

Institutionen för systemteknik
Department of Electrical Engineering
Examensarbete

Survey and Evaluation of Diagnostic Tools

Examensarbete utfört i Fordonssystem
av

Markus Hertzman
Rickard Nilsson

LITH-ISY-EX--08/4100--SE
Linköping 2008

Survey and Evaluation of Diagnostic Tools

Master Thesis

Department of Electrical Engineering
Linköping University

Markus Hertzman

Rickard Nilsson

LITH-ISY-EX--08/4100--SE

Supervisor: **Markus Klein**
Anneli Crona

Examiner: **Mattias Krysander**

Linköping, 28 April 2008

Presentationsdatum 2008-03-17 Publiceringsdatum (elektronisk version) _____	Institution och avdelning Institutionen för systemteknik Department of Electrical Engineering	 Linköpings universitet
---	--	--

Språk ___ Svenska <input checked="" type="checkbox"/> Annat (ange nedan) Engelska _____ Antal sidor 99 _____	Typ av publikation ___ Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete ___ C-uppsats ___ D-uppsats ___ Rapport ___ Annat (ange nedan) _____	ISBN (licentiatavhandling) <hr/> ISRN LITH-ISY-EX--08/4100--SE <hr/> Serietitel (licentiatavhandling) <hr/> Serienummer/ISSN (licentiatavhandling)
---	---	---

URL för elektronisk version http://www.ep.liu.se

Publikationens titel Survey and Evaluation of Diagnostic Tools Författare Markus Hertzman & Rickard Nilsson
--

Sammanfattning If a fault occurs in a technical system, for example in an airplane, it is important to be able to detect that there is a fault and to find what in the system that is faulty. The procedure of determining, given certain observations, if faults are present and if so the location of faults is called a diagnosis. For achieving diagnosis we can use computer software that takes observations of a system as input and that generates a diagnosis as output. This is called a diagnostic system. To build a diagnostic system we need another piece of computer software which is called a diagnostic tool. This thesis will present a market survey for diagnostic tools as well as an analysis of three of the tools found in the survey. The analysis can be seen as constituted by two different aspects, one focusing on the diagnostic methods with which each tool creates diagnostic systems, the other focusing on practical details that determine the usability of each tool. The analysis found that the largest differences were between the methods used in creating the diagnostic systems.

Nyckelord Model based diagnostics, tool, TEAMS, eXpress, Raz'r.

Abstract

If a fault occurs in a technical system, for example in an airplane, it is important to be able to detect that there is a fault and to find what in the system that is faulty. The procedure of determining, given certain observations, if faults are present and if so the location of faults is called a diagnosis. For achieving diagnosis we can use computer software that takes observations of a system as input and that generates a diagnosis as output. This is called a diagnostic system. To build a diagnostic system we need another piece of computer software which is called a diagnostic tool. This thesis will present a market survey for diagnostic tools as well as an analysis of three of the tools found in the survey. The analysis can be seen as constituted by two different aspects, one focusing on the diagnostic methods with which each tool creates diagnostic systems, the other focusing on practical details that determine the usability of each tool. The analysis found that the largest differences were between the methods used in creating the diagnostic systems.

Acknowledgments

We would like to thank our supervisors at Saab, Markus Klein, Anneli Crona, and Torbjörn Fransson. At Linköping University, Mattias Krylander and Erik Frisk deserves a big thanks as well as Anna Axelsson and Lennart Andersson at Lund University. The good people at OCC'M, Oskar Dressler and at DSI, Jim Lauffer and Eric Gould have kindly aided us in our work. Thanks also to the people at TDCM for the company around the coffee table. Finally big thanks to friends and family for supporting us during this project.

