Enhanced V-Model

Mustafa Seçkin Durmuş
Sadenco, Safe, Dependable Engineering and Consultancy, Antalya, Turkey
E-mail: msd@saden.co

İlker Üstoğlu
Yıldız Technical University, Department of Control and Automation Engineering, Istanbul, Turkey
E-mail: ustoglu@yildiz.edu.tr

Roman Yu. Tsarev
Siberian Federal University, Department of Informatics, Krasnoyarsk, Russia
E-mail: tsarev.sfu@mail.ru

Josef Börcsök
Kassel University, Computer Architecture and System Programming Department, Kassel, Germany
E-mail: j.boercsoek@uni-kassel.de

Keywords: Software development lifecycle, V-model, fault diagnosis, discrete event systems, EN 50128, fixed-block railway signaling systems

Received: November 16, 2017

Typically, software development processes are time consuming, expensive, and rigorous, particularly for safety-critical applications. Even if guidelines and recommendations are defined by sector-specific functional safety standards, development process may not be completed because of excessive costs or insufficient planning. The V-model is one of the most well-known software development lifecycle model. In this study, the V-model lifecycle is modified by adding an intermediate step. The proposed modification is realized by checking the fault diagnosability of each module. The proposed modification provides three advantages: (1) it checks whether the constructed model covers all software requirements related with faults; (2) it decreases costs by early detection of modeling deficiencies before the coding and testing phases; and (3) it enables code simplicity in decision of fault occurrence.

1 Introduction

The concept known as Safety Integrity Level (SIL) is used to quantify safety. The SIL is a degree of safety system performance for a Safety Instrumented System (SIS), which is an automatic system used to avoid accidents and to reduce their impact both on humans and the environment. A SIS has to execute one or more Safety Instrumented Functions (SIFs) to maintain a safe state for the equipment under control [1]. Bear in mind that, a safe state is known as the state where the whole system is prevented from falling into a dangerous situation. A SIF has a designated SIL level depending on the ratio of risk that needs to be decreased. IEC 61508, the standard for functional safety of electrical/electronic/programmable-electronic Safety Related System (SRSs), mentions that a SIL should be designated to each SIF and defines the safety integrity as the probability of a SRS adequately performing the required safety functions under all the stated conditions within a given period of time from the lowest requirement level (SIL 1) to highest requirement level (SIL 4).

The third part of IEC 61508 applies to any software used to develop a safety-related system within the scope of first and second parts of the standard, and establishes the requirements for safety lifecycle phases. Industry and domain specific implementations of IEC 61508 include IEC 61511 for industrial processes, IEC 61513 for the nuclear industry, and IEC 62061 for machinery etc.

A lifecycle model is defined in [2] as a model that describes stages in a software product development process. The IEC 61508-4 standard discusses the term lifecycle in the context of both safety lifecycle and software lifecycle. The safety lifecycle includes the necessary activities involved in the implementation of SRSs [3]. IEC 61508 states that a safety lifecycle for software development shall be selected and specified during the safety-planning phase in accordance with Clause 6 of IEC 61508-1. The safety lifecycle includes the definition of scope, hazard and risk analysis, determination of safety requirements, installation, commissioning, validation, operation, maintenance, repair, and decommissioning. On the other hand, the software lifecycle includes the activities occurring from the conception of the software to the decommissioning of the software.
Numerous lifecycle models have been addressed in the literature, such as the waterfall, spiral, iterative development, and butterfly models [4-8]. However, despite the availability of many lifecycle alternatives, safety standards such as IEC 61508, EN 50126, EN 50128, and IEC 62278 recommend using the V-model for software development processes. The V-model lifecycle has been applied to various domains such as the automotive [9], aerospace [10], railways [11], and the nuclear industry [12].

In this study, a Discrete Event System (DES)-based fault diagnosis method is added to the V-model lifecycle as an intermediate step between the module design and the coding phases. A DES is a discrete-state, event-driven system in which the state evolution of the system depends totally on the occurrence of discrete events over time.

