Bridging Hardware and Software Diagnosis: Leveraging Fault Signature Matrix and Spectrum-Based Fault Localization Similarities

Louise Travé-Massuyès
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Franz Wotawa
Christian-Doppler Laboratory for Quality Assurance Methodologies for Autonomous Cyber-Physical Systems, Institute of Software Technology, Graz University of Technology, Austria

Abstract

This paper examines two prominent Fault Detection and Isolation methodologies: the Signature Matrix approach, traditionally used in hardware systems, and the Spectrum-based approach, applied in software fault localization. Despite their distinct operational domains, both methods share the objective of precisely identifying and isolating faults. This study aims to compare these approaches and to highlight their similarities in principle. Through a comparative analysis, we assess how the structured pattern recognition of the Signature Matrix method and the statistical analysis capabilities of the Spectrum-based approach can be synergized to enhance diagnostic processes of cyber-physical systems that are composed of both hardware and software components. The investigation is motivated by the prospect of developing a hybrid Fault Detection and Isolation strategy that incorporates the robust detection mechanisms of hardware diagnostics with the techniques used in software fault localization. The findings are intended to advance the theoretical framework of Fault Detection and Isolation systems and suggest practical implementations across varied technological platforms, thereby improving the reliability and efficiency of fault detection and isolation in both hardware and software contexts.

2012 ACM Subject Classification Computing methodologies → Causal reasoning and diagnostics; Software and its engineering → Software defect analysis; Hardware → Error detection and error correction

Keywords and phrases Diagnosis, Fault detection and identification, Software debugging

Digital Object Identifier 10.4230/OASIcs.DX.2024.5

Funding *Louise Travé-Massuyès*: This work is partially supported by 3IA Artificial and Natural Intelligence Toulouse Institute (ANITI), French "Investing for the Future - PIA3" program under the Grant agreement ANR-19-PI3A-0004".
Franz Wotawa: The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association is gratefully acknowledged.

1 Introduction

In the field of Fault Detection and Isolation (FDI), innovative methodologies have significantly advanced the diagnosis and management of faults in both hardware and software systems. Among these, the Signature Matrix approach and the Spectrum-based approach, particularly as applied in software fault localization, stand out as critical tools in the arsenal of diagnostic techniques. The Signature Matrix method relies on a well-defined structure where fault signatures are cataloged in a matrix format, facilitating the pinpointing of system faults and anomaly detection. On the other hand, the Spectrum-based approach, widely utilized in software fault localization, leverages the statistical analysis of pass/fail data across various execution points to identify potential faults.

Despite their application to different domains – hardware and software respectively – both approaches share a foundational goal: to systematically identify and isolate faults with high accuracy. This common objective underscores the importance of comparing these methodologies to explore their potential synergies and complementary capabilities. The Signature Matrix approach, with its robust framework for hardware systems, and the Spectrum-based approach, with its nuanced analysis of software execution, both strive to enhance fault diagnostic processes. They do so along paths that appear different at first glance but have much in common.

The motivation for this comparative study arises from the potential to cross-pollinate between these domains, leveraging insights from one to enrich the other. Such a comparison not only elucidates the strengths and weaknesses inherent in each method but also sets the stage for hybrid diagnostic strategies that could incorporate the precision of signature matrices and the statistical analysis capabilities of spectrum-based methods. This paper aims to bridge the methodological divide between hardware fault detection and software fault localization, proposing avenues for integrated approaches that enhance the reliability and efficiency of FDI systems across different technological realms. Through a detailed comparative analysis, we seek to contribute to both the theoretical advancement and practical application of FDI methodologies in complex engineering and software systems.

For this purpose, we briefly introduce the basic foundations of both approaches utilizing simple examples for illustration purposes. Furthermore, we discuss the similarities and differences between signature matrix approach and spectrum-based fault localization, and suggest a combined approach focusing on the integration of methods. The latter may lead to a methodology that can be handle diagnosis of hardware and software altogether.

2 The Fault Signature Matrix approach

The FDI control community focuses predominantly on physical systems. These systems, which include mechanical, electrical, and hydraulic components, among others, are integral to industries ranging from manufacturing to aerospace. The models used to represent these systems are typically derived from physical laws and principles, ensuring that the simulations and diagnostics closely mirror real-world dynamics. Such models are crucial for accurately predicting system behaviors and effectively diagnosing issues, providing a reliable basis for further analysis and application in practical settings. The emphasis on physical systems and their corresponding models underscores the importance of precise and applicable solutions in the domain of system engineering.

Diagnosis techniques are based on behavioral models that define the relationships between system quantities. The typical model may be formulated in the temporal domain, then known as a *state-space model* :

$$\Sigma(z,x) : \begin{cases} dx(t)/dt = h(x(t),u(t),\theta), \\ y(t) = g(x(t),u(t),\theta), \\ x(t_0) = x_0, \end{cases} \tag{1}$$

where the functions h and g are linear or nonlinear functions. $x(t) \in \mathbb{R}^n$ denotes the state variable vector, whose components are the internal variables to the system that cannot be directly measured. These variables form the set of unknown variables X. $y(t) \in \mathbb{R}^m$ and and $u(t) \in \mathbb{R}^l$ denote the output and input variable vectors, respectively. Output and input variables are measured. They respectively stand for the measurements delivered by sensors and for the controls applied by actuators to the system. They are gathered in the vector $z(t)$

and they form the set of known variables Z, also called *observations.* $u(t)$ may be equal to 0 in case of an uncontrolled system. The variables z(t) and x(t) are time-dependent functions. It is assumed that all sensor and control values are acquired/applied synchronously according to a given sampling rate. Therefore the explicit mention to time may be omitted and we may write z and x instead of $z(t)$ and $x(t)$, respectively.

In a more abstract form, a system model can be defined as follows.