Table of contents

<i>Abbreviations</i>	7
1 Introduction	9
1.1 Purpose	9
1.2 Problem	9
1.3 Prerequisites	9
1.4 Background	9
1.5 Method	9
1.5.1 Specification of Basic Desired Features of Tools	10
1.5.2 Searching for Suitable Alternatives	10
1.5.3 Further Investigation of Remaining Alternatives	10
1.5.4 Comparative Analysis of Remaining Alternatives	11
1.6 Thesis Outline	11
1.7 Contributions	11
2 Theory	13
2.1 Diagnostics	13
2.1.1 Model Based Diagnosis	14
2.1.2 Model Types	14
2.1.3 To Detect and Isolate Faults	17
2.2 XML	24
2.3 Document Type Definition	25
2.4 XML Schema	25
2.5 Relational Databases	26
2.6 AI-ESTATE Standard	26
3 Survey of Market for Diagnostic Tools	29
3.1 Non-diagnostic Desired Features	29
3.2 Basic Desired Features	29
3.3 The Chosen Tools	31
3.3.1 TEAMS from Qualtech Systems Inc	31
3.3.2 eXpress from DSI	32
3.3.3 Raz'r from OCC'M	32
3.4 Additional Motivations for Choices of Tools	32
3.4.1 Scope of Use	32
3.4.2 Software Based	32
3.4.3 Available Information	33
3.4.4 Named Product	33
3.4.5 Clear Overall Solution	33
3.4.6 Theoretical Foundation	33
3.4.7 Saab's Current Knowledge	33
3.5 Strong Candidates that did not make the Cut	34
3.5.1 Rodon from Sörman	34
3.5.2 Numerous Different Tools from Impact Technologies	34
3.5.3 Diagnostic Profiler & Diagnostician from VSE	34
3.5.4 Livingstone 2	34
3.6 Summary	34

4	<i>Analysis of the Tools</i>	37
4.1	TEAMS	37
4.1.1	Representation of Knowledge	38
4.1.2	Methods for Fault Detection and Isolation	44
4.1.3	Test Sequencing Algorithms	48
4.1.4	Formats for Storing Data	49
4.1.5	Design for Testability Aids	50
4.1.6	Modifiability	50
4.1.7	Compatibleness	51
4.1.8	Libraries	53
4.1.9	Disciplines of Modelling	53
4.2	eXpress	54
4.2.1	Representation of Knowledge	54
4.2.2	Methods for Fault Detection and Isolation	58
4.2.3	Test Sequencing Algorithms	62
4.2.4	Formats for Storing Data	63
4.2.5	Design for Testability Aids	65
4.2.6	Modifiability	65
4.2.7	Compatibleness	65
4.2.8	Libraries	67
4.2.9	Disciplines of Modelling	68
4.3	Raz'r	69
4.3.1	Representation of Knowledge	69
4.3.2	Methods for Fault Detection and Isolation	73
4.3.3	Formats for Storing Data	75
4.3.4	Design for Testability Aids	75
4.3.5	Modifiability	75
4.3.6	Compatibleness	76
4.3.7	Libraries	77
4.3.8	Disciplines of Modelling	78
5	<i>Comparison of tools</i>	79
5.1	Modelling Effort	79
5.2	Fault detection and Isolation Capabilities	80
5.3	Formats for Storing Data	81
5.4	Test Design Capabilities	81
5.5	Compatibleness	82
5.6	Libraries	83
5.7	Disciplines of Modelling	83
5.8	Modifiability	83
6	<i>Conclusions</i>	85
6.1	Discussion	85
6.2	Future Work	86
7	<i>References</i>	87
	<i>Appendix A - Search word list</i>	93
	<i>Appendix B - Found tools</i>	94
	<i>Appendix C - Sample Systems</i>	96

Abbreviations

AI-ESTATE	Artificial Intelligence Exchange and Service Tie to All Test Environments
CAD	Computer Aided Design
CAN	Controller Area Network
DFT	Design For Testability
Diag-ML	Diagnostics Markup Language
ECU	Electronic Control Unit
EDIF	Electronic Design Interchange Format
FDI	Fault Detection and Isolation
FDR	Fault Detection Rate
FIR	Fault Isolation Rate
FMEA	Failure Mode and Effects Analysis
FMECA	Failure Mode, Effects and Criticality Analysis
GDE	General Diagnostics Engine
GUI	Graphical User Interface
HDM	Hybrid Diagnostic Modelling
HTML	HyperText Markup Language
IETM	Interactive Electronic Technical Manual
JPL	Jet Propulsion Laboratory
MBD	Model Based Diagnosis
MTTF	Mean Time To Failure
ODBC	Open Database Connectivity
PDF	Portable Document Format
QSI	Qualtech Systems Inc.
RTF	Rich Text Format
RTML	Reusable Test and Model Library
TEAMATE	TEAM Automatic Test Equipment
TEAMS	Testability, Engineering And Maintenance System
TEAMS-KB	TEAMS Knowledge Base
TEAMS-RT	TEAMS Real Time
UUT	Unit Under Test
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
XML	Extensible Markup Language