The main difference of the proposed enhancement is its simplicity, when compared with the existing model checking tools and techniques in the literature [13, 14]. Because the fault diagnoser is built from the software model itself and; since the modular approach is a must in the Software Design Phase of the V-model in EN 50128 (recommended as mandatory) there is no need for any tool to check the diagnosability of a simple software module (component) model [15]. The remainder of this paper is organized as follows. The V-model lifecycle and the modified V-model lifecycle are explained in Sections 2 and 3, respectively. DES-based fault diagnosis is introduced in Section 4, and conclusion section is given in Section 5.

2 V-model lifecycle

Paul Rook introduced the V-model lifecycle in 1986 as a guideline for software development processes [2]. The primary aim of the V-model is to improve both the efficiency of software development and the reliability of the produced software. The V-model offers a systematic roadmap from project initiation to product phase-out [2]. The V-model also defines the relationship between the development and test activities; it implements verification of each phase of the development process rather than testing at the end of the project. The V-model, as defined in IEC 61508-3, is shown in Figure 1.

Before initializing a software development process according to the V-model, a software planning phase has to be realized, wherein a software quality assurance plan, software verification and software validation plans, and a software maintenance plan are fully defined. Later, the software requirements should be determined in cooperation with both the customer and the stakeholders. Using the selected software architectures (including modeling methods), software modules are developed by the designers. Each phase is verified immediately after completion. Note that, the left side of the V-model in Figure 1 represents the decomposition of the problem from the business world to the technical world [17]. After the coding phase, the right side of the V-model denotes the testing phase of the developed software.

The number of person may expand in the development process but this expansion shall be identified from the very beginning of the project. In [2], the change of the total number of software development teams are illustrated as given in Figure 2. Additionally, the cost of detection of faults in the different phases of V-model is given in Figure 3.

![Figure 1: V-model software safety integrity and development lifecycle [16].](image1)

![Figure 2: Software development teams [2].](image2)

![Figure 3: Software development which Fault was Detected](image3)

The advantages and disadvantages of the V-model can be summarized as follows [4, 5]:

**Advantages**

1. Facilitates greater control due to the standardization of products in the process.
2. Cost estimation is relatively easy due to the repeatability of the process.
3. Each phase has specific products.
4. Greater likelihood of success because of the early development of test plans and documentation before coding.
5. Provides a simple roadmap for the software development process.
Disadvantages
1. Low flexibility, expensive, and difficult to change scope.
2. No early prototypes.
3. Addresses software development within a project rather than a whole organization.
4. Too simple to precisely reflect the software development process and may steer managers into a false sense of security.

3  Modified V-model lifecycle

As mentioned in [19], and [20], the required workforce and the cost of the development process of the software increases towards the end with respect to the initial phases of the development lifecycle. Therefore, the proposed modification is realized on the left side of the V-model.

In the usual software development process according to V-model, the fulfillment of the requirements are checked by realizing the module tests after coding. By checking the module diagnosability, one can decide if the module fully covers the software requirements related with faults or not (will be explained in section 4). This intermediate phase can be considered as the time consuming and an extra workload. However, rather than turning back again from the module testing phase to the module design phase in the V-model, the proposed phase provides a final inspection of modules before proceeding to the coding and module testing phases. The proposed V-model is given in Figure 4.

The proposed modification in Figure 4 has three unique advantages:

a. It checks whether the constructed model covers all software requirements related to faults: If the developed software model (see Table A.2 and Table A.17 of [15]) is not diagnosable, then the software model does not contain all software requirements related with the faults.

b. It decreases costs through early detection of modeling deficiencies before proceeding to coding and testing phases: As can be seen from Figure 4, after proceeding to the coding phase, the designer can only go back to the module design phase at the end of the module tests. Many studies showed that, it is 5 times more expensive to fix a problem at the design stage than in the course of initial requirements, 10 times more expensive to fix it through the coding phase, 20 to 50 times more expensive to fix it at acceptance testing and, 100 to 200 times more expensive to fix that error in the course of actual operation [20-23].

c. It enables designers to write simple and more readable code in decision of the faults: This will be explained with a simple case study in the next section.

