An Automatic Failure Mode and Effect Analysis Technique for Processes Defined in the Little-JIL Process Definition Language

Danhua Wang, Jingui Pan
State Key Laboratory for Novel Software Technology,
Nanjing University
Nanjing, China

George S. Avrunin, Lori A. Clarke, Bin Chen
Department of Computer Science, University of Massachusetts, Amherst
Amherst, USA

***Abstract*—Many processes are safety critical and therefore could benefit from proactive safety analysis techniques that attempt to identify weaknesses of such processes before they are put into use. In this paper, we propose an approach that automatically derives Failure Mode and Effect Analysis (FMEA) information from processes modeled in the Little-JIL process definition language. Typically FMEA information is created manually by skilled experts, an approach that is usually considered to be time-consuming, error-prone, and tedious when applied to complex processes. Although great care must be taken in creating an accurate process definition, with our approach this definition can then be used to create FMEA representations for a wide range of potential failures. In addition, our approach provides a complementary Fault Tree Analysis (FTA), thereby supporting two of the most widely used safety analysis techniques.**

Keywords-FMEA; FTA; automatic; Little-JIL; safety analysis technique

I. INTRODUCTION

Failure Mode and Effect Analysis (FMEA) [1, 2] is a bottom-up approach to analyzing and evaluating safety problems in a system or process in an attempt to reduce the occurrence of severe hazards or their consequences. A failure mode is "the way or manner in which a product or process could fail to meet design intent or process requirements" [3]. The potential impacts of a failure are defined as the effects of the failure mode [3]. In addition to identifying failure modes and effects, identifying the causes of an identified failure mode is another benefit of FMEA analysis. When consistently applied to a whole process, FMEA essentially consists of identifying and listing all potential failure modes, assessing the effects on the overall system for each failure mode, and then identifying all potential causes which could lead to each failure mode. Table 1 presents a small example of traditional FMEA worksheet.

FMEA can be automatically applied to a process if the process is modeled in sufficient detail and in a language with precise semantics. For this project, we used the Little-JIL process definition language, which has precise semantic definitions for all its language constructs.

Typically, FMEA is done manually by skilled experts. If the process being evaluated is at all complex, then this approach is known to be time-consuming, error-prone, and tedious. Our approach automatically applies FMEA to processes defined in the Little-JIL language. We assume that a Little-JIL process definition has been developed and validated with great care. Such a process definition can be used to study and evaluate the process [4], as well as to drive simulations [5] or executions. Thus, we assume that the process definition already exists and can be further leveraged by automatically identifying potential failure modes, and then automatically generating the effects and causes for any selected single failure mode.

Our approach overcomes the traditional shortcomings of FMEA when applied to complex processes. Also, our approach provides an integrated view of FMEA and Fault Tree Analysis (FTA), two of the most widely used safety analysis techniques. Experts can then focus their attention on the provided FMEA and FTA information for a process when trying to detect potential hazards and weaknesses.

To evaluate our approach, we have applied it to several detailed process definitions, defined in Little-JIL. We have selected processes from the healthcare domain, since hazards in health care processes can jeopardize the safety of the individuals served by them and even cause death and suffering. Moreover, FMEA has been used extensively by health care organizations to analyze safety problems in their processes to improve their safety [6-8]. Blood transfusion processes have often been the subject of FMEA [9]. In this paper, we use a simple blood transfusion process as an example.

The rest of this paper is organized as follows. Section II presents related work. Section III provides a brief introduction of the Little-JIL process definition language. Section IV gives a detail description of our automatic FMEA approach, followed by an concluding section.

II. RELATED WORKS

In [10], an approach is presented to provide automated support for FMEA using model checking with the behavior tree system modeling notation. This approach enables safety analysts to work with high-level models in a notation that is close to natural language, while automating the tedious aspects

Sponsor: ①State Key Laboratory for Novel Software Technology at Nanjing University. Project Number: KFKT2009A13.
②Department of Computer Science at University of Massachusetts, Amherst.