1 Introduction

1.1 Purpose

The purpose of this thesis is to examine the market for available tools and standards for creating, maintaining, and developing systems used for diagnostics. The purpose is also to select among the available tools the best suitable tools and to do a comparison and evaluation of them from first of all a theoretical but also to some extent from a practical perspective.

1.2 Problem

The problem intended to be solved in this thesis is to find the currently available best suited tool for Saab. Best suited referring to capabilities such as modelling, updating and developing systems, capabilities for detection and isolation of failures, Design For Testability, DFT, aids, and generation of Failure Mode Effects and Critically Analysis, FMECA. Included in the problem description is also to map available standards within the field.

1.3 Prerequisites

The work is made under some prerequisites. Only tools with an expressed specialization towards model-based diagnostics are considered, this sieved out tools dealing with artificial intelligence, AI, not related to diagnostics and tools only dealing with Bayesian probability.

Furthermore the intention was not to investigate Rodon although this tool represents very much what Saab is looking for. This is because Saab already has made an investigation about this tool [28].

Saab is only interested in pure software with no integrated hardware in the scope of this thesis.

1.4 Background

Saab is increasing its focus on the civil aviation market. While the in-house tools that are used today for developing diagnostic systems serves its purpose, Saab is interested in exploring what diagnostic tools that are available on the market. It would ease the work if, when working with future civilian customers, Saab could use a commercial tool that would be compatible with the world outside Saab. For making work with diagnostics and function monitoring even more effective, Saab is also interested in knowing what standards that are supported by the different tools.

1.5 Method

The method used to find and evaluate the different standards and tools, which can be suitable for Saab, consists of a number of steps. These steps are described in a chronological order below. Also see Figure 1.

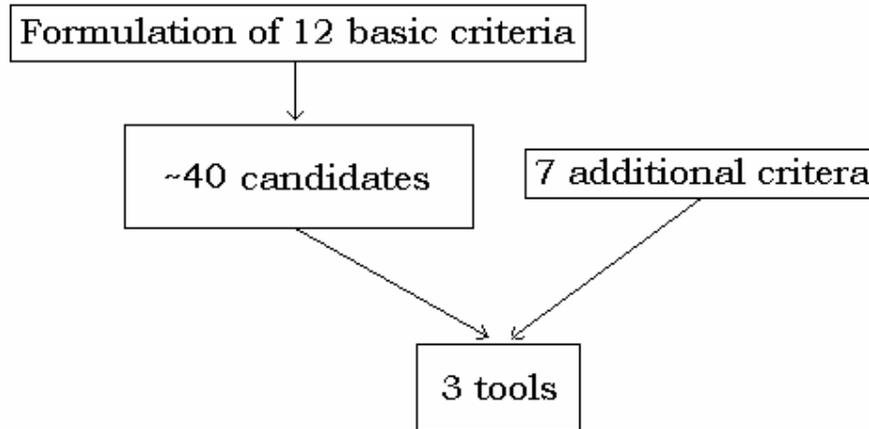


Figure 1. Schematic picture of the search method.

1.5.1 Specification of Basic Desired Features of Tools

To be able to have some basic rules of selection of the tools, a number of basic desired features are formulated. These rules are very loosely formulated and are supposed to give room for a wide variation of tools, although still provide a number of functions or features of which at least one is required to be included for the tool to be considered as an alternative.

1.5.2 Searching for Suitable Alternatives

In order to find available tools, brainstorming is done for possible search words and search phrases that can be used when searching on the internet. These words and phrases are listed in Appendix A. The list is consulted during the search phase and is refined in the searching process as new words and phrases are discovered.

In another document, found in Appendix B internet pages with possible candidates are noted along with description of the different tools.

The searching process results in a list with all the possible candidates that have been found. These are explored in more detail by gathering as much information as possible from internet pages and e-mailing contact. As information is gathered, the candidates on the list are either rejected or accepted for further exploration. This results in a shorter list consisting of a few candidates of tools.

