Maintenance-Oriented Classifications of EFSM Transitions

Nada Almasri[1], Luay Tahat[1*], Mohammad Alshraideh[2]
[1] Gulf University for Science and Technology, Management Information System, Kuwait.
[2] Jordan University, Computer Science Department, Jordan.

* Corresponding author. Email: tahat.l@gust.edu.kw
Manuscript submitted September 23, 2015; accepted November 30, 2015.
doi: 10.17706/jsw.11.1.64-79

Abstract: Extended Finite State Machine modeling is a widely used technique to model state-based systems. Although EFSM models are usually mainly used to simplify the design and implementation of the systems, their use can be extended to enhance and speed up system maintenance (e.g. Error localization, performance enhancement, change management, etc.). In this paper we present a classification approach for EFSM transitions based on their criticality during maintenance. The purpose of this classification is to give the system maintenance team a tool for estimating criticality level for each transition in the EFSM model and consequently to allow them to better plan and manage the change process according the identified criticality of the transitions involved in the required change. Our classification approach is based on transitions' complexity as well as the dependencies between the transitions in the model. An empirical study shows that the classification can be used to enhance and speed up the maintenance process for a required change.

Key words: Extended finite state machine, system modeling, state-based modeling, transition classification, criticality analysis.

1. Introduction

The demand for large and complex software systems has been steadily increasing over time. The development and maintenance of these systems are difficult and costly due to the increased complexity of the systems. System models such as the Extended Finite State Machine (EFSM) are often used during the development of a software system to reduce ambiguity, misunderstanding, and misinterpretation of system specifications [1]. Additionally, they are used for test generation and prioritizing [2-6], test suite reduction [7], model checking [8].

During software maintenance of large and evolving software systems, the specification and implementation are frequently modified to fix defects, enhance or change functionality, add new functionality, or delete an existing functionality. One of the main challenges during software maintenance is to determine the severity of the consequences of applying a change to the system. Considering one component of the system at a time, it will be helpful for the maintenance team to know:

- If a change is applied on the component, will other components be affected by the change? If so, what are these components? And what percentage do they make from the total number of components in the system? Usually, the higher the parentage is, the more severe the expected consequences of the change are.

- If a change is not applied on the component under consideration, what is the likelihood that the component will still be affected by the change? The higher the probability that a component will be affected by a change applied elsewhere in the system, the more sensitive the component is considered.

Knowing these two points for each component beforehand enhances and speeds up the maintenance process since the maintenance team will be able to forecast the scope of the change and properly plan the implementation of the change.

In this context, we propose classifying components into different levels of criticality based on 1) the estimated severity of a change applied on the component and 2) the sensitivity of the component to a change applied on other components in the system. Components have the highest level of criticality when their change severely propagates to other components in the system AND when they have high probability to be affected by a change applied elsewhere in the system.

In this paper we mainly target state-based systems, and we use their existing EFSM models to perform the classification. EFSM models focus on the behavioral aspect of the system. They describe how the system reacts to different events. They consists of two types of components, states and transitions. A states is a snapshot of the system at a particular point in time. A transition is the active component which models the circumstances that lead the system to change its state. Our proposed classification approach is applied on EFSM transitions since they are the active components of the model. We perform the classification of transitions by statically analyzing the dependencies between the transitions. For each transition in the EFSM model we calculate a criticality index and then we rank the transitions accordingly.

The main contribution of our work is:

- Defining the criteria for critical EFSM transitions in the context of systems maintenance
- Defining a formula to calculate a criticality index for each EFSM transition and classifying them accordingly.

The rest of the paper is organized as follows: Section 2 provides an overview of state based modeling and model dependencies. Section 3 identifies the criteria considered for the classification and presents the proposed classification approach. In Section 4 an empirical study is performed, and the results of the study are presented. Section 5 outlines the related work, and in Section 6, conclusions and future research directions are discussed.

2. Preliminaries

2.1. State-Based Modeling

EFSM is a very popular technique for modeling state-based systems [9] such as communications systems, control systems, and embedded systems [10], [11]. As defined in previous work [12], [13], an EFSM consists of a set of states (including a start state and an exit state) and transitions between states. A transition is triggered at its originating state when an event occurs (e.g., an input is received) and an enabling condition (e.g., a Boolean expression) associated with the transition is satisfied. When the transition is triggered, a sequence of actions can be performed (which may manipulate variables and produce an output) and the system is transferred to the terminating state of the transition. The following elements are associated with a transition: an event, a condition, and a sequence of actions.

Fig. 1 shows a graphical representation of an EFSM transition. We distinguish three types of actions: an input action (read), an output action (write), and an assignment action. In our model assignment, actions have syntax of assignment statements and enabling conditions have syntax of conditional statements of C language.

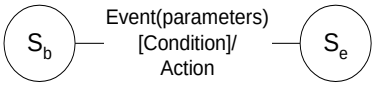


Fig. 1. EFSM transition.

An EFSM M is expressed formally as a 7 tuple: $M = (\Sigma, Q, \text{Start}, \text{Exit}, V, O, R)$ where:

Σ is the set of events,

Q is the set of states,

$Start \in Q$ is the start state,

$Exit \in Q$ is the exit state,

V is a finite set of variables,

O is the set of actions,

R is the set of transitions, where each transition T is represented by the tuple: $T = (E, C, A, S_b, S_e)$ where:

$E \in \Sigma$ is an event,

C is an enabling condition defined over V,

A is a sequence of actions, $A = <a_1, a_2, \ldots, a_j>$, where $a_i \in O$,

$S_b \in Q$ is the transition's originating state,

$S_e \in Q$ is the transition's terminating state.

In addition, the following notation related to a transition T is introduced:

$S_b(T)$ is the originating state of transition T,

$S_e(T)$ is the terminating state of transition T,

$C(T)$ is the enabling condition (a Boolean expression) associated with transition T,

$E(T)$ is the event associated with transition T,

$A(T)$ is a sequence of actions associated with transition T.

In M, Σ is a set of events, each of which is an external stimulus (input) that may be associated with a list of arguments; i.e., an event $E \in \Sigma$ is represented by $E(\text{arg}_1, \text{arg}_2, \ldots, \text{arg}_k)$. States in Q are passive elements in the EFSM model. States are just snapshots of the system and they are not involved in any kind of decision-making or computation. The states *Start* and *Exit* are where the system starts and terminates, respectively. The variables in V provide storage for values that is accessible by enabling conditions and actions in transitions. An action $a_i \in O$ is one of the following types: assignment action, *output* action, or function call. An *assignment* action assigns a value to a variable. An *output* action displays a variable or a constant to the external environment. A *function call* to some function $f(v_1, v_2, \ldots, v_k)$ returns the evaluated value.