4  DES-based fault diagnosis

An event is defined as an encountered specific action, i.e., an unplanned incident that occurred naturally or due to numerous conditions that are encountered simultaneously [24]. Events are classified as observable or unobservable events in a DES.

A DES system is considered as diagnosable if it is possible to identify, within a finite delay, occurrences of precise unobservable events that are referred to as fault events [25]. In other words, a system is diagnosable if the fault type is always identified within a uniformly bounded number of transition firings after the occurrence of the fault [26]. The diagnoser is obtained from the system model itself and carries out diagnostics to observe the system behavior. Diagnoser states involve fault information, and occurrences of faults are identified within a finite delay by examining these states [27].

Finite state machines and Petri nets are considered as DES-based modeling methods and, these methods are also highly recommended by functional safety standards (see [15]).

4.1  Basic petri net (PN) definitions

A Petri net [28] is defined as;

\[ PN = (P, T, F, W, M_0) \]  

(1)

where

- \( P = \{p_1, p_2, \ldots, p_i\} \) is the finite set of places,
- \( T = \{t_1, t_2, \ldots, t_k\} \) is the finite set of transitions,
- \( F \subseteq (P \times T) \cup (T \times P) \) is the set of arcs,
- \( W : F \rightarrow \{1, 2, 3, \ldots\} \) is the weight function,
- \( M_0 : P \rightarrow \{0, 1, 2, 3, \ldots\} \) is the initial marking,
- \( p \cap T = \emptyset \) and \( p \cup T = \emptyset \).

For a marking \( M \), \( M(p) = n \) represents the token number of the \( i \)th place where it is equal to \( n \) [28]. Representation of a marking \( M : P \rightarrow \{1, 2, 3, \ldots\} \) can be realized by a \( k \)-element vector, where \( k \) denotes the total number of places.

**Definition 1 [28]:** If a PN has no self-loops, then it is considered as **pure** and when all arc weights of a PN are 1, then it is said to be **ordinary**.

**Definition 2 [28]:** \( M_t \) means that, the marking \( M_t \) is reachable from the initial marking \( M_0 \) by the \( t_1 \ldots t_k \) transitions sequence. Note
that, \( R(M_0) \) denotes the set of all reachable markings from \( M_0 \).

**Definition 3 [29]:** A PN is said to be free from deadlocks if it is possible to find at least one enabled transition at every reachable marking.

The set of places, \( P \) is partitioned into a set of observable places and a set of unobservable places \( (P_o, P_u) \). Likewise, the set of transitions, \( T \) is partitioned into a set of observable transitions and a set of unobservable transitions \( (T_o, T_u) \). Thus, the partitioned sets for \( P \) and \( T \) can be expressed as

\[
P = P_o \cup P_u \quad \text{and} \quad P_o \cap P_u = \emptyset
\]

\[
T = T_o \cup T_u \quad \text{and} \quad T_o \cap T_u = \emptyset
\]  

(2)

In addition, a subset \( T' \) of \( T_w \) represents a faulty transitions set. It is assumed that there are \( m \) different fault types. Here, \( \Delta = \{ \Delta_1, \Delta_2, \ldots, \Delta_m \} \) is the set of fault types. \( T' \) is expressed as \( T'_i = T_{i1} \cup T_{i2} \cup \cdots \cup T_{im} \), where

\[ T_{ir} \cap T_{js} = \emptyset \quad (\text{if } i \neq j). \]

\[ \Delta = \{ N \} \cup 2^\nu \]

is used to define the label set, where \( N \) is used to represent the label “normal,” which specifies that all fired transitions are not faulty, and \( 2^\nu \) represents the power set of \( \Delta_\nu \). In the remainder of this paper, unobservable transitions and places are represented as shown in Figure 5.