1.5.3 Further Investigation of Remaining Alternatives

The remaining tools are explored more thoroughly by investigating the different pros and cons, and if possible by using the tool. The differences and capabilities of the different tools regarding choice of model types, methods for Fault Detection and Isolation (FDI), compatibleness, formats for storing the data and aids for designing a more testable system is examined for each program. Also some attention is paid to softer values such as user friendliness and difficulty of implementing models in the tool. Different aspects of the tools are considered by each author due to different knowledge.

1.5.4 Comparative Analysis of Remaining Alternatives

In the comparison of the tools the different pros and cons are weighed up against each other. Compatibility to other software commonly used in diagnostics is also taken into account. Long term softer values such as for how long it has been on the market and development possibilities are also investigated.

1.6 Thesis Outline

In Chapter 2 the theory needed to understand this thesis is described. Chapter 3 describes how the search for tools was done and how the tools were selected. Some of the found tools are listed and commented here. Also how the tools finally chosen for further investigation were selected is described here. In Chapter 4 the chosen tools are analysed and in Chapter 5 the chosen tools are compared. Chapter 6 concludes the work. Finally appendixes are attached.

1.7 Contributions

The authors have with this thesis contributed with a market survey of diagnostic software tools and performed a thorough evaluation of three of the tools that were found in the survey. Some standards within the field have been found and described. Markus has focused on the parts concerning data formats, model storing and compatibilities. Rickard has focused on model types, inference algorithms and design aids.

2 Theory

This section starts off by presenting the concept of diagnosis and then goes on to explain model based diagnosis, focusing on different model types. The section that follows after that deals with two different methods for inferring diagnoses. Thereafter different topics that are needed for understanding the remainder of the report are explained. These topics are XML, Document Type Definition, XML Schema, Relational databases, and finally the AI-ESTATE standard.

2.1 Diagnostics

For a technical system, there is a process for which there are variables which can be observed either by sensors or by an actual observation made by a human being [60]. These variables could be properties such as the water level in a tank or the cylinder pressure in an internal combustion engine. We know how these variables behave in a fault free mode of operation and in some cases also how they behave in fault modes of the system. The behaviour of the variables could refer to the value of the variable or the value of the derivative of the variable. If we compare the observations with the expected behaviours of the variables we might be able to determine if there is a fault present in the system. We might also be able to identify the fault as well. A simple example is that the water level in the tank should rise when more water is poured into the tank, given that the tank is in its fault free behavioural mode. If the level sinks even though no water is poured out of the tank we can suspect that the tank is in its leakage behavioural mode. It is called a diagnosis when a behavioural mode is found that could explain the current behaviours of the variables [60].

Performance of diagnostics can be divided into two kinds: Off-line or on-line, and the difference of those is that on-line diagnosis is performed at a run time mode of the system to be diagnosed. Off-line diagnostics on the other hand deals with already collected data series and tries to detect and isolate faults in this data [60].

If the diagnostics and tests are done manually by an engineer, service technician, or other person, the diagnosis system follows a diagnostic strategy. A diagnostic strategy is an ordered sequence of specific tests. The order of these tests has been chosen to optimize some criteria that can be selected by the person performing the tests manually. The criteria could be for instance to isolate the fault with the fewest tests or to first confirm that the fault is not in some specific subsystem. The strategy could also be viewed as a tree where each test is a vertex and the binary outcome leads via an edge to either a new test or to a leaf representing a diagnosis. This form of representation is called a diagnostic tree [60].

One common analysis method in diagnostics is FMECA. FMECA stands for Failure Mode, Effects, and Criticality Analysis and is an extension of FMEA, Failure Mode and Effects Analysis. In FMEA a systems possible failure modes are analysed in terms of what effects the failure modes have on the system. In FMECA the effects of the failure modes are also weighed against the criticality of the failure modes. The goal in an FMECA analysis is to find failure modes that have severe effects in the system so that actions can take place that would prevent these failure modes to occur [73].