A transition T in R is triggered when the system is in the originating state $S_b(T)$, the event $E(T)$ occurs, and the enabling condition $C(T)$ is evaluated to TRUE. When transition T is triggered, the $A(T)$ sequence of actions is performed and the system is transferred to the terminating state $S_e(T)$. If a transition T is specified at a state with no enabling condition, no other transition from that state can be associated with $E(T)$.

EFSM models may be depicted as graphs where states are represented by nodes and transitions are represented by directed edges between states. A simple EFSM model of an ATM system is shown in Fig. 2. This ATM system supports three types of transactions: withdrawal, deposit and balance. Before ATM transactions can be performed, user must enter a valid pin that is matched against the pin stored in an ATM card. A user is allowed a maximum of three attempts to enter the valid pin. For example, transition T2 is triggered when: 1) the model is in state S1, 2) event PIN(p) is received, 3) the value of parameter p does not equal to variable pin, and 4) the value of variable attempts is less than three. When the transition is triggered: 1) an error message is displayed, 2) the value of variable attempts is incremented, and 3) the user is prompted to enter PIN. Notice that in this example, for transition $T2$, $Sb(T2) = S1$, $Se(T2) = S1$, $C(T2)$

= (p != pin) and (attempts < 3), $E(T2)$ = PIN(p).

In this paper, we assume that the EFSM model is executable, i.e., enough details are provided in the model so that the model executor can execute the model based on the model specification (or an executable program corresponding to the model can be generated from the model specification). In order to support model execution, some actions may not be implemented (they are represented by "empty" actions). However, all actions are implemented during the development of the system. An input to the EFSM is a sequence of events with values for arguments associated with the events. For example, consider the following input for the EFSM of the ATM system of Fig. 2:

t = Card(1234, 200), PIN(1234), Deposit(20), Continue(), Withdrawal(50), Continue(), Exit().

When the model of Fig. 2 is executed on the sequence of events t above, the following sequence of transitions is executed: (t) = <$T1$, $T4$, $T6$, $T7$, $T11$, $T7$, $T8$>.

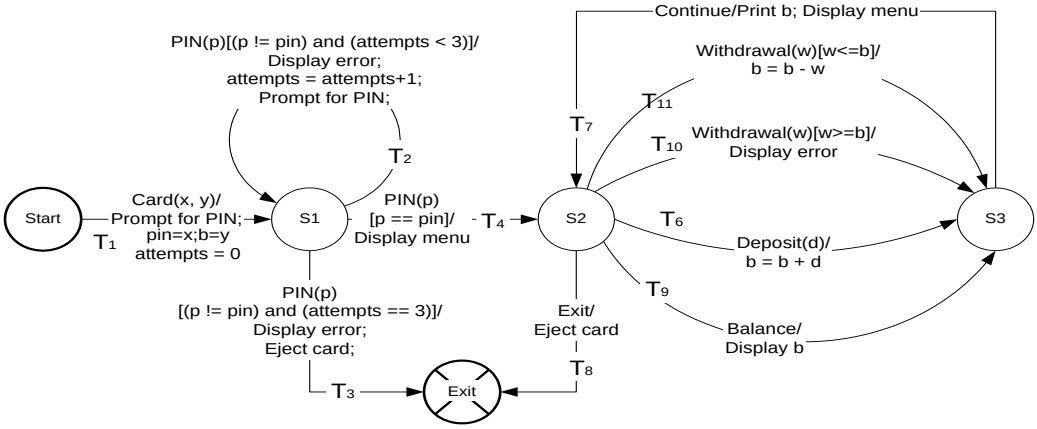


Fig. 2. EFSM model for ATM.

In this paper, we assume that the EFSM model is deterministic, i.e., for every event $E_i(x_i)$ where x_i = arg_1, arg_2, …, arg_k, in t there is one and only one possible execution of model M (at most one transition is executed for a given event $E_i(x_i)$). When model M is executed for a given sequence of events t = <$E_1(x_1)$, $E_2(x_2)$, …, $E_n(x_n)$>, a sequence of transitions (t) = <T_{i1}, T_{i2}, …, T_{im}> is executed.

2.2. Model-Dependence

The concept of data and control dependencies is well known at the code level and expanded on the model level [14]. Model dependencies are used in different research work for different goals: slicing [15], generating test cases, and prioritizing test cases for regression testing. In this paper, we use model dependencies to measure how critical applying a change to the model can be.

We identify two types of dependencies between transitions: data dependency and control dependency [16]-[18].

Transition dependence analysis with respect to data dependence focuses on occurrences of variables within the system model. Each variable occurrence is classified as being a variable definition or a variable use. We refer to these as definition and use, respectively. A definition of a variable v in a transition is any occurrence of v at which v is assigned a value. A transition can define a variable v by defining v as a part of the action(s) (e.g., $v = x + 5$). A use of a variable v in a transition is any occurrence of v that references the value of v. A transition can reference a variable v in a Boolean expression associated with the transition (e.g., [v < 0]) or by using v in action(s) associated with the transition (e.g., $x = v + 5$).

Let T be a transition. The following concepts related to transition T are introduced:

$D(T)$ is a set of variables defined by transition T, i.e., variables defined by an action(s) or by a triggering event of T.

- U(T) is a set of variables used in transition T, i.e., variables used in a condition and an action(s) of T.

For example, in the EFSM model of Fig. 2, for transition $T1$, D($T1$) = {pin, b, attempts} and $U(T1)$ = {x, y}.

Data dependence captures the notion that one transition defines a value of a variable and another transition may potentially use this value. There exists a *data dependence* between transitions T_i and T_k if transition T_i modifies the value of variable v, transition T_k uses v, and there exists a path (transition sequence) in the model from T_i to T_k along which v is not modified. More formally, there exists *data dependence* between transitions T_i and T_k if there exists a variable v such that: 1) $v \in D(T_i)$, 2) $v \in U(T_k)$, and 3) there exists a path (transition sequence) in the EFSM model from T_i to T_k along which v is not modified; such a path is referred to as a *definition-clear path.* For example, there exists a data dependence between transitions T_1 and T_7 in the model of Fig. 2. This is because transition T_1 assigns a value to variable b in the action "$b = y$", transition T_7 uses variable b in the action *"Print b"*, and there exists a path from T_1 to T_7 along which b is not modified (sequence of transitions T_1, T_4, T_{10}, T_7).