TABLE I. TRADITIONAL FMEA WORKSHEET EXAMPLE

Failure Mode and Effect Analysis Worksheet								
Process: Simple Blood Transfusion Process								
SEV = How severe is the effect on the customer?								
OCC = How frequently is the cause likely to occur?								
DET = How probable is the detection of the cause?								
RPN = Risk priority number in order to rank concerns; calculated as SEV x OCC x DET								
Process Step	***Failure Mode***	***Effects***	***SEV***	***Causes***	***OCC***	***DET***	***RPN***	***Actions***
Obtain patient's blood type	"Blood Type" is wrong	"Blood Unit" is wrong	10	Wrong "Patient Name"	2	6	210	
				"Test patient's blood type" produces wrong "Blood Type"	5	4		
				"Contact lab for patient's blood type" produces wrong "Blood Type"	3	7		

of FMEA. Another model-based FMEA approach is proposed in [11, 12] by Papadopoulos et al., which realizes the semi-automatic synthesis of FMEAs which builds upon automatic fault tree analysis for system-level hazards. Many approaches are proposed to automate the creation of FMEA information from software [13, 14]. Another approach proposed by Robin Lutz is used to combine FMEA and FTA to analyze the requirements of safety critical software (e.g. spacecraft software) [15]. There have been several published studies demonstrating the benefits of employing FMEA in various domains. For example, Snooke N. et al. describes how model-based simulation can be employed to automatically generate the system-level effects of all possible failures on systems within the aircraft systems [16]. The application of functional modeling to the automatically produce FMEA information for mechanical systems is described in [17]. It is worth mentioning that the approach proposed in [11, 12] is not restricted to particular domains, i.e. applicable to a range of widely used engineering models.

III. LITTLE-JIL PROCESS DEFINITION LANGUAGE

Little-JIL is "an executable, high-level process programming language with a formal (yet graphical) syntax and rigorously defined operational semantics" [18]. It provides a process modeling method basing on activities, which are defined as steps in Little-JIL processes. Little-JIL processes coordinate the activities of autonomous human or computer agents and their use of resources during the performance of their activities. A Little-JIL process model for blood transfusion is shown in Fig. 1. Here, we only give a brief introduction to the semantics of Little-JIL. Details of the language can be found in [19].

A Little-JIL process definition is a hierarchy of steps, each of which represents a single unit of work. Every step specifies all artifacts and resources it uses in its interface. A step can optionally be proceeded by one or several pre-requisite step(s) or/and be followed by one or several post-requisite step(s). A step without any sub-steps is called a leaf step. Each non-leaf step has a sequencing badge which indicates the execution order of its sub-steps. Artifacts, which are objects such as a medical chart or prescription, are passed between different steps via parameter bindings. There are four parameter types, IN, OUT, IN/OUT and Locals. In Little-JIL, resources are "special kinds of artifacts for which there is contention for access" [19]. Resources are managed by an external resource manager and their acquisitions need to be explicitly specified in step interfaces. Steps may throw exceptions, which can be handled by exception handler steps. Each step in Little-JIL is assigned to an execution agent (human or automated), which is responsible for performing the work associated with a step.

In the Little-JIL process definition, shown in Fig. 1, the root of the process, "Perform in-patient blood transfusion", is a sequential step, which means its sub-steps "Obtain patient's blood type", "Pick up blood from blood bank", and "Administer blood transfusion", should be executed from left to right one by one. "Patient Name" is passed to "Obtain patient's blood type" as an artifact via a parameter binding. Since "Obtain patient's blood type" is a try step, its sub-step "Contact for patient's blood type" and "Test patient's blood type" should be tried from left to right until one of them is executed successfully and returns an artifact "Blood Type" to "Contact for patient's blood type". Then this "Blood Type" should be passed to "Pick up blood from blood bank". After "Pick up blood from blood bank" is completed, the root step "Perform in-patient blood transfusion" should receive "Blood Unit" and then pass it to "Administer blood transfusion", along with "Patient Name", another artifact. "Patient Name" is passed to "Find patient location in computer" to get "Patient Bed Location". Then "Blood Unit" and "Patient Bed Location" are finally passed to "Blood transfusion".

Each of the leaf steps in Fig. 1 could be further decomposed into sub-steps, but that elaboration is not presented here.