2.1.1 Model Based Diagnosis

Diagnosis that is based on an explicit formal model of a system is called model based diagnosis [60]. In traditional diagnostics it is common to use what is called limit checking which means that a sensor signal is given a range in which it may vary but an alarm goes off if the signal goes either over or under the maximum respectively minimum value of the range. Redundant sensors are also often used in traditional diagnosis to make sure that the sensor itself is not faulty, e.g. one redundant sensor is added to be able to detect if any of the sensors are faulty and a third sensor can be added in order to isolate a faulty sensor [60].

In model based diagnosis, instead of comparing with two redundant sensors, we can compare the signal from the one sensor with a prediction from a model and from that comparison detect faults in the sensor [60].

Advantages of model based diagnosis, compared to limit checking, can be that it can detect smaller faults with shorter detection time and that it is more likely that faults can be isolated [60]. Compared to using redundant sensors, model based diagnosis has the advantage of not needing extra hardware which is desirable from both a cost perspective and a space perspective. The main disadvantage though is that the models that are needed in the model based diagnosis system can be time consuming and expensive to develop. In some cases a model with a high enough accuracy can not even be constructed if the behaviour of the system is unknown.

2.1.2 Model Types

Models in model based diagnosis can be of different types and constructed in different ways. These types infer pros and cons; a very detailed model gives the opportunity to isolate failures very precisely since the exact fault effect can be traced from the test points through the system. This does on the other hand require a fair amount of computing power to execute and the bigger the system the more computational power is required. A not so detailed model requires less computational power and can generate a diagnosis faster but the relationship between the fault and its effect on the test is less clear [16].

2.1.2.1 Structural Models

Parts of the spectrum of models will be described here starting with the most basic one, the structural model given in [16]. This type of model merely states that there is a relation between property one, P1, and property two, P2. The properties can be signals, components, or a property of a component in the model. The model thus consists of a set of ordered pairs, where each pair states that a fault in the first member will affect member two. Take as an example a fault in a resistor, r , that affects the current through it, i .

$$\{\{r, i\}\}$$

Notice that the model says nothing about how the faults affect the members, only that a fault in r is modelled to always affect i .

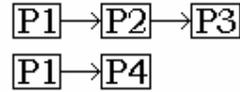


Figure 2. Illustration of a structural model.

The structural model of the system dependencies shown in Figure becomes

$$\{\{P1, P2\}, \{P1, P4\}, \{P2, P3\}\}$$

From this we can see that only the first order dependencies are included, the connection between $P1$ and $P3$ must be derived via $P2$.

2.1.2.2 Dependency Models

If the structural model is extended to also contain different fault modes, where each fault has its own set of properties, the model is said to be a dependency model. A fault mode is denoted by Px_i , where x represents the component number and i represents the current fault mode.

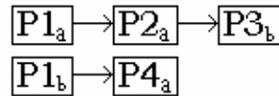


Figure 3. Illustration of a dependency model.

In this model type the sets of ordered pairs are component and fault mode specific, meaning that the model in Figure 3 could be modelled as

$$\{\{P1_a, P2_a\}, \{P1_b, P4_a\}, \{P2_a, P3_b\}\}$$

In this description all the dependencies have an index referring to the affecting or affected failure mode of the specific components. Thus $P1_a$ will affect $P2$ and make this component behave according to its fault mode a .

2.1.2.3 Quantitative Models

A more refined model type is so called quantitative models [7,16]. This model type specifies also the relation of how the components and signals affect each other. This can be illustrated by a component $P1$ that has as input x and from this input it generates an output y , as shown in Figure 4. The generation of the output is done by a function. Usually this function is derived from a physical relation such as Ohms law.

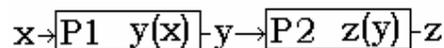


Figure 4. Illustration of a quantitative model.

This form of modelling is also called physical modelling since it often uses relations known from the physics of the systems. A problem with this model type is that many times the exact relations of the physical variables are not known.

A problem with the quantitative model type from a diagnostics perspective is that in large models the computations of all the needed variables can be demanding. This applies especially if the model contains stiff differential equations, i.e. differential equations with a large difference in the time coefficients [75]. A way to reduce the required computational power is to use qualitative models instead.