▶ **Definition 1** (System model)**.** *A system model $\Sigma(z,x)$ is composed of a set of algebraic and/or differential equations $e_k(z^k,x^k), k=1,\ldots,n_e$, where z^k is a subvector of the vector of known variables z, and x^k is a subvector of the vector of unknown variables x.*

At the heart of FDI control methods lies the concept of *residual*, and a key challenge is the *generation of residuals.* Residual generators define *diagnosis tests.* The evaluation of residuals indeed brings the information of consistency or non consistency of a model with observations.

▶ **Definition 2** (Consistency of observations with a model)**.** *An observed trajectory z is deemed consistent with a model $\Sigma(z,x)$ if there is a trajectory for x that satisfies the equations of $\Sigma(z,x)$.*

▶ **Definition 3** (Diagnosis test for $\Sigma(z,x)$)**.** *A diagnosis test $\mathcal{T}$, also called a residual generator, for the model $\Sigma(z,x)$ is a system that inputs a subset of known variables $\tilde{Z} \subseteq Z$ and their derivatives and outputs a scalar r, named the* residual *such that, for all z consistent with $\Sigma(z,x)$, $\lim_{t\to\infty} r(t)=0$.*

When the model aligns with observations, the residuals tend to zero as time t tends to infinity. Conversely, discrepancies in residuals may occur if the model does not align. Practically, residuals are never precisely *zero* due to the impracticality of the noise-free assumption inherent to system (1). The evaluation of residuals involves statistical testing that considers the noise characteristics, as discussed in sources like [1]. Furthermore, optimization of residuals aims to enhance robustness against disturbances [7] and to incorporate uncertainties [4, 8]. In this paper, we assume an ideal system without noise.

Diagnosis tests are derived from the model (1) by using different subsets of equations, each forming a minimal structurally overdetermined (MSO) set [6], i.e. a subset of equations just overdetermined. A diagnosis test $\mathcal{T}$ may take the form of an Analytical Redundancy Relation (ARR) in which case $\mathcal{T}$ expresses as a relation of the form $r = arr(\bar{z}')$, where r is the residual and $\bar{z}'$ is a subvector of $\bar{z}$, and $\bar{z}$ stands for z and its time derivatives up to some (unspecified) order.

Assumption 1 – Let us assume, without loss of generality, that every equation of $\Sigma(z,x)$ represents the behavior of one and only one component $\mathcal{C} \in COMPS$, where $COMPS$ is the set of components composing the system $\Sigma(z,x)$.

▶ **Definition 4** (Equation/component support of a diagnosis test)**.** *The equation/component support $\Sigma' \subseteq \Sigma$ of a diagnosis test $\mathcal{T}$ is defined as the set of equations of the MSO set underlying $\mathcal{T}$, i.e., the subset of equations of model (1) that are used to derive $\mathcal{T}$. Given Assumption 1, the component support of a diagnosis test is isomorphic to its equation support.*

In general, two categories of faults are recognized: additive faults and multiplicative faults. Additive faults typically introduce a bias into the system, altering measurements or actuators by a consistent amount. This can occur due to issues like incorrect settings on an analog-to-digital converter or erroneous application of control voltages. On the other hand, multiplicative faults, also referred to as parameter faults, impact the system's dynamics and can compromise the stability of the system and restrict the functionality of components.

▶ **Definition 5** (Additive Fault). *An additive fault is characterized by the alteration of a measured quantity $z_i \in \mathbb{R}$ through the addition of an offset $f_i \in \mathbb{R}$, resulting in the sensor/actuator reading $z'_i = z_i + f_i$, rather than the original z_i.*

▶ **Definition 6** (Multiplicative Fault). *A multiplicative fault involves the alteration by a factor $f_i \in \mathbb{R}$ of a parameter $\theta_i \in \mathbb{R}$ defining one of the functions h or g of model 1, which modifies the dynamics of the system.*

Faults occur in system components, including sensors, actuators, or internal parts. A fault in a component $\mathcal{C}$, whose normal behavior is represented by the equation e, disrupts this equation. Consequently, this disruption may affect the diagnostic tests that include e in their equation support, or equivalently that include $\mathcal{C}$ in their component support.

▶ **Definition 7** (Fault signature matrix). *Given a model $\Sigma(z, x)$ and a set of diagnosis tests DT, a fault signature matrix (FSM) is a matrix where each column represents a component $\mathcal{C} \in COMP$ and each row a diagnosis test $\mathcal{T} \in DT$. A cell for a component $\mathcal{C} \in COMP$ and a diagnosis test $\mathcal{T} \in DT$ is set to 1 if and only if $\mathcal{C}$ belongs to the component support of $\mathcal{T}$, and 0 otherwise.*

The columns of the FSM provide the *fault signatures* of the faulty components while the lines provide the component supports of the diagnosis tests [3].

During the operation of the system, diagnosis tests are evaluated with the measurements. Their evaluation provides a value to the residuals and this value is binarized: 0 if the value is zero and 1 otherwise. If the residual of a test is 0, the diagnosis test passes while if it is different from zero, it fails. The evaluated residual binary vector is called the *observed signature.*

Diagnoses can be obtained in two ways [3]:

- By columns (the FDI way), i.e. by comparing the observation signature to the fault signatures,
- By rows (the DX way), i.e. by considering that at least one of the components in the component support of the failing tests must be incriminated (these provide *conflicts* in the sense of Reiter [11]).