![Unobservable transition and place](image)

![Observable transition and place](image)

**Figure 5:** Representation of places and transitions.

### 4.2 Diagnosis of faults by using PN models

Since the system model contains unobservable places, it is not always possible to distinguish some markings. Thus, if \( M_i (p) = M_j (p) \) for any \( p_i \in P_o \), then it is denoted as \( M_i = M_j \). That is to say, \( M_i \) and \( M_j \) markings have the same observations. As done in [30], the definition of the quotient set \( \hat{R}(M_o) \) according to the equivalence relation (=) is useful; \( \hat{R}(M_o) := R(M_o) / := \{ M_o, \ldots, M_o, \ldots \} \) where \( M_o \in \hat{M}_o \). An observable marking or the observation of a marking is represented by each member of \( \hat{R}(M_o) \). We assume the following two statements are true for simplicity.

**Assumption [25, 26]:** Only deadlock free PNs are considered and there does not exist an order of unobservable transitions whose firing produces a cycle of markings that have the same observation.

A diagnoser is given for a PN [26, 27] by

\[
G_d = (Q_d, \Sigma_d, \delta_d, q_0) .
\]  

(3)

The diagnoser given by (3) is an automaton where the set of states are represented by \( Q_d \subseteq Q \), the set of events are represented by \( \Sigma_d = \hat{R}(M_o) \cup T_o \), the notation \( \delta_d : Q_d \times \Sigma_o \rightarrow Q_d \) represents the partial state transition function, and the initial state is denoted by \( q_0 = (M_0, N) \). A diagnoser state \( q_d \) is given as \( q_d = ((M_i, l_i), (M_i, l_i), \ldots, (M_i, l_i)) \), which involves pairs of a marking \( M_i \in R(M_o) \) and a label \( l_i \in \Delta \). The state set \( Q \subseteq Q \) represents the reachable states from the initial state \( q_0 \) by using \( \delta_d \). Each observed event \( \sigma_o \in \Sigma_o \) represents an observation of a marking in \( \hat{R}(M_o) \) or an observable transition in \( T_o \). The state transition function \( \delta_d \) is defined with the use of the label propagation function and the range function.

The label propagation function associates a label (faulty or normal) over a sequence of transitions. If the sequence of transitions does not contain any faulty transition, the resulting marking is labeled as normal (N). Detailed explanation of the label propagation function, the range function and the state transition function can be seen from [25-27, 30, 31].

### 4.3 Obtaining diagnosability

A PN is diagnosable if, and only if, the states of the diagnoser given by (3) shall be \( F_\nu \)-certain or does not involve any \( F_\nu \)-indeterminate cycle for any fault type \( F_\nu \).

Due to page restriction, the reader is referred [25-27] for detailed explanation of DES-based fault diagnosis and the proof of this theorem.

### 4.4 Railway point example

Trains can move from one track to another by the help of railway points (rail switches or point machines) placed at necessary locations. Points have two position indications, i.e., Normal (Nr) and Reverse (Rev). At any railway point, three main faults may occur. These faults are identified in the V-model software requirements specification phase as follows:

- **F1:** Point may not reach the desired position in a predefined time (e.g., 5 sec) while moving from Nr to Rev.
- **F2:** Point may not reach the desired position in a predefined time while moving from Rev to Nr.
- **F3:** Both position indications may be received simultaneously.

Examples of diagnosable and not diagnosable PN models of a railway point are given in Figure 6 and Figure 7, respectively. The meanings of the transitions and places of the models in Figure 6 and Figure 7 are given in Table 1 and Table 2, respectively. Note that the striped places and transitions represent unobservable places \( P_u \) and transitions \( T_u \) whereas the other places \( P_o \) and transitions \( T_o \) are observable. \( M_0 \) represents the initial marking of the PN. The underlined numbers in the diagnoser are used to represent the marking of an unobservable place.
Representation of the PN model in Figure 6 is as follows:

$$P_\pi = \{ P_{PM,1}, P_{PM,2}, P_{PM,5}, P_{PM,6}, P_{PM,8}, P_{PM,10}, P_{PM,12} \},$$

$$P_{\omega} = \{ P_{PM,3}, P_{PM,4}, P_{PM,7}, P_{PM,9} \},$$

$$T_\pi = \{ t_{PM,1}, t_{PM,2}, t_{PM,3}, t_{PM,4}, t_{PM,5}, t_{PM,6}, t_{PM,7}, t_{PM,8}, t_{PM,9}, t_{PM,10}, t_{PM,11}, t_{PM,12}, t_{PM,13} \},$$