2.1.2.4 Qualitative Models

Qualitative models differ from quantitative models by not specifying or treating an exact value of the model variables but rather using notations as for example “low”, “middle”, and “high”. The number and spacing of possible outcomes of this quantification (defining the granularity) is a matter of much research and no general theory has been accepted. There are also different types of qualitative models, parted by the different ways they interpret the time representation. Here we are mostly interested in the type called naïve physics models. A more through description of this model type and the other types of qualitative models are given in [20].

All the naïve physics models have in common that they represent the state of a physical system in a very crude model, where the state variables only can assume one of three values: $\{-,0,+\}$. Sometimes the 0-state is considered only as a limit between the negative and positive states and other times it is considered as a state of its own. The reason for the blunt trisected division of states is that it in many times is relevant only in what direction the magnitude of a force, motion or other physical entity is changing. There are for instance applications where the exact magnitude of the voltage is irrelevant from a diagnostic perspective [20].

Consider for instance the simple example shown in Figure 5. In this circuit it is sufficient to know if the voltmeter indicates a voltage drop over the lamp or not. The voltage can be divided into two domains, 0 and +. Setting the limit between the domains in which the voltage is divided to a voltage corresponding to where it is possible to detect if the light is lit or not, 1 and 0. This gives enough information to differentiate between the systems no fault mode and two of its fault modes. One of these fault modes is “light broken”, when the light is not lit although there is a voltage drop over its connectors. Another of these fault modes is “battery drained”, which is when the battery no longer supplies a voltage difference between its connectors.

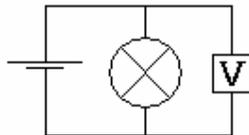


Figure 5. A schematic illustration of a very simple circuit.

If the voltmeter is showing + but the light is not lit, 0, then it is presumably the light that is broken. If the light is not lit, 0, and the voltmeter shows 0 then, presumably the battery is drained.

The method using the trisected variables is used by some while others employ a little more sophisticated method and increases the granularity of the variables but still uses the original ternary set for the derivatives.

2.1.3 To Detect and Isolate Faults

To be able to generate a diagnosis it is not enough to just have a model of the system. There must also be some way to compare the observations of the system with the observations we expect. A method to compare this and to from the comparison generate a conclusion is called an inference algorithm. In this section two different kinds of inference methods are described, fault detection and isolation, FDI and general diagnostics engine, GDE.

The efficiency of the method could be measured by using the Fault Detection Rate, FDR , or the Fault Isolation Rate for L failure modes, FIR_L , as specified in [3]. The first rate is the ratio between the number of times the fault have been detected, N_D , and the times the fault has occurred during a specific time, N . The second rate is the ratio between the number of diagnoses containing at most L failure states N_L and the number of detected failure modes under a specific time.

$$FDR = \frac{N_D}{N} \quad FIR_L = \frac{N_L}{N_D}$$

2.1.3.1 Fault Detection and Isolation

FDI for a system can be described as follows [60]. Let sys denote the system which is made up by the components $c_i, i = 1, 2, \dots, N$. Each component can have many ways in which it behaves and all these ways can be portioned into groups which we call component behavioural modes. If we consider a component c_i that has a behavioural mode bm we would write $c_i = bm$ to describe that the component behaves according to this certain behavioural mode. The fact that component c_i has k_i number of behavioural modes we denote as

$$c_i \in \{bm_1, \dots, bm_{k_i}\}.$$

The component c_i at each point of time behaves according to exactly one of its component behavioural modes. The behavioural mode of the system is represented by sys . To describe the behavioural mode of all the components, we write BM . The set of all the component behavioural modes are the same as the system behavioural mode,

$$sys = BM = \{c_1 = bm_{j_1}, c_2 = bm_{j_2}, \dots, c_N = bm_{j_N}\},$$

where $1 \leq j_i \leq k_i, \forall i \in \{1, 2, \dots, N\}$.

Let nf denote the component behaviour mode *no fault* and then for denoting that there is no fault in the system we write

$$sys = NF = \{c_1 = nf, c_2 = nf, \dots, c_N = nf\}.$$

If we let u denote control signals and y measurements of the system we can define $z = (u \ y)$ as a vector of all known signals in the system. Considering a model, Θ_{BM} is defined as the set of all observations z that are consistent with $sys = BM$.