To illustrate the above concepts and diagnosis process, consider the following two-tank system, illustrated in Fig. 1 [2, 10] whose objective is to provide water flow Q_o to a consumer area. The system comprises two tanks, $Tank_1$ and $Tank_2$, connected by a pipe. $Tank_1$ is filled by a pump $Pump$ to reach a nominal water level of $h_1 = 0.5m$. The water level in $Tank_1$ is regulated by a PI level controller $Cont$ that adjusts the inlet flow Q_p provided by the pump. The water flow Q_{12} between tanks is managed by valve V_b using an "ON/OFF" controller to maintain the water level h_2 within the range of $0.09m \leq h_2 \leq 0.11m$ in $Tank_2$. The outflow Q_o to a consumer area depends on the position of valve V_o, which is open in the nominal regime. The connecting pipe between the tanks is positioned at the bottom, equipped with valve V_b. The dynamic model of the non faulty behavior of the two-tank system is given by the two following equations:

$$Tank_1 : \quad \dot{h}_1 = \frac{Q_p - C_{vb} \cdot \mathrm{sgn}(h_1 - h_2)\sqrt{|h_1 - h_2|} \cdot U_b}{A_1} \tag{eq.1}$$

$$Tank_2 : \quad \dot{h}_2 = \frac{C_{vb} \cdot \mathrm{sgn}(h_1 - h_2)\sqrt{|h_1 - h_2|} \cdot U_b - C_{vo} \cdot \sqrt{h_2} \cdot U_o}{A_2} \tag{eq.2}$$

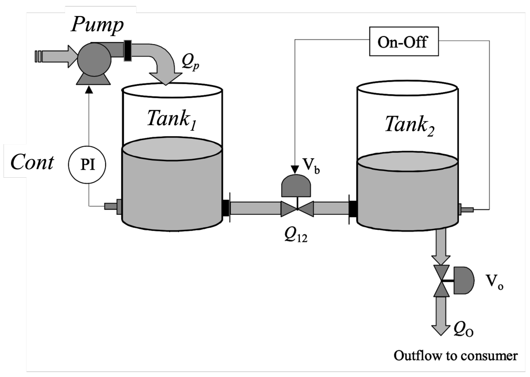

Figure 1 Two-tank system (from [2]).

We assume that Q_p is given by the output U_p of the PI controller, hence we have the following PI controller and pump equations:

$$Cont: \quad U_p = K_p(h_{1c} - h_1(t)) + K_i \int (h_{1c} - h_1(t))\, dt \tag{eq.3}$$

$$Pump: \quad Q_p = U_p. \tag{eq.4}$$

U_o and U_b represent the position of valve V_o and V_b respectively. These valves can only be open or closed, hence U_o and $U_b \in \{0, 1\}$. C_{vb} and C_{vo} are constant hydraulic flow coefficients, A_1 and A_2 are constant parameters of the tanks, h_{1c} is the target level for tank $Tank_1$, and K_p and K_i are constant parameters of the PI controller $Cont$.

Sensor equations are the following:

$$\text{Sensor } S_{h_1} \text{ of } h_1: \quad mh_1 = h_1 \tag{eq.5}$$
$$\text{Sensor } S_{h_2} \text{ of } h_2: \quad mh_2 = h_2 \tag{eq.6}$$
$$\text{Sensor } S_{Q_p} \text{ of } Q_p: \quad mQ_p = Q_p \tag{eq.7}$$
$$\text{Sensor } S_{U_b} \text{ of } U_b: \quad mU_b = U_b \tag{eq.8}$$
$$\text{Sensor } S_{U_p} \text{ of } U_p: \quad mU_p = U_p \tag{eq.9}$$
$$\text{Sensor } S_{U_o} \text{ of } U_o: \quad mU_o = U_o. \tag{eq.10}$$

Given the available measurements, ARRs are easily obtained by substituting unknown variables in the behavior equations. We hence obtain the following four diagnosis tests:

$$\mathcal{T}_1 : r_1 = -C_{vb}\,\text{sgn}(mh_1 - mh_2)\sqrt{|mh_1 - mh_2|}\, mU_b + mQ_p - A_1\left(\frac{dmh_1}{dt}\right)$$

$$\mathcal{T}_2 : r_2 = C_{vb}\,\text{sgn}(mh_1 - mh_2)\sqrt{|mh_1 - mh_2|}\, mU_b - C_{vo} \cdot \sqrt{(mh_2)} \cdot mU_o - A_2\left(\frac{dmh_2}{dt}\right)$$

$$\mathcal{T}_3 : r_3 = mU_p - K_p(h_{1c} - mh_1(t)) - K_i \int (h_{1c} - mh_1(t))\, dt$$

$$\mathcal{T}_4 : r_4 = mQ_p - mU_p.$$

The components and the corresponding equations representing their behavior are given in Table 1. The equation and component supports of the four diagnosis tests $\mathcal{T}_1$ to $\mathcal{T}_4$ are given in Table 2.

Let us consider a set of possible faulty components, without need to precisely identify the faults nor their type:

Table 1 Components and their corresponding equations in the two-tank system.

$Tank_1$	$Tank_2$	$Cont$	$Pump$	S_{h_1}	S_{h_2}	S_{Q_p}	S_{U_b}	S_{U_p}	S_{U_o}
(eq.1)	(eq.2)	(eq.3)	(eq.4)	(eq.5)	(eq.6)	(eq.7)	(eq.8)	(eq.9)	(eq.10)

Table 2 Equation and component supports of the diagnosis tests.

Test	Equation support	Component support
$\mathcal{T}_1$	{(eq.1),(eq.5),(eq.6),(eq.7),(eq.8)}	$\{Tank_1, S_{h_1}, S_{h_2}, S_{Q_p}, S_{U_b}\}$
$\mathcal{T}_2$	{(eq.2),(eq.5),(eq.6),(eq.8),(eq.10)}	$\{Tank_2, S_{h_1}, S_{h_2}, S_{U_b}, S_{U_o}\}$
$\mathcal{T}_3$	{(eq.3), (eq.5), (eq.9)}	$\{Cont, S_{h_1}, S_{U_p}\}$
$\mathcal{T}_4$	{(eq.4),(eq.7),(eq.9)}	$\{Pump, S_{Q_p}, S_{U_p}\}$

- Faulty plant components, i.e., $Tank_1$ and $Tank_2$. These components being faulty would disrupt equation (eq.1) and equation (eq.2) respectively,
- Faulty controller $Cont$, which would disrupt equation (eq.3),
- Faulty actuator $Pump$, which would disrupt equation (eq.4),
- Faulty sensors S_{h_1}, S_{h_2}, S_{Q_p}, S_{U_b}, S_{U_p}, and S_{U_o} that would disrupt equations (eq.5) to (eq.10).