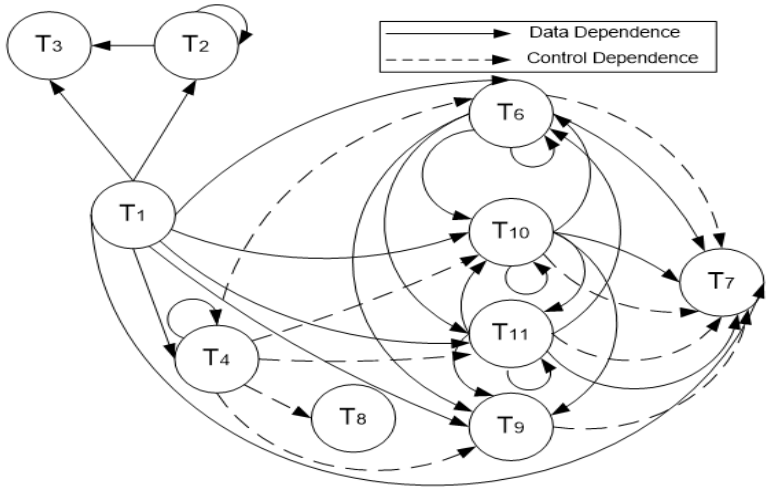


Fig. 3. Model dependence graph of the ATM model.

Control dependence was originally defined for a program's Control Flow Graph (CFG) [19]. Control dependence captures the notion that one node in the control graph may affect the execution of another node. Control dependence in an EFSM exists between transitions and it captures the notion that one transition may affect traversal of another transition. Control dependence between transitions is defined similarly to control dependence between nodes of a CFG i.e., in terms of the concept of post-dominance. Let Y and Z be two states (nodes) and T be an outgoing transition (edge) from Y. State Z *post-dominates* state Y if Z is on every path from Y to the exit state of the EFSM. State Z post-dominates transition T if Z is on every path from Y to the exit state of the EFSM through transition T. Transition T_k is control dependent on transition T_i iff: (1) $S_b(T_k)$ does not post-dominate $S_b(T_i)$ and (2) $S_b(T_k)$ post-dominates transition T_i. Notice that the definition of control dependence presented in this paper captures the same view as the definition of control dependence between nodes in a CFG.

For example, transition T_4 has control dependence on transition T_9 in the model of Fig. 2 because state S_2 does not post-dominate state S_1 (condition 1 of control dependence definition is true) and state S_2 post-dominates transition T_4 (condition 2 is TRUE).

Data and control dependence can be graphically represented by a directed graph where nodes represent model transitions and directed edges represent model data and control dependencies.

More formally, let M = (Σ, *Q, Start, Exit, V, O, R*) be an EFSM model and let G=(R, E) be a model dependence graph of model M *w*here:

R is a set of nodes (set of transitions)

E is a binary relation on R, $E \subseteq R \times R$, referred to a set of directed edges where: edge $(T_i, T_k) \in E$, if there exists data or control dependence between transitions T_i and T_k.

Throughout this paper, we will be consistent on using the statement "there exists dependence between transitions T_i and T_k". It will always mean that T_k depends on T_i (not the opposite), so in the dependence graph, the relation will be represented by the directed edge (T_i, T_k).

Due to space limitation, Fig. 3 shows only a partial model dependence graph of the model of Fig. 2. Variables associated with data dependencies are also not shown. Note that data dependencies are shown as solid edges and control dependencies are shown as dashed edges.

3. Identifying Critical Transitions

Software criticality is usually considered from the safety point of view. In this paper, however, our focus is on system maintenance. In this context, we define critical transitions as influential and/or important transitions which require greater attention during the maintenance process. During system maintenance for state based systems, a change can be applied to one or more transition/s. For a given transition Ti, when a change is applied on the model, the change is either (1) applied to Ti or (2) it is applied to other transitions in the model. Consequently, we identify two criticality measures for each transition.

- The first measure considers the criticality of a transition with respect to a change applied to the transition itself. When the change is applied to Ti, if the change propagates to a large number of other transitions in the model, we consider that Ti is a critical transition as its change affects a large portion of the model. *We call this measure: change-severity of Ti.*
- The second measure considers the criticality of a transition with respect to a change applied to other transitions in the model. When the change is applied on a transition other than Ti, we consider that Ti is critical if it has a high probability to be affected by the change. We call this measure: *Ti's sensitivity to change*
- The overall criticality of a transition can be calculated using both measures and taking into consideration the probability that the transition will undergo a change.

3.1. Calculating Severity and Sensitivity Measures

To measure the criticality of a transition Ti when a change is applied to it, we identify the transitions that are control or data dependent on T_i. Recursively, for each identified transition, we identify their dependent transitions. We call the set of all identified transitions *Affected* transitions (with respect to T_i) because a change performed on T_i propagates to these transitions either directly or indirectly, and consequently they are affected by the change. The larger the set of affected transitions is, the more critical T_i is.

Similarly to measure the criticality of a transition Ti when a change is applied to other transitions in the model, we identify all transitions on which T_i is recursively either control dependent or data dependent. We call these transitions *Affecting* transitions (with respect to T_i) because if a change is performed on any of these transitions, the change propagates to, and potentially affects, T_i. The larger the set of affecting transitions is, the more critical the transition is. To formally define the set of affecting transitions and the set of affected transitions, we define the relationship *"affects"* as follows:

Let G=(R, E) be the dependence *graph of the model M. A transition T in R "affects" another transition T′ in R if and only if there is a non-null path from T to T′ in G.*

Since the dependence relationship itself is not transitive, then we can look at the "affects" relationship as the transitive relation of the dependence relation between transtions. For example, if transition T depends on transition Q, and transition Q depends on transion S, then S "affects" T.

Definition **1:** *Let G=(R, E) be the dependence graph of the model M. The set of affected transitions for a*

transition T in G is the set of all transitions T′, where T "affects" T′. Formally, we define this set as:

$$AD(T) = R', \text{ where } R' \subseteq R, \text{and } T' \in R' \text{if and only if } \text{"}T \text{ affects } T'\text{"} \text{ on } R. \quad (1)$$

Definition **2:** *Let G=(R, E) be the dependence graph of the model M. The set of affecting transitions for a transition T in G is the set of all transitions T′ that "affects" the transition T. Formally, we define this set as:*

$$AG(T) = R', \text{ where } R' \subseteq R, \text{and } T' \in R' \text{if and only if } \text{"}T' \text{ affects } T\text{"} \text{ on } R. \quad (2)$$

After identifying the set of affecting transitions and the set of affected transitions for each transition T in the model, transitions are classified according to two different classifications: one classification is based on the severity of a change applied to a transition *Ti*, the second classification is based on the sensitivity of a transition Ti to a change applied on other transitions in the model. Accordingly, for each transition two criticality indices are considered; severity index and sensitivity index.

The severity index for a transition T_i (denoted as Sv_{Ti}) gives an indication of the severity of applying a change to *Ti*. It is calculated as the size of the set of affected transitions calculated as a percentage of the total size of the model; where the size of the model is calculated in terms of the number of transitions in the model. The size of the affected transitions is denoted as $|AD(Ti)|$ and the size of the model is denoted as |M|.