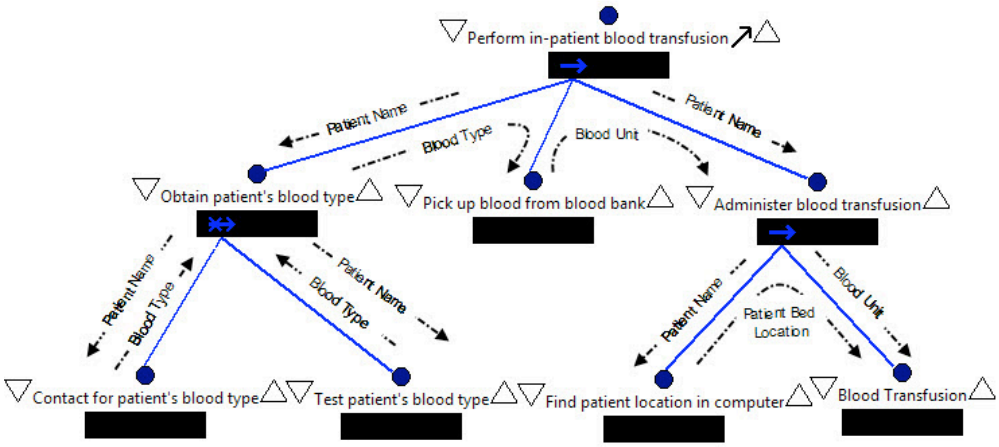


Figure 1. Simple Blood Transfusion Process

IV. AUTOMATIC FMEA FOR LITTLE-JIL PROCESSES

Our approach to automatically derive the FMEA information from Little-JIL involves three steps: identify potential failure modes, identify effects for each failure mode, and identify causes for a given failure mode. We describe each step in turn.

A. *Identify Failure Modes*

To automatically generate failure modes from Little-JIL process definitions, one needs to first identify appropriate failure modes for each language construct. In our work, we limit our attention to the failure modes related to artifacts and define two types of artifact-related failure modes:

Type 1: Artifact p to Step S is wrong.

Type 2: Artifact p from Step S is wrong.

Although not all failures can be associated with these failure modes, a large number of interesting failure modes are artifact-related failure modes or can easily be turned into artifact-related failure modes. For example, if a step is done incorrectly, we would expect that failure to be evident in one or more of the out artifacts associated with that step. When there are no such out artifacts, a hypothetical output could be created to represent the erroneous behavior of the step.

Each step, the basic elements of a Little-JIL process definition, has an interface that specifies the artifacts it uses as parameters along with each ones type. The failure modes related to artifacts for each step are created using the following rules:

- For an IN parameter p declared in the interface of Step S, the failure mode "Artifact p to Step S is wrong" (Type 1) is generated.
- For an OUT parameter p declared in the interface of Step S, the failure mode "Artifact p from Step S is wrong" (Type 2) is generated.
- For an IN/OUT parameter p declared in the interface of Step S, both the failure mode "Artifact p to Step S is wrong" (Type 1) and the failure mode "Artifact p from Step S is wrong" (Type 2) are generated.
- A local parameter is a special IN or OUT or IN/OUT parameter with a limited scope such that that associated artifact can only be passed between a step and its sub-steps. Thus, for this case, failure mode(s) can be generated the same as no-local IN, OUT, or IN/OUT parameters.

By generating potential failure mode(s) for each step in a process, all potential failure modes related to artifacts in the process definition are identified. For each of these failure modes, the potential effects need to be identified.

B. *Identify Potential Effects for Each Failure Mode*

The effects derivation algorithm consists of two phases:

1) Phase1. Construct the Artifact Flow Graph (AFG) from the unrolled Little-JIL process. The AFG can be easily constructed by traversing the process tree with an algorithm (not shown) that can be done in polynomial time.

The AFG is used to determine whether an artifact is data dependent on another artifact. An AFG is a direct graph $G_a = <P_a, E_a>$, where P_a is the set of artifacts in the process and E_a is the set of edges, which represent the data dependent relationships between artifacts. Suppose both p_1 and p_2 are artifacts, there is an edge from p_1 to p_2 if and only if p_2 is data dependent on p_1. If there is a parameter binding indicating that p_1 is passed to p_2 or if p_1 is an input parameter and p_2 is an output parameter of the same step, we create an edge from p_1 to p_2 indicating that p_2 is data dependent on p_1.

Fig. 2 gives the corresponding AFG of the simple blood transfusion process modeled in Fig. 1. Every Node in Fig. 2 indicates an artifact in the process, which is represented as "artifact name (step name)" in the graph. The AFG can be generated directly from a Little-JIL process by traversing the process tree.