$$T_{\omega} = \{ t_{PM_f,1}, t_{PM_f,2}, t_{PM_f,3} \}, \quad (4)$$

$$M_0 = \{ P_{PM,1}(M_0), P_{PM,2}(M_0), P_{PM,3}(M_0), P_{PM,4}(M_0), P_{PM,5}(M_0), M_0(\omega) \}.$$  

$$... M_0(\omega), M_0(\omega), M_0(\omega), M_0(\omega), M_0(\omega), M_0(\omega), M_0(\omega) = (0, 0, 0, 1, 0, 0, 0, 1, 0).$$

Figure 6: PN model of a railway point and its diagnoser (diagnosable).

Three different fault types given in Figure 6 are $\Delta_f = \{ F_1, F_2, F_3 \}$, where $T_{f_1} = \{ t_{PM_f,1} \}$, $T_{f_2} = \{ t_{PM_f,2} \}$, and $T_{f_3} = \{ t_{PM_f,3} \}$. The rectangles are used to diminish the complexity of the PN model. Each rectangle represents the label of the related place.

The diagnoser illustrated in Figure 6 is built from the PN model of railway point. A rectangle is used to denote each state and each state contains a pair of place markings and an attached label, normal (N) or fault. In other words, in parts of the diagnoser, a marking immediately after an observed event is detected precisely.

In accordance with the definition of the diagnoser in (3), a label which represents an observable transition or the observation of a marking is attached to all diagnoser state transitions.

In this study, with a slight abuse of notation, labels containing the observation of a marking or a pair of the observation of a marking and an observable transition are attached to all state transitions of the diagnoser.

<table>
<thead>
<tr>
<th>Place</th>
<th>Definition</th>
<th>Transition</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{PM,1}$</td>
<td>$N_r$ position requested</td>
<td>$t_{PM,1}$</td>
<td>$N_r$ position request</td>
</tr>
<tr>
<td>$P_{PM,2}$</td>
<td>$Rev$ position requested</td>
<td>$t_{PM,2}$</td>
<td>$Rev$ position request</td>
</tr>
<tr>
<td>$P_{PM,3}$</td>
<td>Point is moving to $N_r$</td>
<td>$t_{PM,3}$</td>
<td>$N_r$ ignored</td>
</tr>
<tr>
<td>$P_{PM,4}$</td>
<td>Point is moving to $Rev$</td>
<td>$t_{PM,4}$</td>
<td>$Rev$ ignored</td>
</tr>
<tr>
<td>$P_{PM,5}$</td>
<td>Point is in $N_r$</td>
<td>$t_{PM,5}$</td>
<td>$N_r$ Left</td>
</tr>
<tr>
<td>$P_{PM,6}$</td>
<td>Point is in $Rev$</td>
<td>$t_{PM,6}$</td>
<td>$Rev$ reached</td>
</tr>
<tr>
<td>$P_{PM,7}$</td>
<td>Fault type $F_1$ has occurred</td>
<td>$t_{PM,7}$</td>
<td>$N_r$ Filtered</td>
</tr>
<tr>
<td>$P_{PM,8}$</td>
<td>Points faulty ($F_1$)</td>
<td>$t_{PM,8}$</td>
<td>$N_r$ Fault acknowledged</td>
</tr>
<tr>
<td>$P_{PM,9}$</td>
<td>Fault type $F_2$ has occurred</td>
<td>$t_{PM,9}$</td>
<td>$N_r$ Fault acknowledged</td>
</tr>
<tr>
<td>$P_{PM,10}$</td>
<td>Point is faulty ($F_2$)</td>
<td>$t_{PM,10}$</td>
<td>$N_r$ Fault acknowledged</td>
</tr>
<tr>
<td>$P_{PM,11}$</td>
<td>Unobservable fault restriction</td>
<td>$t_{PM,11}$</td>
<td>$N_r$ Fault acknowledged</td>
</tr>
<tr>
<td>$P_{PM,12}$</td>
<td>Point is faulty ($F_3$)</td>
<td>$t_{PM,12}$</td>
<td>$N_r$ Fault acknowledged</td>
</tr>
</tbody>
</table>

Table 1: Definition of transitions and places in the models given in Figure 6.