The sensitivity index for a transition *Ti* (denoted as Sn_{Ti}) gives an indication of *Ti*'s sensitivity to a change applied to other transitions in the model. It is calculated as the size of the set of affecting transitions calculated as a percentage of the total size of the model.

The following formulas are used to calculate the sensitivity and the severity of a given transition *Ti*.

$$Sv_{Ti} = \frac{|AD(Ti)|}{|M|} \quad (3)$$

$$Sn_{Ti} = \frac{|AG(Ti)|}{|M|} \quad (4)$$

Table 1. Criteria for Severity and Sensitivity Criticality Levels

Criticality Level	*Classification Criteria based on Severity*	*Classification Criteria based on Sensitivity*
Very High	$Sv_{Ti} \geq 60\%$	$Sn_{Ti} \geq 60\%$
High	$40\% \leq Sv_{Ti} < 60\%$	$40\% \leq Sn_{Ti} < 60\%$
Medium	$20\% \leq Sv_{Ti} < 60\%$	$20\% \leq Sn_{Ti} < 60\%$
Low	$0 < Sv_{Ti} \leq 20\%$	$0 < Sn_{Ti} \leq 20\%$
Negligible	$Sv_{Ti} = 0$	$Sn_{Ti} = 0$

Once the severity and the sensitivity measures are calculated for each transition in the model, the transitions can be ranked into different levels of criticality. Table 1 declares boundaries for five levels of criticality. The ranges are only sample suggestion. For different systems, the maintenance team may setup different ranges for the criticality levels based on the nature of the system. The suggested criticality levels are: very high, high, medium, low, and negligible. According to Table 1, when the severity of applying a change to *Ti* is greater than 60%, it means that the change at Ti can propagate to more than 60% of the other transitions in the model. We classify transitions with a severity of 60% or more in the highest criticality level based on severity. On the other hand, the criticality of a transition *Ti* is ranked as highly critical within the sensitivity classification if *Ti* is affected by any change performed on any transition out of more than 60% of the transitions in the model. Table1 declares the boundaries of each level within each

criticality category.

3.2. Calculating Overall Criticality Index

Finally, we calculate the overall criticality for each transition Ti taking into consideration the probability that Ti will be changed or affected by a change. For a given transition Ti, when a change is applied on the model, there are three possible scenarios:

1) The change is applied directly to *Ti*
2) The change is applied to a transition that *Ti* depends on, and consequently *Ti* can be affected by the change
3) *Ti* is neither directly changed nor indirectly affected by the change

For the first scenario, to calculate the probability that a change will be applied to a given transition T on the model, we look at the transition's complexity. In EFSM, as explained in Section 2, transition T is represented by the tuple: $T = (E, C, A, S_b, S_e)$ where: E is an event, C is an enabling condition, A is a sequence of actions, $A = <a_1, a_2, \ldots, a_j>$, S_b is the transition's originating state, and S_e is the transition's terminating state. The event, originating state, and terminating states are the mandatory components of a transition, while the condition and actions are optional. A basic transition has only the mandatory components, while a more complex transition include an enabling condition and a set of actions. During maintenance, a change can be applied to any of the transition's components. Complex transitions have higher probability to undergo a change compared to basic ones because they have more components that can be subject to change. For a model M consisting of N basic transitions, we assume that they all have equal probability to change, and consequently the probability that a change will be applied to a particular transition T is 1/N. A realistic EFSM model, however, contains transitions of different complexity, and therefore the probability that a transition change should be calculated as a function of the transition's complexity.

To measure transitions complexity, we follow a simple approach of counting the constituent parts of each component of the transition. For the events, the number of parameters is counted. For conditions and actions, the variables, values, and operators are counted. We assume that a basic transition consists of a simple event without parameters, and it doesn't have any conditions or actions. A basic transition is assigned a complexity value of 1. For more complex transitions, the complexity of transition is calculated as the sum of the event's complexity, the condition complexity, and the actions complexity.

Formally, we write the complexity formula for a given transition Ti as follows:

$$Cx(T_i) = Cx(E) + Cx(C) + \sum Cx(a_j) \tag{5}$$

where:

$Cx(E) = 1 + number\ of\ paramters^*$

$Cx(C) = number\ of\ variables, values, and\ operators$

$Cx(a) = number\ of\ variables, values, and\ operators, \quad for\ assignment\ actions$

$Cx(a) = 1, \ for\ output\ actions$

$Cx(a) = 1 + number\ of$ parameters, $for\ funtion\ calls^{**}$

* The 1 represents the complexity of a transition which have a simple event without any parameters

** The 1 represent the complexity of a function call without any parameters

The probability that a given transition *Ti* will change based on its complexity compared to the complexity of the other transitions in the model, where the model consists of *n* transitions, is formulated as follows:

$$Pc(Ti) = \frac{Cx(Ti)}{\sum_{i=1}^{n} Cx(Ti)} \tag{6}$$

For the second scenario listed above, the probability that a given transition Ti can be affected by a change applied to a transition on which *Ti* has control or data dependence is calculated as follows:

$$Pc(AG(Ti)) = \frac{\sum_{\forall T_j \in AD(T_i)} Cx(T_j)}{\sum_{i=1}^{n} Cx(T_i)} \quad (7)$$

Whether a transition changes or is affected by a change, once a change reaches Ti, it can propagate to other transitions in AD(Ti) as measured by the severity index Sv_i. Consequently, the overall criticality of a transition Ti is the severity of the transition Ti taking into consideration the probability that the transition is either changed or affected by a change. The overall criticality is formulated as follows:

$$Cr(Ti) = Pc(Ti) + Pc\big(AG(Ti)\big) * (Sv_{T_i} + \frac{1}{n}) \quad (8)$$

The fraction $(1/n)$ in the formula is added to the severity because the severity index considers the transitions affected by Ti excluding Ti itself, so adding the fraction will include Ti itself. This will also avoid getting an overall criticality of zero when the severity index is zero (for transitions that do not affect any other transitions in the model).

The possible criticality values generated from this formula is a positive number less than or equal to one. The minimum possible value diverges to zero for basic transitions in very large models and which don't have any dependencies with any other transition in the model. The maximum value of 1 is very rare as it is obtained when a transition Ti is always either changed or affected by a change (so Pc(Ti) + Pc(AG(Ti) is 1), and all other transitions in the model have dependency on it (so Sv_{Ti} is 1). In fact, any value more than 0.5 is very rare as it requires a transition which have both the sensitivity and severity values higher than 70%.

After calculating the overall criticality for all transitions in the model, they are classified into five different criticality levels: very high, high, medium, Low, and very low according to the criteria presented in the table below.