2) Phase2. Derive FMEA information using the AFG:

After generating the AFG for a Little-JIL process definition, the effects for each failure mode can be identified. If there is a path from artifact p_1 to artifact p_2 in the AFG, then a fault in p_1 may be propagated to p_2. Thus, given a Type 1 failure mode ("Artifact p to Step S is wrong") or a Type 2

failure mode ("Artifact p from Step S is wrong"), it is straightforward to determine incrementally the artifacts at which steps that could be contaminated by such a faulty p by traversing the AFG. In other words, a fault in p could be propagated to these artifacts causing them to be faulty. Therefore the fault of these artifacts is defined as the effects of the given failure mode.

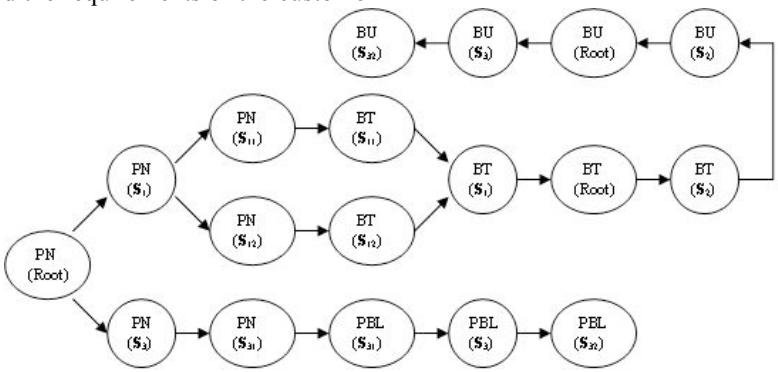
A high-level description of the algorithm for deriving the FMEA information is given here:

//Initialization:
Initialize the visited AFG nodes ***vn*** set to empty;
Initialize AFG nodes ***nd*** set to all AFG nodes;
Initialize worklist ***wl*** set to empty;
//Main Cycle:
While ***nd*** is not empty do
    Remove an AFG node ***n*** from ***nd***;
    Add ***n*** to ***wl***;
    If ***vn*** contains ***n*** then continue;
    Maps ***n*** to the set of AFG nodes vn_{to} set that directly flow to ***n***;
    Replace ***n*** with set vn_{to};
    For each node ***n'*** in set vn_{to}
        Add ***n'*** to ***wl***;
    End for
    Add ***n*** to vn_{to};
End while

Formats of FMEA information may vary based on the needs of the organization and the requirements of the customer [1]. Familiar formats are FMEA tables or FMEA worksheets. In our approach, we proposed a new representation, call an Effect Tree, which provides a way to organize the failure modes and the effects of a process into a tree view. The top level of an Effect Tree lists the step name of each process definition step. The second level lists the failure modes of each step. The third level lists the effects of each failure mode. The next and all subsequent levels list the effects resulting from the failure of their parent. This expansion continues until there is no subsequent effect that can be propagated, according to the AFG.

Fig. 3 shows one failure mode of step "Obtain patient's blood type" and its effects. It indicates that wrong "Blood Type" produced by "Obtain patient's blood type" could lead to the wrong "Blood Type" being sent to "Pick up blood from blood bank", and subsequently results in the wrong "Blood Unit" being returned and then passed to "Perform in-patient blood transfusion". Finally this could result in wrong "Blood unit" being provided to "Blood transfusion". The artifact flow paths in the AFG that represent the fault propagation paths can be tracked by expanding the tree nodes in the Effect Tree.

Inspecting all effects of each failure mode should help identify effects that could result in significant damage. For the blood transfusion example, there are two effects that deserve more attention: "Blood Unit" to "Blood Transfusion" is wrong, "Patient Bed Location" to "Blood Transfusion" is wrong. The first one indicates that the wrong blood unit is transfused to the patient, and the second indicates that the blood unit is transfused to the wrong patient. Both of these could have



Annotations: Artifacts: PN-Patient Name, BT-Blood Type, BU-Blood Unit, PBL-Patient Bed Location; Steps: Root-Perform in-patient blood transfusion, S_1-Obtain patient's blood type, S_{11}- Contact for patient's blood type, S_{12}-Test patient's blood type, S_2-Pick up blood from blood bank, S_3-Administer blood transfusion, S_{31}-Find patient location in computer, S_{32}-Blood Transfusion.