For example, at $\dot{M}_0$ in Figure 6, the event label $\dot{M}_0$ represents the observable marking $\dot{M}_0$ is observed by firing the unobservable transition $\dot{t}_{PM,10}$. Similarly, the diagnoser state changes by firing the unobservable transition $\dot{t}_{PM,10}$. Similarly, the diagnoser state changes from $\{((0,0,0,0,1,0,0,0,0,0,1,0),N)\}$, to $\{((0,1,0,1,0,0,0,1,0,0,0,1,0),N)\}$, as a function of firing the observable transition $t_{PM,10}$ with the observation $\dot{M}_1$ of the resulting marking. According to the definition given in Section 4.3, since all states are $F_0$-certain and there is no $F_0$-indeterminate cycle in the diagnoser, the PN model is diagnosable.

Representation of the PN model in Figure 7 is as follows:
\( P_0 = \{ P_{PM_1}, P_{PM_2}, P_{PM_3}, P_{PM_4}, P_{PM_5}, P_{PM_6}, P_{PM_7} \}, \quad P_\infty = \{ P_{PM_7} \}, \)

\( T_0 = \{ t_{PM_1}, t_{PM_2}, t_{PM_3}, t_{PM_4}, t_{PM_5}, t_{PM_6}, t_{PM_7} \}, \quad T_\infty = \{ t_{PM_{f1}}, t_{PM_{f2}}, t_{PM_{f3}} \}, \)

\[ M_0 = \left[ M_0 \left( P_{PM_1} \right), M_0 \left( P_{PM_2} \right), M_0 \left( P_{PM_3} \right), M_0 \left( P_{PM_4} \right), M_0 \left( P_{PM_5} \right), M_0 \left( P_{PM_6} \right), M_0 \left( P_{PM_7} \right) \right] = (0,0,1,0,0,0,0). \]

Figure 7: PN model of a railway point and its diagnoser (not diagnosable).

The diagnoser given in Figure 7 is not diagnosable because it is not possible to distinguish the fault type after observing the marking \( M_0 \). In this case, the PN model will identify only one of the faults \((F_1\) or \(F_2)\) while the obtained code from this PN model is running. Therefore, the designers should revise the PN model before proceeding to the coding phase; otherwise, this deficiency will result in an unsuccessful test case in the module testing phase.

### 4.5 Railway signal example

An example PN model of a Two-Aspect Signal (TAS) and its diagnoser is given in Figure 8. TAS is generally used in railway depot areas and has two signal color indications (red means stop and green means proceed). The meanings of the transitions and places of the model in Figure 8 are given in Table 3. It is assumed that two different faults may occur in TAS which are \(F_1\): Both signal aspects are lit at the same time; and \(F_2\): No signals are lit.

#### Table 2: Definition of transitions and places in the models given in Figure 7.

<table>
<thead>
<tr>
<th>Place</th>
<th>Definition</th>
<th>Transition</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{PM_2} )</td>
<td>Point is moving to (Rv)</td>
<td>( t_{PM_3} ) ((t_{PM_4}))</td>
<td>Point reached to (N_r) ((Rev))</td>
</tr>
<tr>
<td>( P_{PM_3} )</td>
<td>Point position is (N_r)</td>
<td>( t_{PM_5} ) ((t_{PM_6}))</td>
<td>Point request to (N_r) ((Rev))</td>
</tr>
<tr>
<td>( P_{PM_4} )</td>
<td>Point position is (Rev)</td>
<td>( t_{PM_7} ) ((t_{PM_8}))</td>
<td>Point moved to (N_r) ((Rev)) and the fault acknowledged</td>
</tr>
<tr>
<td>( P_{PM_5} )</td>
<td>Fault type (F_1) or (F_2) has occurred</td>
<td>( t_{PM_9} )</td>
<td>Filter time has expired</td>
</tr>
<tr>
<td>( P_{PM_6} )</td>
<td>Point is faulty ((F_1))</td>
<td>( t_{PM_{f1}} ) ((t_{PM_{f2}}))</td>
<td>Point indication fault</td>
</tr>
<tr>
<td>( P_{PM_7} )</td>
<td>Point is moving from one position to another</td>
<td>( t_{PM_{f3}} )</td>
<td>Point position fault</td>
</tr>
</tbody>
</table>