Let us say that the model and system have the system behavioural modes $\{BM_1, \dots, BM_k\}$, and then the set of all possible observations is

$$Z = \Theta_{BM_1} \cup \dots \cup \Theta_{BM_k}.$$

Now we can define what a diagnosis is [28]: A system behavioural mode BM is a diagnosis if and only if $z \in \Theta_{BM}$ for an observation $z \in Z$.

To be able to create a diagnosis, a number of hypothesis tests are created. A hypothesis test is a test that can accept or reject a hypothesis, H^0 . H^0 is the assumption that the system is behaving according to some $BM \in S^0$. Here S^0 is a set of behavioural modes of components that would make the test pass. If H^0 is rejected the system behaves according to some behavioural mode in S^1 . A test, T , provides information if the null hypothesis, H^0 can be rejected or not. H^0 will not be rejected if $z \in \Theta_T$ and it will be rejected if $z \in \Theta_T^C$. A rejection of H^0 will result in the inference that the system is not behaving according to any of the behavioural modes in S^0 . If the test is not able to reject H^0 then a behavioural mode in S^0 can describe the behaviour of the system.

This means that all behavioural modes in S^0 are included in the set Θ_T . This in turn means that a behavioural mode in S^0 explains the behaviour of the system.

This will be illustrated with an example from [28].

Assume a system sys that has four different system behavioural modes: NF (No Fault), F_1 , F_2 , and F_3 . A hypothesis test T_1 has been created with the purpose to detect and alarm for F_1 and F_2 . The rejection region of the test is $\Theta_{T_1}^C$ and can be seen in Figure 6 together with the behavioural mode sets. If $z \in \Theta_{T_1}^C$ this means that the test have failed.

Since Θ_{F_1} is a subset of $\Theta_{T_1}^C$ and Θ_{F_2} has a non empty intersection with $\Theta_{T_1}^C$, these two system behavioural modes, F_1 and F_2 , will be included in S^1 . Analogous, since Θ_{NF} and Θ_{F_3} are subsets of Θ_{T_1} , and Θ_{F_2} has a non empty intersection with Θ_{T_1} , these three system behavioural modes will be included in S^0 . In summary the rejection region of the test results in $S^1 = \{F_1, F_2\}$ and $S^0 = \{NF, F_2, F_3\}$.

If the system behaves according to F_1 then $z \in \Theta_{F_1} \subset \Theta_{T_1}^C$ and results in rejecting the null hypothesis and the decision will be that $sys \in S^1 = \{F_1, F_2\}$.

The null hypothesis will only be rejected for $sys = F_2$ when $z \in \Theta_{F_2} \cap \Theta_{T_1}^C$. This means that the test will fail to react and detect $sys = F_2$ when $z \in (\Theta_{F_2} \setminus \Theta_{T_1}^C)$ and as a consequence alarms will be missed.

The test will never react when $sys \in \{NF, F_3\}$ and the null hypothesis will not be rejected and the decision will be that $sys \in \{NF, F_2, F_3\}$.

In summary this means that the following conclusions can be drawn from the test:

$$\begin{aligned} z \in \Theta_{T_1}^C &\rightarrow H^0 \text{ rejected, } sys \in S^1 = \{F_1, F_2\} \\ z \notin \Theta_{T_1}^C &\rightarrow H^0 \text{ not rejected, } sys \in S^0 = \{NF, F_2, F_3\} \end{aligned}$$

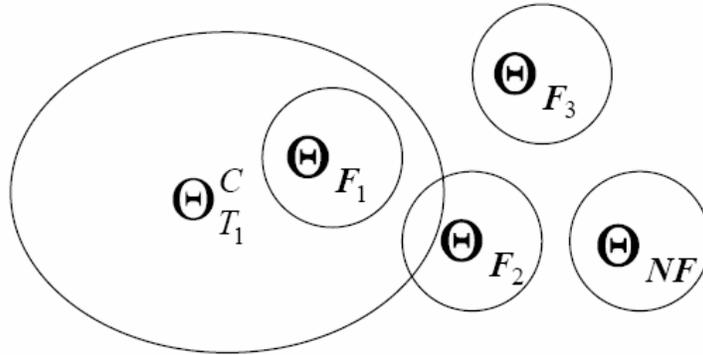


Figure 6. A system with four behavioural mode sets and the rejection region for the test used in the diagnosis system in the above example [28].