The FSM is given by Table 3.

Table 3 FSM of the two-tank system.

	$Tank_1$	$Tank_2$	$Cont$	$Pump$	S_{h_1}	S_{h_2}	S_{Q_p}	S_{U_b}	S_{U_p}	S_{U_o}
$\mathcal{T}_1$	1	0	0	0	1	1	1	1	0	0
$\mathcal{T}_2$	0	1	0	0	1	1	0	1	0	1
$\mathcal{T}_3$	0	0	1	0	1	0	0	0	1	0
$\mathcal{T}_4$	0	0	0	1	0	0	1	0	1	0

Assume that the actual mode of the two-tank system is $Cont$ faulty and all the other components normal. Also assume that the evaluation of the diagnosis tests $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$, and $\mathcal{T}_4$ with measurements provides the observed signature is $(0, 0, 1, 0)^T$. The observed signature is matched against the faulty component signatures found int the FSM columns, which provides a unique one single diagnosis $\{Cont\}$, i.e. the PI controller.

3 Spectrum-based approach

Spectrum-based fault localization (SFL) is an approach based on probabilities considering different executions of a program. The underlying idea is to count how often statements are executed in passing and failing runs to compute a suspicious score for statements. For example, a statement that is only executed in passing runs can be considered least suspicious in causing a failure whereas a statement that is executed in all failing runs is most likely to be the reason behind wrong behavior. The first who carried out this idea were Jones and Harrold [5]. In their paper the authors introduced the basic concepts and their Tarantula tool. In the past decades, SFL has gained much attention in the debugging community as pointed out by Wong and colleagues [13].

In the following, we discuss the basic definitions and algorithms behind SFL considering an example from a previous publication of one of the authors [14]. Figure 2 shows a simple program comprising one class `Foo` and a method `foo`. The purpose of this program is to

```
1.  public class Foo {
2.   public static float a;
3.   public static float c;
4.   public static float pi = 3.14159;
5.   public static void foo (
6.             float d, int type) {
7.       if (type == 0) {      // Circle
8.          float r = d / 2;
9.          a = r * r * pi;
10.         c = d * pi;
11.      } else {
12.          if (type == 1) {  // Square
13.             a = d * d;
14.             c = 4 * d;
15.          } else {          // Error
16.             a = -1;
17.             c = -1;
18.          }
19.      }
20.   }
21. }
```

Figure 2 Simple program `Foo.foo` computing the circumference and the area of a circle and square depending on the second input (taken from [14]).

compute the circumference and the area of a circle and square. Table 4 shows typical tests we would apply for verifying the correctness of `Foo.foo`. In order to show how SFL computes the suspiciousness index, we need a fault version of a program, i.e., in our case `Foo.foo'`. In `Foo.foo'` we replace line 10 with `c = r * pi;` leading to a failing execution of test case T_1 from Table 4. Note that all other test cases are passing ones, i.e., delivering the expected values for both variables `a` and `c`.

To show SFL in action, we first define the notation of a *program spectrum* representing the execution of statements of a program for all test cases.

▶ **Definition 8** (Program spectrum). *Given a program* Π *and a test suite* TS*. A program spectrum is a matrix, where each line represents a statement of* Π *and each row a test case* $T \in TS$*. A cell for a statement* $S \in \Pi$ *and a test case* $T \in TS$ *is set to 1 if and only if the statement* S *is executed using test case* T*, and 0 otherwise.*

To access an element of the program spectrum in column i and row j, we write x_{ij}. The second important part of SFL is the error vector e. For each test case $T \in TS$ there is a corresponding entrance in e. This entrance is 1 if TS is a failing test case, i.e., the program is returning an unexpected output, and 0, otherwise.

In SFL the program spectrum and the error vector together are the *observation matrix*. In Table 5, we depict the observation matrix for `Foo.foo'`.

To compute the suspiciousness values for each statement, SFL, maps the observation matrix, i.e., the spectrum and the error vector, into 4 different kinds of metrics a_{ef} for each statement. Note that the a_{ef} represents the number of involvements of the given statement to executions or non-executions in passing and failing runs, where e is 1 if the statement is

Table 4 Test cases for program `Foo.foo` from Figure 2 (taken from [14].

Test	Fct. call	Expected result
T_1	`foo(1,0)`	`a=1.570795, c=3.14159`
T_2	`foo(1,1)`	`a=1.0, c=4.0`
T_3	`foo(1,2)`	`a=-1.0, c=-1.0`
T_4	`foo(0,0)`	`a=0.0, c=0.0`
T_5	`foo(0,1)`	`a=0.0, c=0.0`
T_6	`foo(0,2)`	`a=-1.0, c=-1.0`

Table 5 Applying SFL to `Foo.foo'` utilizing all 6 test cases.