Figure 8: PN model of TAS and its diagnoser.

<table>
<thead>
<tr>
<th>Place (P_{S2})</th>
<th>Definition</th>
<th>Transition</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_{S2,1})</td>
<td>Signal is red</td>
<td>(t_{S2,1})</td>
<td>Turn signal to green</td>
</tr>
<tr>
<td>$P_{S2_2}$</td>
<td>Signal is green</td>
<td>$t_{S2_2}$</td>
<td>Turn signal to red</td>
</tr>
<tr>
<td>$P_{S2_3}$</td>
<td>Fault type $F_1$ has occurred</td>
<td>$t_{S2_3}$</td>
<td>Signal turned to red and the fault acknowledged</td>
</tr>
<tr>
<td>$P_{S2_4}$</td>
<td>Fault type $F_2$ has occurred</td>
<td>$t_{S2_4}$</td>
<td>Signal turned to red and the fault acknowledged</td>
</tr>
<tr>
<td>$P_{S2_F1}$</td>
<td>Unobservable fault restriction</td>
<td>$t_{S2_F1}$</td>
<td>Point aspect fault</td>
</tr>
<tr>
<td>$P_{S2_F2}$</td>
<td>Unobservable fault restriction</td>
<td>$t_{S2_F2}$</td>
<td>Point indication fault</td>
</tr>
</tbody>
</table>

Table 3: Definition of transitions and places in the models given in Figure 8.

To compare the simplicity in decision of the faults with and without a diagnoser, an example Programmable Logic Controller (PLC) code snippet of TAS model is shown in Figure 9 and Figure 10, respectively.

As can be seen from Figure 9 and Figure 10, decision of fault occurrence with a diagnoser is simpler than without a diagnoser. For the PLC code given in Figure 9, the diagnoser compares the actual states of the $PN$ model with predefined faulty states. When the faulty state of the diagnoser is fully matched with the marking of the actual $PN$ states, the diagnoser sets the corresponding fault label.

Moreover, since the V-model is modified by adding an additional step, we also defined a new task for the organizational structure of the software development team. The Diagnoser designer (DDes) is added to the referred organizational structure of the enhanced V-model as given in Figure 11 (PM: Project Manager, RQM: Requirement Manager, Des: Designer, IMP: Implementer, VER: Verifier, VAL: Validator, DDes: Diagnoser Designer, ASR: Assessor). The original organizational structure can be seen from EN 50128 [15].

Figure 10: Decision of fault for TAS without diagnoser.

Figure 11: Recommended organizational structure for the enhanced V-model for SIL3&SIL4 software.

5 Conclusion

Faults in a safety-critical system may cause severe harm to humans. Therefore, the development steps of software for such safety-critical systems must be executed very carefully. Designers, developers, and engineers must consider the recommendations of both the international safety standards and the national rules to satisfy the required safety level and fulfill requirements.

Although enhancing the V-model with DES-based fault diagnosis is time consuming, however, the advantages of this intermediate step are threefold: (1) it checks whether the developed model fulfills all software requirements related to the faults; (2) decision of faults with a diagnoser is simpler than without a diagnoser; and (3) an early check of the models is possible before
proceeding to the coding and testing phase because the V-model leads developers from the module testing phase to the module design phase rather than the coding phase.

On the other hand, when costs and work hours are considered, adding such an intermediate step to the V-model can result in considerable benefits to both project management and product development departments.

Acknowledgement

The authors are thankful to Enago (www.enago.com) for the review of the English language of the paper.

References