Table 2. Criteria for Overall Criticality Levels

Level	*Criteria*
Very high	$Cr(Ti) \geq 0.36$
High	$0.18 \leq Cr(Ti) < 0.36$
Medium	$0.06 \leq Cr(Ti) < 0.18$
Low	$0.03 \leq Cr(Ti) < 0.06$
Very low	$Cr(Ti) < 0.03$

4. Empirical Study and Evaluation

4.1. Methodology

To evaluate the effectiveness of the presented classification approach, we apply it to six different EFSM models. We show and discuss in details the results for one of the models, and we present and discuss the summary results for all of the six models. The main purpose of applying the classification approach on six models, is to validate that the approach can be useful during system maintenance. A high concentration of transitions in one or two levels of criticality is an indication that all transitions of the models are relatively homogeneous with respect to transitions' severity, sensitivity and complexity, and in this case the classification is not necessary. On the other hand, classifying transitions proportionally into three or more transition is an indication that the classification can be useful during maintenance to distinguish between highly critical transitions and transitions with low criticality.

Table 3. Details of Six EFSM Models

Model	Model Description		
	Number of variables	Number of states	Number of transitions
Fuel Pumps	10	9	15
Cruise Control	18	5	20
ATM	8	8	19
TCP-Dialer	31	17	46
ISDN	4	20	92
Print Token	5	11	98

The six EFSM models used for this study are publicly available, and they are: an ATM (Automatic Teller Machine) model, a cruise control model, a fuel pump model, the Transfer Control Protocol-communication dialer (TCP), Print-Token, and the Integrated Service Digital Network (ISDN) protocol. The sizes of the models range from 5 to 20 states and 20 to 89 transitions. More details about the models are shown in table 3 above.

As shown in Fig. 4, the fuel pump model provides a basic specification for pay-at the pump fuel dispensing applications. The system supports two types of payment methods: Cash or Credit. After selecting the type of payment, the system supports two choices Gas filling: Regular Gas or Super Gas. The model consists of 8 state and 15 transitions. The classification approach was applied to the model, and the detailed results are shown in the Table 4 and Fig. 5.

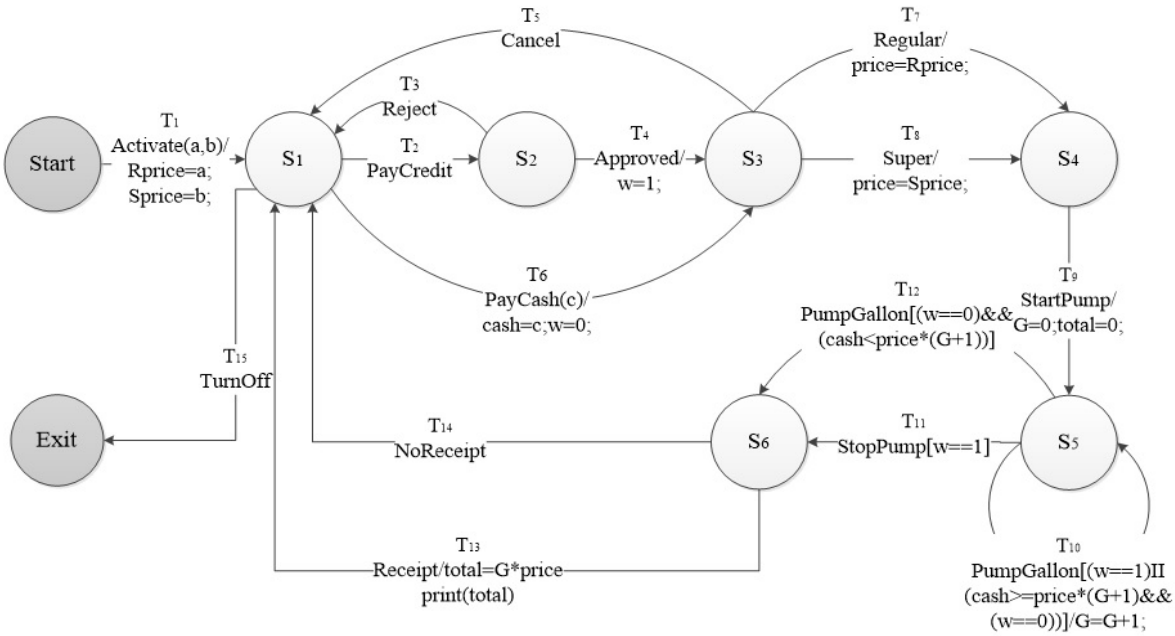


Fig. 4. Fuel pump model.

As shown in Fig. 4, the fuel pump model provides a basic specification for pay-at the pump fuel dispensing applications. The system supports two types of payment methods: Cash or Credit. After selecting the type of payment, the system supports two choices GAS filling: Regular Gas or Super Gas. The model consists of 8 state and 15 transitions. The classification approach was applied to the model, and the detailed results are shown in the Table 4 and Fig. 5.

Table 4. Fuel Pump Model Criticality Values

T	AG(T)	AD(T)	Sv	Sn	Cx(T)	Pc(Ti)	Pc(AG(T))	Cr(T)
1	{}	{7..14}	0.533	0	7	0.092	0	0.0552
2	{}	{ 3..5,7..14}	0.733	0	1	0.013	0	0.0104
3	{2}	{}	0	0.067	1	0.013	0.013	0.0017
4	{2}	{5, 7..14 }	0.6	0.067	3	0.039	0.013	0.0348
5	{2,4,6}	{}	0	0.2	1	0.013	0.132	0.0096
6	{}	{5, 7..14 }	0.6	0	6	0.079	0	0.0527
7	{1,2,4,6}	{9..14}	0.4	0.267	3	0.039	0.224	0.1226
8	{1,2,4,6}	{9..14}	0.4	0.267	3	0.039	0.224	0.1226
9	{1,2,4,6,7,8}	{10,12,13}	0.2	0.4	5	0.066	0.303	0.0983
10	{1,2,4,6,7,8,9,10}	{10,12,13}	0.2	0.533	21	0.276	0.645	0.2455
11	{1,2,4,6,7,8}	{}	0	0.4	4	0.053	0.303	0.0237
12	{1,2,4,6,7,8,9,10}	{}	0	0.533	12	0.158	0.645	0.0535
13	{1,2,4,6,7,8,9,10}	{}	0	0.533	7	0.092	0.645	0.0491
14	{1,2,4,6,7,8}	{}	0	0.4	1	0.013	0.303	0.0552
15	{}	{}	0	0	1	0.013	0	0.0104
				Total	76	1		