Figure 2. AFG of the Simple Blood Transfusion Process

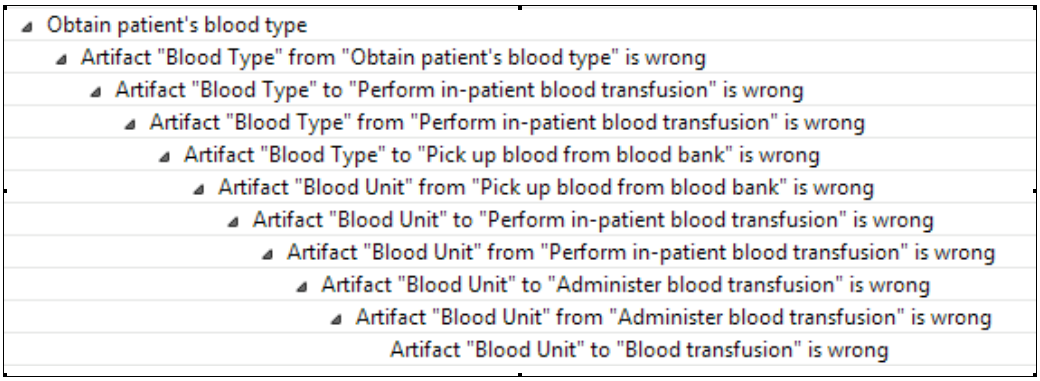


Figure 3. Part of FMEA Tree View Result for the step "Obtain patient's blood type"

serious consequences.

C. Identify Causes for a Given Failure Mode

In Fault Tree Analysis (FTA), severe consequences, such as the two listed above, are considered hazards. These identified hazards can then be treated as the TOP-events of a fault tree. Using the fault tree, FTA tries to determine which combinations of events must transpire for the hazard to actually occur. In our approach, we proactively identify possible hazards of a process using FMEA, and then use FTA to determine what events must occur for each hazard to arise.

The fault tree is a graphic model of the various parallel and sequential combinations of faults (events) that will result in the occurrence of the predefined undesired event (top event) [20]. Logical gates (e.g. AND, OR) are used to represent the interrelationships between events and the top event. The FTA approach involves two steps, deriving a fault tree and analyzing the fault tree. Deriving a fault tree starts with the top event, which is then further developed. All intermediate and necessary events that may lead to the top event are connected to the top event using appropriate gates. These new events will then be developed if they are not primary events. This procedure continues until all new events are primary events. Once a fault tree has been derived, both qualitative and quantitative analysis can be applied to analysis it. Through analyzing a fault tree, all minimal cut sets (MCSs), that is minimum combinations of events that would cause the top event to occur, can be found and their probability calculated. MCSs of a fault tree indicate whether the process is exposed to single points of failure or combinations of high-probability events. Subsequent changes may need to be made to the process to remove these weaknesses. Fig. 4 presents the generated fault tree for one of the hazards identified in the simple blood transfusion process using FMEA.

By generating the fault tree for the effects which may lead to a hazard, and doing FTA, process modifications may be recommended to try to prevent the hazard from occurring. Previous work has addressed how to automatically derive a Fault Tree from a Little-JIL process definition automatically and thus, we only give a brief introduction of the approach here. For a detailed description of it, see [21].