	T_1	T_2	T_3	T_4	T_5	T_6	a_{00}	a_{01}	a_{10}	a_{11}	c_O	Rank
`7. if (type == 0) {`	1	1	1	1	1	1	0	0	5	1	0.408	2
`8.     float r = d / 2;`	1	0	0	1	0	0	4	0	1	1	0.707	1
`9.     a = r * r * pi;`	1	0	0	1	0	0	4	0	1	1	0.707	1
`10.    c = r * pi;`	1	0	0	1	0	0	4	0	1	1	0.707	1
`11. } else {`												
`12.    if (type == 1) {`	0	1	1	0	1	1	1	1	4	0	0.000	3
`13.        a = d * d;`	0	1	0	0	1	0	3	1	2	0	0.000	3
`14.        c = 4 * d;`	0	1	0	0	1	0	3	1	2	0	0.000	3
`15.    } else {`												
`16.        a = -1;`	0	0	1	0	0	1	3	1	2	0	0.000	3
`17.        c = -1;`	0	0	1	0	0	1	3	1	2	0	0.000	3
`18.    }`												
`19. }`												
Error	1	0	0	0	0	0						

executed and 0 otherwise, and f is 1 if the test case is failing and 0 otherwise. Formally, we define a_{ef} for $e = 0, 1, f = 0, 1$ as follows:

$$a_{00}(j) = |\{i|x_{ij} = 0 \wedge e_i = 0\}|$$
$$a_{01}(j) = |\{i|x_{ij} = 0 \wedge e_i = 1\}|$$
$$a_{10}(j) = |\{i|x_{ij} = 1 \wedge e_i = 0\}|$$
$$a_{11}(j) = |\{i|x_{ij} = 1 \wedge e_i = 1\}|$$

For example, the $a_{10}(j)$ for a statement j indicates the number of passing test cases where j is executed, whereas $a_{01}(j)$ represents the number of failing test cases where j is not executed. SFL now takes these a_{ef} values to calculate a suspicious value for each statement. It is worth noting that there are a lot of different similarity coefficients described in literature. For example, Wong et al. [13] mentioned more than 31 of them. However, in the following we utilize the Ochiai coefficient, which is one that is often used in practice because of delivering good results.

The Ochiai coefficient for statement j is defined as follows:

$$c_O(j) = \frac{a_{11}(j)}{\sqrt{(a_{11}(j) + a_{01}(j)) \cdot (a_{11}(j) + a_{10}(j))}}$$

Note that the fraction's denominator in the Ochiai coefficient might become 0. In this case, c_O is defined to be 0. Note that the Ochiai coefficients for the `Foo.foo'` program and their ranks, defined from the coefficients ordering, are depecited in Table 5. We see that three statements have the same coefficient and, therefore, the same rank and cannot be distinguished by SFL. However, there are extensions of SFL also considering data dependencies, e.g., [14] leading to better results.

4 Fault Signature Matrix versus Spectrum-based Fault Localization

The FSM approach is used primarily in hardware systems to detect and isolate faults. It involves monitoring system inputs and outputs through diagnostic tests that are built from a model of the physical system considering the impact of faults to parts of the model, e.g., the residual equations.

The SFL approach is a method used in software engineering to identify the location of faults within software by analyzing the program's behavior during execution. In particular, it considers executed statements for computing a fault likelihood of statements.

This section aims to highlight the similarities between the FSM and SFL approaches. Although the first is dedicated to hardware systems and the second to software, the concepts and objects they manipulate are comparable and the underlying principles and techniques share common ground.

4.1 About tests

In software engineering and in SFL, a test case targets to execute a set of statements and test cases are applied to the real program, i.e. the program is run with specific inputs for which the program output is known.

On the other hand, when diagnosing a physical system that is in operation, it is generally not possible to apply specific inputs to the system except in particular cases in which active diagnosis [9] is viable but we do not consider this case in this paper. So, the inputs and outputs of the physical system are taken as they are applied to the system by the controller, designed to achieve the current control objectives. In the FSM approach, diagnosis tests – or residual generators – are obtained from a mathematical model that is assumed to be a proper representation of the physical system. They establish a relation between observable variables. The physical quantities corresponding to observable variables that are linked by diagnosis tests are connected through a subset of components of the system that establish a path between these variables. These paths are identified through the equations representing these components. Indeed, a diagnostic test results from a process of eliminating non-observable variables, which can be implemented in the simplest case through sequential substitution [12]. When a fault occurs in a physical component $\mathcal{C}$ whose normal behavior is represented by the equation e, the inputs to the physical system that activate $\mathcal{C}$ generate outputs that should disrupt equation e and hence all the tests that have e in their equation support, or equivalently $\mathcal{C}$ in their component support. These test would then generate a non zero residual.

Let us remember that a test case in the SFL framework is noted as T and a test in the FSM framework as $\mathcal{T}$. Although the differences above in their design, they share the following commonalities:

- they are supported by a subset of statements (SFL) and a subset of components (FDI), respectively,
- the tests pass when the statements or the components represented by the equations that support the tests are normal,
- the tests may fail when the statements or the components represented by the equations that support the tests are faulty.

4.2 FDI fault signature matrix vs SFL observation matrix

Interestingly, it can be noticed that transposing the FSM of Table 3, and adding a row for the residual binary vector resulting from the evaluation of the tests, i.e. for the *observed signature* (called the *error vector* in SFL), we obtain an object that has the same semantics as the SFL observation matrix (recalled in Table 7) as displayed below in Table 6.

Table 6 Transp. FS matrix.

	T_1	T_2	T_3	T_4
$Tank_1$	1	0	0	0
$Tank_2$	0	1	0	0
$Cont$	0	0	1	0
$Pump$	0	0	0	1
S_{h_1}	1	1	1	0
S_{h_2}	1	1	0	0
S_{Q_p}	1	0	0	1
S_{U_b}	1	1	0	0
S_{U_p}	0	0	1	1
S_{U_o}	0	1	0	0
obs-error	?	?	?	?

Table 7 SFL observation matrix.

	T_1	T_2	T_3	T_4	T_5	T_6
`7.  if (type == 0) {`	1	1	1	1	1	1
`8.      float r = d / 2;`	1	0	0	1	0	0
`9.      a = r * r * pi;`	1	0	0	1	0	0
`10.     c = r * pi;`	1	0	0	1	0	0
`12.  if (type == 1) {`	0	1	1	0	1	1
`13.     a = d * d;`	0	1	0	0	1	0
`14.     c = 4 * d;`	0	1	0	0	1	0
`16.  a = -1;`	0	0	1	0	0	1
`17.  c = -1;`	0	0	1	0	0	1
obs-error	?	?	?	?	?	?