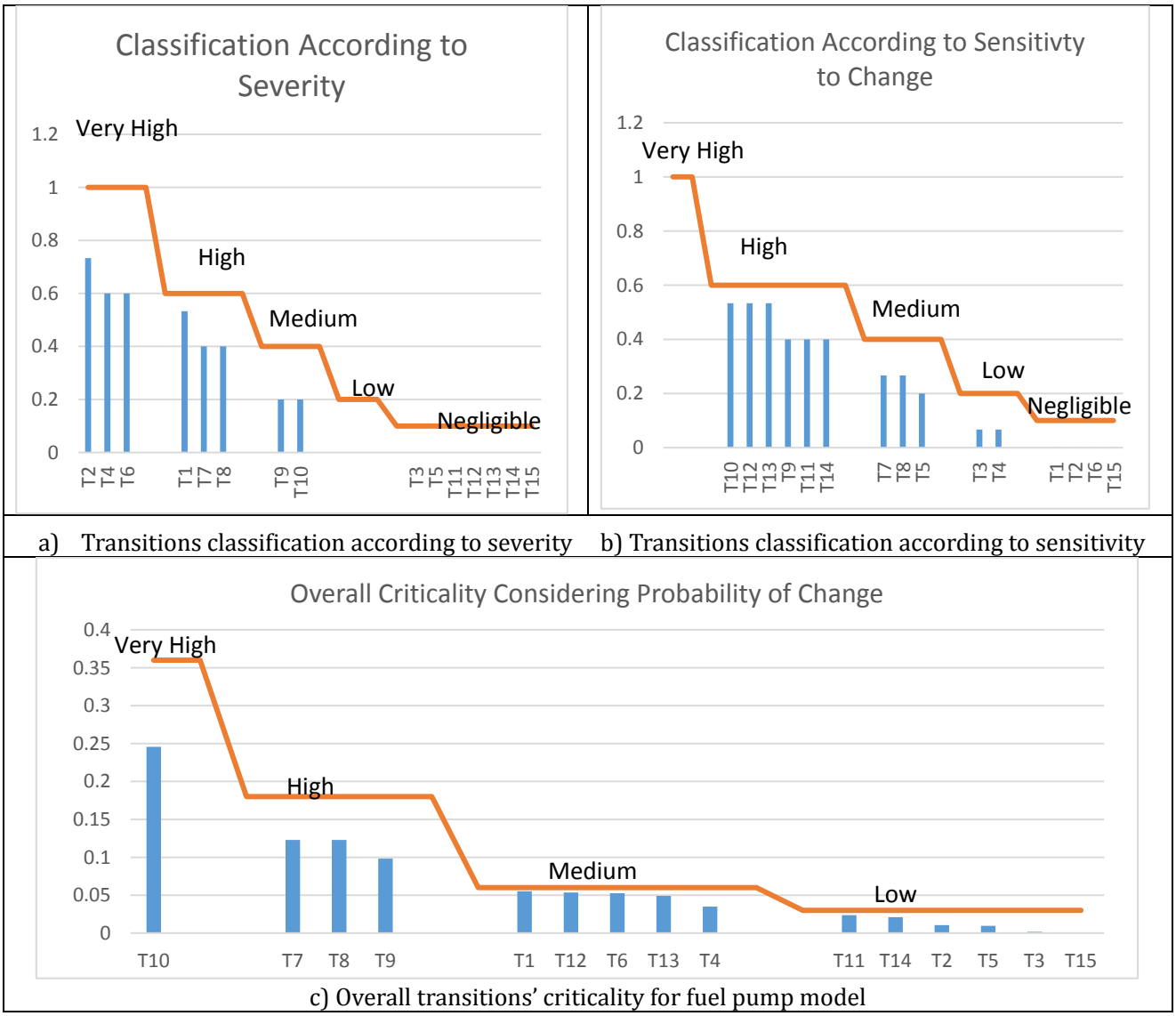
According to the severity classification, *T*2, *T*4, and T6 are classified with very high criticality. Although this is not easily visible from the EFSM model, a change performed on *T*2 can propagate to 73% of the model given the recursive dependencies of other transitions on T2. On the other hand, *T*3, 5, *T*11, *T* 12, *T*13, *T*14, *T*15 are ranked with negligible criticality since no other transitions depend on them. T1, T7, and T8 are ranked with high criticality while *T*9 and *T*10 are ranked with medium criticality.

According to the sensitivity classification, transitions *T*10, *T*12, and *T*13, followed by *T*9, *T*11, and *T*14 are all ranked with high criticality. This is mainly because T10, T12, and T13 use a large number of variables which are defined by other transitions, consequently they are highly sensitive to any change done to these variables by the other transitions. As for *T*9, *T*11, and *T*14, their execution depends on the successful execution of a large number of transitions in the model. Transitions *T*2 and *T*10 are the most interesting transitions to discuss for this model.

Transition *T*2 has the highest severity index of 73.3%. This is mainly because *T*3 and *T*4 both have control dependency on *T*2. Recursively, *T*5, *T*7, and *T*8 have control dependency on *T*4, and *T*9, *T*10, *T*11, *T*12, *T*13, and *T*14 have dependencies on *T*7 and *T*8. So, recursively, all these transitions can be affected by a change applied to *T*2. This large number of transitions potentially affected by *T*2 is the reason behind ranking *T*2 as highly critical with respect to its change severity. On, the other hand, *T*2 doesn't have any dependencies on any transition in the model. This means that a change applied to a transition other than T2 will not propagate and affect *T*2, and consequently *T*2 is ranked with negligible sensitivity. The overall criticality of *T*2 depends on the probability that *T*2 will be the subject of a change during maintenance. The calculated probability is mainly based on the transition's complexity, and since *T*2 is a basic transition with a complexity of 1, it has a very low probability to change (1/76 = 1.3%). With this low probability and despite of its high severity, *T*2's overall criticality is 0.0104 which is considered very low.

As for transition *T*10, it has the highest complexity in the model which highly increases the probability that it will undergo a change during maintenance. Combined with its medium severity and high sensitivity, *T*10's overall criticality is the highest in the model, and consequently it is ranked as very highly critical.

Finally, while the reasons for the classifications of some transitions is very obvious to a developer (e.g. the classification of *T*15, *T*3, and *T*5 with very low criticality), the classification of other transitions reveals invisible critical aspects of the transition (e.g. *T*10 is unexpectedly ranked as the most critical transition in the model).