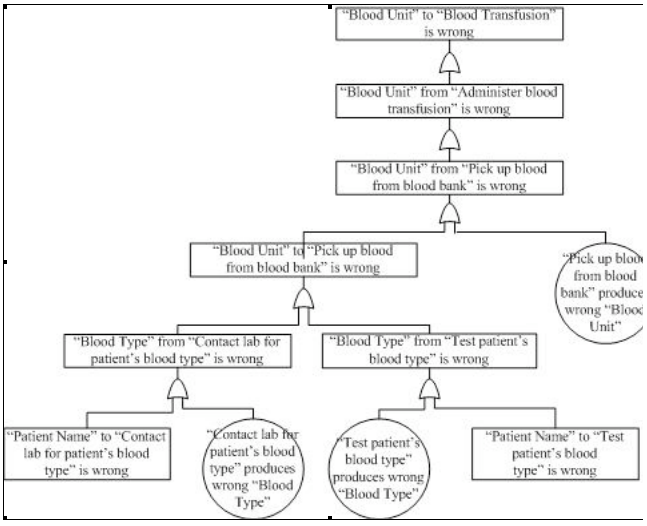


Figure 4. An example of generated fault tree

The fault tree derivation starts with the given TOP-event, which is an intermediate event representing the given hazard (in our work, it is a given failure mode or effect). Then the intermediate event will be developed – all immediate and necessary events that could lead to this event are identified and connected to this event using appropriate gates. Those new events themselves may be intermediate events and need to be developed further. This process continues until all leaf nodes in the fault tree are primary events that do not need to be developed further as determined by the analyst. To automatically derive a fault tree from a Little-JIL process definition, there are two issues that have to be addressed: how to automatically extract fault tree events that could lead to a given event and how to connect them to this event using the appropriate gates. For the first issue, a few types of events are predefined which can be easily identified from the process definition. To address the second issue, a collection of templates are defined based on the Little-JIL process definition language. Different templates are used to develop different types of events.

V. Conclusions and Future Work

FMEA is an inductive technique for analyzing and evaluating potential reliability problems in a process or system. It is well accepted and applied in various kinds of industries especially safety critical processes, such as medical safety processes [22, 23, 24]. In this paper, we present an approach that can be used to automatically generate FMEA information from a Little-JIL process. Since Little-JIL steps have simple uniform interfaces, failure modes related to artifacts can be automatically generated. Effects of single failure mode can be identified by traversing the AFG of the process. The automatic FTA approach proposed in [21] is employed to generate fault trees for hazards identified by carefully checking FMEA information.

Performing fault tree analysis for all potential failure modes and effects in a process might be a huge undertaking and sometimes a waste of time and energy. The effects which may cause hazards should be identified and then further evaluated. We suggest that after generating all potential failure modes and their potential effects for a process, the effects of each failure mode should be examined carefully. If the effect is critical, a fault tree needs to be generated to find out the possible causes. By evaluating the possible causes of such effects, actions can be recommended to avoid the hazard. With our approach, such recommended changes would be made to the process definition which would then be reevaluated to assure that the problem had been effectively addressed.

Our proposed approach focuses on the problems that can arise though artifacts. Erroneous artifacts are probably the most common way in which problems will propagate though a process. There are other ways that this could happen however, such as though the use of erroneous resources or the faulty execution of a step that nonetheless does not contaminate the output parameters of that step.

After being acquired, Little-JIL resources can be passed as parameters like other artifacts [19]. Similarly, the artifacts produced by one step might be subsequently used as resources in another step. Therefore, faulty resources and resource passing between different steps might lead to fault propagation in the process. Also, as described in Section III, agents are treated as a special kind of resource in Little-JIL, and therefore errors introduced by agents should also be taken into account. Without considering resource faults leads to incomplete FMEA result. In other words, we need to determine the dependences between parameters and resources. Thus, one part of our future work should be finding out how resource fault affects the fault propagation in Little-JIL process definitions.

Another limitation is that subsequent steps could be dependent on a previous step being done correctly, where the failure to do is not reflected in an artifact. If we create a hypothetical output artifact to represent the erroneous step behavior, this artifact would not propagate erroneous information beyond this step. Therefore, we may want to find a way to add fault propagations introduced by erroneous step behavior to generate the FMEA information. This information would be overwhelming, however, unless it is guided by the analyst or some other external information.

In summary, automatically generating FMEA information from Little-JIL processes addresses the major weaknesses of traditional FMEA approaches. This information can be examined to determine which are the events of major concern, and these events can then be used to drive FTA. Thus, our approach aims to coordinate FTA and FMEA, two complimentary approaches, to help improve processes.

REFERENCES

[1] Department of Defense, Procedures for Performing a Failure Mode, Effects and Criticality Analysis, MIL-STD-1629, Washington D.C., 1980.

[2] N.G. Leveson, Safeware: System Safety and Computers, vol. 20, Pubished by Addison-Wesley, 1995.

[3] Potential Failure Mode and Effects Analysis (FMEA), Automotive Industry Action Group (AIAG), 4th ed., 2008.

[4] B. Chen, G.S. Avrunin, and E.A. Henneman, et al., "Analyzing medical processes," International Conference on Software Engineering(ICSE), Germany, 2008, pp. 623-632.