At this point, we can already say that the processes used by FSM and by SFL could be applied interchangeably on the FSM matrix or the observation matrix to obtain the diagnoses. In particular, we would be able to use SFL to come up with the evaluation of the suspiciousness value for each component and a ranking, which might be an improvement in some practical cases. The results of applying the SFL Ochiai method to the two-tank system for an observed signature/error vector $(1,0,0,0)^T$ – that we denote *obs-error* in the following – are given in Table 8.

Whenever an error vector is equivalently represented in a row, the suspiciousness index becomes 1.0. Hence, such a row would be ranked on the top indicating that the corresponding component is faulty. For example, in Table 8 the error vector is equivalent to the vector stored in the row of component $Tank_1$, which is ranked first in this example. This result is the same as the one we would obtain using the FSM method. However, there is a difference. If this error vector is not visible to 100%, the ranking of faults might help in case of double and triple faults. In such a case the ordinary FSM would not deliver any output and would have to be extended with the signatures of multiple faults to be effective [3]. Hence, the use of SFL could provide another way to handle multiple faults, avoiding the burdensome task of enumerating their signatures. This shows that utilizing the ideas from SFL could improve diagnosis reasoning of hardware. Conversely, directly comparing the error vector with the *execution signature* of each statement – defined as the program spectrum rows by analogy to fault signatures in FSM – using a specific metric such as Hamming distance (commonly applied in FSM) could also be a viable approach. Moreover, the methodology used in the FSM framework to construct diagnostic tests by eliminating unmeasured variables could offer useful insights for designing tests in software diagnostics.

4.3 Summary

The FSM and the SFL approaches share many commonalities considering components as program statements and the residual generators (i.e., diagnosis tests) as test cases. Considering an FSM as a program spectrum allows for computing a suspiciousness values (e.g., the Ochiai coefficients) and a fault ranking. The advantage of this mapping is that

Table 8 Applying SFL to the Two-tanks system.

	T_1	T_2	T_3	T_4	a_{00}	a_{01}	a_{10}	a_{11}	c_O	Rank
$Tank_1$	1	0	0	0	3	0	0	1	1.000	1
$Tank_2$	0	1	0	0	2	1	1	0	0.000	4
$Cont$	0	0	1	0	2	1	1	0	0.000	4
$Pump$	0	0	0	1	2	1	1	0	0.000	4
S_{h_1}	1	1	1	0	1	0	2	1	0.5774	3
S_{h_2}	1	1	0	0	2	0	1	1	0.7071	2
S_{Q_P}	1	0	0	1	2	0	1	1	0.7071	2
S_{U_b}	1	1	0	0	2	0	1	1	0.7071	2
S_{U_p}	0	0	1	1	1	1	2	0	0.000	4
S_{U_o}	0	1	0	0	2	1	1	0	0.000	4
obs-error	1	0	0	0						

also double and triple faults of hardware can be handled directly. On the other hand, the concept of an “observed signature” in FSM could be translated into an “execution signature” in SFL, and the methodology for constructing diagnostic tests in FDI could provide valuable guidance for test design in software diagnostics.

Based on these suggestions, we further discuss an integrated approach that allows a combined diagnosis of hardware and software in the following section.

5 Integrating FSM and SFL

In this section, we discuss how FSM and SFL can be combined together. For this purpose, we use a small example to illustrate the basic underlying ideas behind the method integration.

Let us consider the two tanks system and assume that the control U_p provided by the controller to control the pump is now provided by the program `Foo.foo2` that is a slightly modified version of the program `Foo.foo`. The PI controller is replaced by a controller delivering a control U_p proportional to the weight of water in the tank, which is equal to the area of the tank times the height of water. The tank may be circular or rectangular. Figure 3 depicts the program `Foo.foo2`.

In the model of this cyber-physical system, equation (eq.3) modeling the PI controller is now replaced by the code of the program `Foo.foo2`. We then can integrate the FSM and SFL observation matrices as shown in Table 9, where the grey row for *Cont* indicates that this row must be removed.

Note that test $\mathcal{T}_3$ that was previously supported by the component *Cont* (ghost line in gray) is now supported by the statements that are executed to deliver U_p (marked in bold in Table 9).

Let us consider two scenarios for this cyber-physical system:

- *Scenario 1*: the two-tank cyber-physical system suffers a single fault in the software `Foo.foo2` controller in statement 9, i.e.,`a=r*pi` instead of `a=r*r*pi`, which is marked in bold red in Table 9.
- *Scenario 2*: the two-tank cyber-physical system suffers a double fault, including the above fault in statement 9 plus a fault in the hardware sensor S_{h_1}.

```
1.  public class Foo {
2.   public static float a;
3.   public static float Up;
4.   public static float pi = 3.14159;
5.   public static void foo2 (
6.              float d, int type, float h) {
7.       if (type == 0) {      // Circle
8.          float r = d / 2;
9.          a = r * r * pi;
10.         Up = a * h;
11.      } else {
12.          if (type == 1) {  // Square
13.             a = d * d;
14.          Up = a * h;
15.          } else {          // Error
16.             a = -1;
17.             Up = -1;
18.          }
19.      }
20.   }
21. }
```

Figure 3 Program `Foo.foo2` computing the control of the pump U_p in the two-tank system for a circle or square tank shape depending on the second input.

5.1 Scenario 1: single fault

Scenario 1: the two-tank cyber-physical system suffers a single fault in the software `Foo.foo2` controller in statement 9, i.e.,`a=r*pi` instead of `a=r*r*pi`, which is marked in bold red in Table 9.

The application of the two methods, FSM and SFL to the two-tank system in Scenario 1 is illustrated in Table 10. The obs-error vector, resulting from the evaluation of $\mathcal{T}_1$ to $\mathcal{T}_4$ and from running test cases T_1 to T_6 is given by $(0, 0, 1, 0, 1, 0, 0, 1, 0, 0)^T$.

