Comput. Inf. Sci.
[46] Verma, R., and Bhatia, R., 2012, "Behavior Based Automated Test Case Generation for Object Oriented Systems," Int. J. Comput. Appl. Technol., **54**(13), pp. 49–60.
[47] Swain, S. K., Mohapatra, D. P., and Mall, R., 2010, "Test Case Generation Based on State and Activity Models," J. Object Technol., **9**(5), pp. 1–27.
[48] Sapna, P., and Mohanty, H., 2008, "Automated Scenario Generation Based on Uml Activity Diagrams," 2008 International Conference on Information Technology, Bhubaneswar, India, Dec. 17–20, IEEE, pp. 209–214.
[49] Shanthi, A. V. K., and MohanKumar, G., 2012, "A Novel Approach for Automated Test Path Generation Using Tabu Search Algorithm," Int. J. Comput. Appl. Technol., **48**(13), pp. 28–34.
[50] Stallbaum, H., Metzger, A., and Pohl, K., 2008, "An Automated Technique for Risk-Based Test Case Generation and Prioritization," Proceedings of the 3rd International Workshop on Automation of Software Test, Leipzig, Germany, May, 11, ACM, pp. 67–70.
[51] Teixeira, F. A. D., 2018, "Easytest: An Approach for Automatic Test Cases Generation From Uml Activity Diagrams," Information Technology-New Generations, Las Vegas, NV, Apr. 3–5, Springer, pp. 411–417.
[52] Nebut, C., Fleurey, F., Le Traon, Y., and Jezequel, J.-M., 2006, "Automatic Test Generation: A Use Case Driven Approach," IEEE Trans. Software Eng., **32**(3), pp. 140–155.
[53] Sarma, M., and Mall, R., 2007, "Automatic Test Case Generation From Uml Models," 10th International Conference on Information Technology (ICIT 2007), Orissa, India, Dec. 17–20, IEEE, pp. 196–201.
[54] Raza, N., Nadeem, A., and Iqbal, M. Z. Z., 2007, "An Automated Approach to System Testing Based on Scenarios and Operations Contracts," Seventh International Conference on Quality Software (QSIC 2007), Portland, OR, Oct. 11–12, IEEE, pp. 256–261.
[55] Prasanna, M., and Chandran, K., 2009, "Automatic Test Case Generation for Uml Object Diagrams Using Genetic Algorithm," Int. J. Adv. Soft Comput. Appl., **1**(1), pp. 19–32.
[56] Aho, A. V., and Hopcroft, J. E., 1974, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, Reading, MA.
[57] Irshad, L., Ahmed, S., Demirel, O., and Tumer, I. Y., 2018, "Identification of Human Errors During Early Design Stage Functional Failure Analysis," ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Quebec City, Quebec, Canada, Aug. 26–29, American Society of Mechanical Engineers, p. V01BT02A007.
[58] Aldemir, T., 1987, "Computer-Assisted Markov Failure Modeling of Process Control Systems," IEEE Trans. Reliab., **36**(1), pp. 133–144.
[59] Siu, N., 1994, "Risk Assessment for Dynamic Systems: An Overview," Reliab. Eng. Syst. Saf., **43**(1), pp. 43–73.
[60] Cojazzi, G., 1996, "The Dylam Approach for the Dynamic Reliability Analysis of Systems," Reliab. Eng. Syst. Saf., **52**(3), pp. 279–296.
[61] Hofer, E., Kloos, M., Krzykacz-Hausmann, B., Peschke, J., and Woltereck, M., 2002, "An Approximate Epistemic Uncertainty Analysis Approach in the Presence of Epistemic and Aleatory Uncertainties," Reliab. Eng. Syst. Saf., **77**(3), pp. 229–238.
[62] Harris, D., Stanton, N. A., Marshall, A., Young, M. S., Demagalski, J., and Salmon, P., 2005, "Using Sherpa to Predict Design-Induced Error on the Flight Deck," Aerosp. Sci. Technol., **9**(6), pp. 525–532.
[63] Billings, C. E., 1991, Human-Centered Aircraft Automation: A Concept and Guidelines, NASA Ames Research Center, Technical Report No. NASA-TM-103885.
[64] Stanton, N. A., 2014, "Representing Distributed Cognition in Complex Systems: How a Submarine Returns to Periscope Depth," Ergonomics, **57**(3), pp. 403–418.
[65] Kurtoglu, T., Tumer, I. Y., and Jensen, D. C., 2010, "A Functional Failure Reasoning Methodology for Evaluation of Conceptual System Architectures," Res. Eng. Des., **21**(4), pp. 209–234.