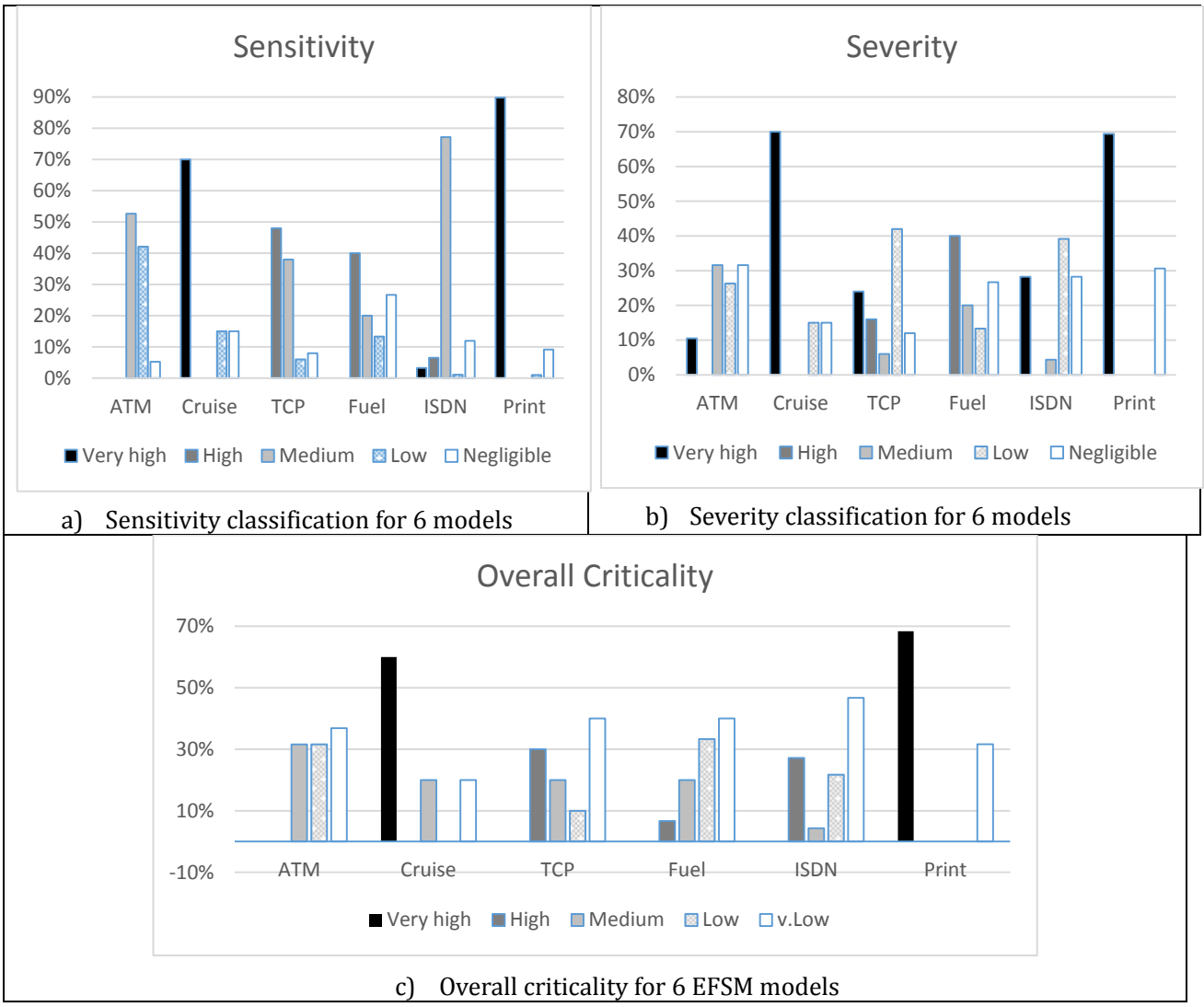
a) Transitions classification according to severity b) Transitions classification according to sensitivity

c) Overall transitions' criticality for fuel pump model

Fig. 2. (a) Transitions classifications according to severtiy (b) Transitions classifications according to sensitivity (c) Overall transitions criticality classification for fuel pump model.

4.2. Results for All Six EFSM Models

The classification approach is applied on all six EFSM models, and the summary of the classification results are presented in the Fig. 3 below. The main purpose of applying the classification approach on these six models, is to validate that the approach can be useful during system maintenance. The results show that out of the six models, two models will not greatly benefit from classifying their transitions. These two models are the cruise control and print-token models. These two models don't benefit from the classification approach since all their transitions are classified within only two or three criticality levels out of 5, and more than 60% of their transitions are classified as very highly critical. With the very high transition-per-state ratio and with the cyclic nature of these two models, it is natural to get this result. On the other hand, for the remaining four models (which have a more regular distribution of transitions over states), the transitions are distributed over three to five levels. For these models the classification can be very helpful as it clearly draws the maintainer's attention to a relatively small set of highly critical transitions which need careful planning during maintenance.



a) Sensitivity classification for 6 models

b) Severity classification for 6 models

c) Overall criticality for 6 EFSM models

Fig. 3. (a) Sensitivity classification for 6 models (b) Severity classification for 6 models (c) Overall criticality for 6 EFSM models

4.3. Evaluation

The presented empirical study shows that our approach for transition classification can be helpful when applied to certain type of models such as protocol modeling similar to ISDN and TCP. For example, when a system requirement should be changed, the system maintainer can use the severity classification to check the expected severity of the change based on severity of the transitions involved with the change. On the other hand, when an erroneous behavior is reported, and the system should be checked to identify the source of the problem, the system maintainer can use the sensitivity classification to identify the transitions affecting the erroneous transition. Additionally, the sensitivity classification can be used during regression testing after updating the system. Transitions classified with high criticality are the ones that have the highest probability to reveal erroneous behavior, so the system maintainer can start validating the system by running test cases with the highest number of sensitive transitions. Finally, the overall criticality classification can be useful when a performance issue should be addressed in the system. In this case it is helpful to look at complex transitions classified as highly critical first.

5. Related Work

Criticality analysis within the "failure mode, effects and criticality analysis" [20] is a well-known analytical technique in certain fields of engineering such as automotive and aviation. It is applied during the

design and production of new products to address potential risks and hazards, and to assess their criticality. FEMCA is mostly used in the context of safety, and it is based on brain storming the potential failures for a product, the consequences of these failures and their criticality especially in the context of safety. In software maintenance context, impact analysis is usually applied to assess the potential impact of a required change. There is a large number of code based impact analysis techniques and tools [21], as well as other research work exploring model based impact analysis techniques [22], [23]. While impact analysis techniques are applied to assess the impact of a well specified change on the system, our approach to criticality classification doesn't consider a specific change, instead it attempts to predict the criticality of the EFSM transitions for any change expected in the future.