According to the FSM method, the obs-error vector matches the fault signatures of the single faults {stat. 8}, {stat. 9} and {stat. 10} ("stat." is used as an abbreviation of "statement"), which definitively indicates a fault in the software controller but does not discriminate between the three statements. The corresponding rows are colored green in Table 10.

According to the SFL method, the Ochiai coefficients provide a ranking that also puts at the first rank all three single faults {stat. 8}, {stat. 9} and {stat. 10}.

The results of the two methods are hence consistent. Note that these results are quite understandable because the actual fault, `a=r*pi` instead of `a=r*r*pi`, is not discriminable from a fault in statement 8, e.g. a wrong value for `d` in `r=d/2` or a fault in statement 10, e.g., a wrong value for `h` in `Up=a*h`.

5.2 Scenario 2: double fault

Scenario 2: the two-tank cyber-physical system suffers a double fault, including the above fault in statement 9 plus a fault in the hardware sensor S_{h_1}.

Table 9 FSM and SFL observation matrices integrated.

	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	T_1	T_2	T_3	T_4	T_5	T_6
$Tank_1$	1	0	0	0	0	0	0	0	0	0
$Tank_2$	0	1	0	0	0	0	0	0	0	0
$Cont$	0	0	1	0	0	0	0	0	0	0
$Pump$	0	0	0	1	0	0	0	0	0	0
S_{h_1}	1	1	1	0	0	0	0	0	0	0
S_{h_2}	1	1	0	0	0	0	0	0	0	0
S_{Q_p}	1	0	0	1	0	0	0	0	0	0
S_{U_b}	1	1	0	0	0	0	0	0	0	0
S_{U_p}	0	0	1	1	0	0	0	0	0	0
S_{U_o}	0	1	0	0	0	0	0	0	0	0
`7. if (type == 0) {`	0	0	**1**	0	1	1	1	1	1	1
`8.     float r = d / 2;`	0	0	**1**	0	1	0	0	1	0	0
`9.     a = r * pi;`	0	0	**1**	0	1	0	0	1	0	0
`10.    Up = a * h;`	0	0	**1**	0	1	0	0	1	0	0
`12.    if (type == 1) {`	0	0	0	0	0	1	1	0	1	1
`13.        a = d * d;`	0	0	0	0	0	1	0	0	1	0
`14.        Up = a * h;`	0	0	0	0	0	1	0	0	1	0
`16.        a = -1;`	0	0	0	0	0	0	1	0	0	1
`17.        Up = -1;`	0	0	0	0	0	0	1	0	0	1
obs-error	0	0	**1**	0	**1**	0	0	**1**	0	0

Dealing with multiple faults with the FSM method would require to generate their fault signature, which is commonly performed by applying a logical OR to the single fault signatures. In our scenario, according to the FSM method, the obs-error vector would then match the fault signatures of the double faults {stat. 8, S_{h_1}}, {stat. 9, S_{h_1}}, and {stat. 10, S_{h_1}}, identifying correctly and unambiguously the sensor S_{h_1} and the controller while obviously leaving undiscriminated {stat. 8}, {stat. 9} and {stat. 10}. The corresponding rows are colored green in Table 11.

According to the SFL method, the Ochiai coefficients provide a ranking that also puts at the first rank all three the three double faults {stat. 8, S_{h_1}}, {stat. 9, S_{h_1}}, and {stat. 10, S_{h_1}}.

The results of the two methods are here again hence consistent.

5.3 Summary

The two scenarios solved using the FSM and SFL methods within the integrated framework yield consistent results, demonstrating the compatibility of the two approaches and highlighting the effectiveness of the integrated framework in diagnosing cyber-physical systems.

6 Conclusion

In this paper, we have undertaken a comparative analysis of two key Fault Detection and Isolation (FDI) methodologies: the Faults Signature Matrix approach (FSM), commonly applied in hardware diagnostics, and the Spectrum-based fault localization (SFL) method, prevalent in software systems. Despite their distinct application domains, our analysis has shown that both approaches share fundamental principles. By leveraging the strengths of each method – structured pattern recognition in hardware systems and statistical analysis in software fault localization – there is significant potential to develop hybrid FDI strategies that address the challenges of cyber-physical systems, which increasingly integrate both hardware and software components.

Table 10 Single fault scenario – FSM and SFL observation matrices integrated (left of the table), Ochiai results (right of the table).

	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	T_1	T_2	T_3	T_4	T_5	T_6	a_{00}	a_{01}	a_{10}	a_{11}	c_O	Rank
$Tank_1$	1	0	0	0	0	0	0	0	0	0	6	3	1	0	0,0000	5
$Tank_2$	0	1	0	0	0	0	0	0	0	0	6	3	1	0	0,0000	5
$Pump$	0	0	0	1	0	0	0	0	0	0	6	3	1	0	0,0000	5
S_{h_1}	1	1	1	0	0	0	0	0	0	0	5	2	2	1	0,3333	4
S_{h_2}	1	1	0	0	0	0	0	0	0	0	5	3	2	0	0,0000	5
S_{Q_p}	1	0	0	1	0	0	0	0	0	0	5	3	2	0	0,0000	5
S_{U_b}	1	1	0	0	0	0	0	0	0	0	5	3	2	0	0,0000	5
S_{U_p}	0	0	1	1	0	0	0	0	0	0	6	2	1	1	0,4082	3
S_{U_o}	0	1	0	0	0	0	0	0	0	0	6	3	1	0	0,0000	5
stat. 7	0	0	1	0	1	1	1	1	1	1	3	0	4	3	0,6547	2
stat. 8	0	0	1	0	1	0	0	1	0	0	7	0	0	3	1,0000	1
stat. 9	0	0	1	0	1	0	0	1	0	0	7	0	0	3	1,0000	1
stat. 10	0	0	1	0	1	0	0	1	0	0	7	0	0	3	1,0000	1
stat. 12	0	0	0	0	0	1	1	0	1	1	3	3	4	0	0,0000	5
stat. 13	0	0	0	0	0	1	0	0	1	0	5	3	2	0	0,0000	5
stat. 14	0	0	0	0	0	1	0	0	1	0	5	3	2	0	0,0000	5
stat. 16	0	0	0	0	0	0	1	0	0	1	5	3	2	0	0,0000	5
stat. 17	0	0	0	0	0	0	1	0	0	1	5	3	2	0	0,0000	5
obs-error	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**						

Table 11 Double fault scenario – FSM and SFL observation matrices integrated (left of the table), Ochiai results (right of the table).