[5] M.S. Raunak, L.J. Osterweil, A. Wise, L.A. Clarke, and P. Henneman, "Simulating patient flow through an emergency department using process-driven discrete event simulation," Software Engineering in Health Care(SEHC), Canada, 2009, pp. 73-83.

[6] E. Stalhandske, J. DeRosier, R. Wilson, and J. Murphy, "Healthcare FMEA in the veterans health administration," http://www.va.gov/ncps/SafetyTopics/HFMEA/PSQHarticle.pdf, unpublished.

[7] Failure Mode and Effects Analysis (FMEA): A framework for proactively identifying risk in healthcare, 1st ed., vol.6. Toronto ON: ISMP Canada, 2006.

[8] Example of a Health Care Failure Mode and Effects Analysis for IV Patient Controlled Analgesia (PCA)," http://www.ismp.org/Tools/FMEAofPCA.pdf, unpublished.

[9] J. Burgmeier, "Failure mode and effect analysis: an application in reducing risk in blood transfusion," Jt Comm J Qual Improv, 2002.

[10] G. Lars, L. Peter, and Y. Nisansala, and M.H. Lee, "An automated failure mode and effect analysis based on high-level design specification with behavior trees," International Conference on Integrated Formal Methods (IFM), 2005.

[11] Y. Papadopoulos, D. Parker, and C. Grante, "Automating the failure modes and effects analysis of safety critical systems," In: Int. Symposium on High-Assurance Systems Engineering (HASE), IEEE Computer Society, 2004, pp. 310‑311.

[12] Y. Papadopoulos, and M. Maruhn, "Model-based automated synthesis of fault trees from Matlab-Simulink models," Int'l Conf. on Dependable Systems and Networks(DSN'01), 2001, pp. 77-82.

[13] C. Price, and N. Snooke, "An automated software FMEA," Proceedings of the International System Safety Regional Conference(ISSRC), Singapore, 2008.

[14] H. Hecht, and R. Menes, "Software FMEA Automated and as a Design Tool,"http://www.sohar.com/proj_pub/download/Software_FMEA_Auto_As_Design_Tool.pdf, unpublished.

[15] R.R. Lutz, and R.M. Woodhouse, "Requirements analysis using forward and backward search," Annals of Software Engineering, Special Volume on Requirements Engineering, vol.3, 1999, pp. 459-475.

[16] N. Snooke, C. Price, and C. Downes, and A. Carol, "Automated failure effect analysis for PHM of UAV," Proceedings of the International System Safety and Reliability Conference (ISSRC), April 2008.

[17] N. Hughes, E.X Chou, C.J. Price, and L. Mark, "Automating mechanical FMEA using functional models," Proc 12th Int. Florida AI Research Soc. Conf(FLAIRS-99), 1999, pp. 394-398.

[18] A. Wise, A.G. Cass, B.S. Lerner, E.K. McCall, L.J. Osterweil, and S.M. J Sutton, "Using Little-JIL to coordinate agents in software engineering," Automated Software Engineering Conference(ASE), Grenoble, France, 2000, pp. 155-163.

[19] Little-JIL 1.5 Language Report, Department of Computer Science, University of Massachusetts, Amherst, 2006, unpublished.

[20] W. E. Vesely, Fault Tree Handbook, U.S. Nuclear, Regulatory Commission, 1981, pp. 34-44.

[21] B. Chen B, G.S. Avrunin, L.A. Clarke, and L.J. Osterweil, "Automatic fault tree derivation from Little-JIL process definitions," Proc. of the Int'l Software Process Workshop and Int'l Workshop on Software Process Simulation and Modeling (SPW/ProSim), 2006, pp. 150−158.

[22] B. Duwe, B.D. Fuchs, and J.H. Flaschen, "Failure mode and effects analysis application to critical care medicine," Critical Care Clinics, vol(21), pp. 21-30.

[23] M. Apkon, J. Leonard, L. Probst, L. Delizio, and R. Vitale, "Design of a safer approach to intravenous drug infusions: failure mode effects analysis," Qual Saf Health Care, vol(13), 2004, pp. 265–271.

[24] W. Adachi, A.E. Lodolce, "Use of failure mode and effects analysis in improving the safety of i.v. drug administration, Am J Health-Syst Pharm, vol(62), 2005, pp917–920.