6. Conclusion

In this paper we presented a classification approach for transitions; the active components of EFSM models. Two different classifications were presented, the first classification is based on the transition's sensitivity to change, and the second one is based on the severity of a change applied to the transition. An overall classification taking into account the transition's complexity in addition to its severity and sensitivity is also presented. The presented empirical study shows that the classification approach can be very helpful during maintenance, and it can be useful to: estimate the severity of a requested change in order to plan the development project, locate the source of an erroneous behavior, prioritize test cases during regression testing, and enhance a performance issue with the model. Additionally, the empirical study shows that for some highly cyclical models with high and disproportional transition-to-state ratio don't benefit from the classifications as the transitions are classified within only two to three levels of criticality and most of them fall within the highest level. The correlation between the design of an EFSM model and the criticality of its transition needs further investigation in a future study.

Acknowledgement

This work is supported by Kuwait Foundation for Advancement of Science (KFAS) number P114-18EO-03

References

[1] Korel, B., & Tahat, L. (2004). Understanding modification in state-based system. *Proceedings of the 12th IEEE International Conference on Program Comprehension* (pp. 246-250).

[2] Dalal, S., Jain, A., Karunanithi, N., Leaton, J., Lott, C., Patton, G., & Horowitz, B. (1999). Model-based testing and practice. *Proceeding of the Intern. Conference on Software Engineering* (pp. 185–194).

[3] Dick, J., & Faivre, A. (1992). Automating the generation and sequencing of test case from model-based specification. *Proceedings of the 5th International Symposium on Formal Methods Industrial Strength Formal Methods* (pp. 268 – 284).

[4] Cheng, K., & Krishnakumar, A. (1993). Automatic functional test generation using the extended finite state machine model. *Proceedings of the 30th ACM/IEEE Design Automation Conference* (pp. 86–91).

[5] Zhang, N., Gang, C., & Hongwei, L. (2013). Software reliability analysis using queuing-based model with testing effort journal of software, *10(4)*, 1301-1307.

[6] Vaysburg, B., Tahat, L., & Korel, B. (2001). Automating test case generation from SDL specifications. *Proceedings of the 18th International Conference on Testing Computer Software* (pp. 130–139)

[7] Cheng, K., & Krishnakumar, A. (1993). Automatic functional test generation using the extended finite state machine model. *Proceedings of the 30th ACM/IEEE Design Automation Conference* (pp. 86–91).

[8] Rongsheng, D., Zhao, W., Xiangyu, L., & Fang, L. (2013). Testing conformance of BPEL business process based on model checking. *Journal of Software*, *10(4)*, 1030-1037.

[9] Zhang, N., Gang, C., Hongwei, L. Software Reliability Analysis using Queuing-based Model with Testing Effort (2013). *Journal of Software*, *10(4)*, 1301-1307.

[10] Vaysburg, B., Tahat, L., & Korel, B. (2001). Automating test case generation from SDL specifications. *Proceedings of the 18th International Conference on Testing Computer Software* (pp. 130–139).

[11] Harel, D., & Eran G. (1996). Executable object modeling with statecharts. *Proceedings of the 18th International Conference on Software Engineering* (pp. 246-247).

[12] Wagner, F. (1992). VFSM executable specification. *Proceedings of the CompEuro* (pp. 226-231).

[13] Korel, B., Singh, I., Tahat, L., & Vaysburg B. (2003). Slicing of state-based models. *Proceedings of the International. Conference on Software Maintenance* (pp. 34-43).

[14] Korel, B., Tahat, L., & Vaysburg, B. (2002). Model based regression test reduction using dependence analysis. *Proceedings of the IEEE Conference on Software Maintenance* (pp. 214-223).

[15] Androutsopoulos K., *et al.* (2009). Control dependence for extended finite state machines. *Proceedings of the Fundamental Approaches to Soft Engineering* (pp. 216-230).

[16] Vaysburg, B., Tahat, L., & Korel, B. (2002). Dependence analysis in reduction of requirement based test suites. *Proceedings of the* ACM *Intern. Symposium on Software Testing and Analysis* (pp. 107–111).

[17] Korel, B., Koutsogiannakis, G., & Tahat, L. (2007). Prioritization algorithms for regression testing in model based systems. *Proceedings of the 3rd ACM Workshop on Advances in Model Based Testing* (pp. 34-43).

[18] Tahat, L., & Almasri, N. (2012). Identifying the effect of model modifications in State-Based Models and System. *Journal of Advanced Computer Science and Technology*, *2(1)*, 9-27.

[19] Ferrante, K., Ottenstein, K., & Warren, J. (1987). The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, *9(5)*, 319–349.

[20] Stamatis, D. H. (2003). *Failure Mode and Effect Analysis: FMEA from Theory to Execution*. ASQ Quality Press.

[21] Bixin, L. Sun, X., & Leung, H. (2013). A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability*, *23(8)*, 613-646.

[22] Briand, L. Labiche, Y., O'Sullivan, L., & Sowka, M. (2006), Automated impact analysis of UML models. *Journal of Systems and Software*, *79(1)*, 339–352.

[23] Korel, B., Tahat, L., & Harman, M. (2005). Test prioritization using system models. *Proceedings of the IEEE International Conference on Software Maintenance* (pp. 559-568).

Nada Almasri is an assistant professor of Management Information Systems at Gulf University for Science and Technology. Prior to that, she worked as a full-time lecturer at the University of Waterloo, David R. Cheriton School of Computer Science, Canada. She received her master's degree in information systems from INSA Lyon University (The National Institute of Applied Sciences of Lyon) in 2000 and she obtained her Ph.D. degree in computer science in 2005 also from INSA Lyon, France. Her research interests include software development, maintenance, and deployment, in addition to component-based distributed systems and systems management.

Luay Tahat is an assistant professor in the Management Information System and Computer Sceince Department at Gulf University for Sceince and technology since 2008. Prior to that, He was the lead mobile network solution architect at Alcatel-Lucent in Naperville, USA. He has a master's degree in computer science from Northeastern Illinois University in Chicago and a Ph.D. in computer science from the Illinois Institute of Technology (IIT), also in Chicago. In his time at Alcatel-Lucent, Dr. Tahat has held several positions in software development, system engineering, and system architecture and has contributed to several areas in the fields of software engineering. His research interests include software testing, software maintenance, and wireless network solutions. The results of his research were published in several Journals and conference proceedings.

Mohammad Aref Alshraideh is an associate professor of computer science at the University of Jordan, Jordan. He received his B.Sc. degree in computer science in 1988 from Mu'tah University in Jordan and a master degree in computer science in 2000 from University of Jordan. He obtained his Ph.D. degree in computer science from University of Hull, UK, in 2007. During his graduate studies he obtained a fellowship from the University of Jordan. He was a head director assistant for computer technology at the Hospital of the University of Jordan until June 2012. Also he was working as a human resource director at the University of Jordan until 2015. Dr. Alshraideh is currently working as a registrar general at the University of Jordan. His research interests include software testing, artificial intelligence, and data mining.