	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	T_1	T_2	T_3	T_4	T_5	T_6	a_{00}	a_{01}	a_{10}	a_{11}	c_O	Rank
$Tank_1$	1	0	0	0	0	0	0	0	0	0	5	4	0	1	0,4472	4
$Tank_2$	0	1	0	0	0	0	0	0	0	0	5	4	0	1	0,4472	4
$Pump$	0	0	0	1	0	0	0	0	0	0	4	5	1	0	0,0000	6
S_{h_1}	1	1	1	0	0	0	0	0	0	0	5	2	0	3	0,7746	1
S_{h_2}	1	1	0	0	0	0	0	0	0	0	5	3	0	2	0,6325	2
S_{Q_p}	1	0	0	1	0	0	0	0	0	0	4	4	1	1	0,3162	5
S_{U_b}	1	1	0	0	0	0	0	0	0	0	5	3	0	2	0,6325	2
S_{U_p}	0	0	1	1	0	0	0	0	0	0	4	4	1	1	0,3162	5
S_{U_o}	0	1	0	0	0	0	0	0	0	0	5	4	0	1	0,4472	4
stat. 7	0	0	1	0	1	1	1	1	1	1	1	2	4	3	0,5071	3
stat. 8	0	0	1	0	1	0	0	1	0	0	5	2	0	3	0,7746	1
stat. 9	0	0	1	0	1	0	0	1	0	0	5	2	0	3	0,7746	1
stat. 10	0	0	1	0	1	0	0	1	0	0	5	2	0	3	0,7746	1
stat. 12	0	0	0	0	0	1	1	0	1	1	1	5	4	0	0,0000	6
stat. 13	0	0	0	0	0	1	0	0	1	0	3	5	2	0	0,0000	6
stat. 14	0	0	0	0	0	1	0	0	1	0	3	5	2	0	0,0000	6
stat. 16	0	0	0	0	0	0	1	0	0	1	3	5	2	0	0,0000	6
stat. 17	0	0	0	0	0	0	1	0	0	1	3	5	2	0	0,0000	6
obs-error	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**						

The findings of this investigation highlight the complementary nature of these two methodologies, suggesting that combining their capabilities could lead to more robust and comprehensive diagnostic systems. This synergy could result in an integrated diagnostic framework capable of managing the intricacies of both hardware and software failures in modern, complex systems.

The theoretical insights gained from this study lay the groundwork for future research into hybrid FDI systems, with practical applications that span a wide range of technological platforms. By advancing the integration of these diagnostic techniques, we aim to contribute to the development of more reliable, efficient, and adaptable fault detection systems, ultimately improving the resilience and performance of cyber-physical systems.

7 References

References

1 Michele Basseville, Igor V Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. prentice Hall Englewood Cliffs, 1993.

2 B Ould Bouamama, R Mrani Alaoui, P Taillibert, and M Staroswiecki. Diagnosis of a two-tank system. *Intern Report of CHEM-project, USTL, Lille, France*, 2001.

3 M-O Cordier, Philippe Dague, François Lévy, Jacky Montmain, Marcel Staroswiecki, and Louise Travé-Massuyès. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2163–2177, 2004. doi:10.1109/TSMCB.2004.835010.

4 Carine Jauberthie, Nathalie Verdière, and Louise Travé-Massuyès. Fault detection and identification relying on set-membership identifiability. *Annual Reviews in Control*, 37(1):129–136, 2013. doi:10.1016/J.ARCONTROL.2013.04.002.

5 J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings ASE'05*, pages 273–282. ACM Press, 2005.

6 Mattias Krysander, Jan Åslund, and Mattias Nyberg. An efficient algorithm for finding minimal overconstrained subsystems for model-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(1):197–206, 2007. doi:10.1109/TSMCA.2007.909555.

7 Ronald John Patton and Jie Chen. Design methods for robust fault diagnosis. *Control Syst. Robot. Autom*, 16:84–111, 2009.

8 Vicenç Puig and Masoud Pourasghar. Fault diagnosis using set-membership approaches. *Fault Diagnosis of Dynamic Systems: Quantitative and Qualitative Approaches*, pages 237–261, 2019.

9 Ivo Punčochář and Jan Škach. A survey of active fault diagnosis methods. *IFAC-PapersOnLine*, 51(24):1091–1098, 2018.

10 Joseba Quevedo, Helem Sánchez, Damiano Rotondo, Teresa Escobet, and Vicenç Puig. A two-tank benchmark for detection and isolation of cyber attacks. *IFAC-PapersOnLine*, 51(24):770–775, 2018.

11 Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

12 Christofer Sundström, Erik Frisk, and Lars Nielsen. Selecting and utilizing sequential residual generators in fdi applied to hybrid vehicles. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(2):172–185, 2013. doi:10.1109/TSMC.2013.2248147.

13 W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Trans. Software Eng.*, 42(8):707–740, 2016. doi:10.1109/TSE.2016.2521368.

14 Franz Wotawa. Surveying and generalizing methods for combining dynamic slicing with spectrum-based fault localization. In *In* Proceedings of the International Workshop on Principles of Diagnosis (DX), Loma Mar, CA, USA, September 2023.