As seen in the example, with the results of the different hypothesis tests we can derive a set of possible behavioural modes that can explain the outcome of all tests. The different sets of behavioural modes being rejected or not in each test could be represented in a matrix form. This form is called a decision structure and each row in this matrix represents a test and each column represents a behavioural mode. The example shown in Figure 6 is represented in Table 1.

A “1” on row a and column b mean that test a , T_a , will reject behavioural mode b , bm_b , if the test passes, that is if $z \in \Theta_{T_a}$. If the test triggers an alarm, i.e. $z \in \Theta_{T_a}^C$, bm_b explains the behaviour of the system.

If there is an “X” on the same position this would mean that no matter if the test passed or failed bm_b explains the behaviour of the system.

With “0” in the same place and the test failed, $z \in \Theta_{T_a}^C$, bm_b can not explain the behaviour. If the test passed bm_b could explain the system behaviour.

Table 1. The decision structure for the test T_1 .

	NF	F_1	F_2	F_3
T_1	0	1	X	0

If there are more tests available the decision structure is expanded with more rows. If a decision structure were to represent the sets derived from the tests in the example from [28] it would look like Table 2.

Table 2. The decision structure of the tests and fault modes from the example in [28].

	NF	F_1	F_2	F_3
T_1	0	1	X	0
T_2	0	0	X	X
T_3	0	X	0	X

When the information from each test now have been condensed into this compact form the inference is done by the different tests that have reacted to what behavioural modes they indicate, or do not indicate.

As an example, consider the test results,

$$(T_1 \ T_2 \ T_3)^T = (\text{passed} \ \text{passed} \ \text{failed})^T .$$

This means since test T_1 not has reacted that the system can not be in behavioural mode F_1 . T_3 has reacted and this mean that the behavioural mode F_1 or F_3 will explain the system behaviour. Since the F_1 mode already have been ruled out by T_1 we now know that the system is in behavioural mode F_3 .

If only the first test, T_1 , would have reacted then we would not have been able to say whether the system is in mode F_1 or F_2 . This is due to the fact that T_1 is sensitive to both these modes. If we are not able to fully isolate between behavioural modes given certain test results we have a so called ambiguity group. The group consists of the behavioural modes between which we can not isolate.

2.1.3.2 General Diagnostic Engine

GDE is presented within the field of AI and is one of the most well known algorithms for fault diagnosis. The algorithm can be used to derive diagnoses and can also suggest additional measurements for improving the diagnosis. The input to the algorithm is a model of the system and observations, i.e. measurements made on the system. In this section we will only give a brief description of the diagnosis generation. The information in this section is taken from [60].

In GDE the components have only two modes of operation, i.e. behavioural modes. Either they are functioning correctly, inferring that the component is in behavioural mode OK or they are not functioning correctly, and the system is in mode NOK.

Assumption based Truth Maintenance System (ATMS) are used together with local propagation to derive a diagnosis. ATMS is an advanced data structure originally used as a general problem solver within AI. Local propagation is used to calculate the values of the variables in the model. A consequence of the local propagation is that the model can not contain circular dependencies. A circular dependency is when e.g. x depends on y , y depends on z and z in turn depends on x .

The ATMS consists of one assertion, A and one needed supporting environment, $\{cs_1, cs_2, \dots\}$, grouped together into an entity in the ATMS. The assertion contains a statement about one or more variables in the model, e.g. $x = 1$.

The needed supporting environment state sets of what components that needs to be functional for the linked assertion to hold, e.g. $\{\{c_1, c_3\}, \{c_5, c_6\}, \{c_9\}\}$. In [60] the needed supporting environment is called only supporting environment.

An entity in the ATMS could thus be stated as

$$\langle A, \{cs_1, cs_2, \dots\} \rangle = \langle x = 1, \{\{c_1, c_3\}, \{c_5, c_6\}, \{c_9\}\} \rangle$$

where A is a assertion $x = 1$ and cs are sets of components needed to be functioning, in this example $\{\{c_1, c_3\}, \{c_5, c_6\}, \{c_9\}\}$. The entity could also be stated in a logic fashion as

$$OK(c_1) \wedge OK(c_3) \vee OK(c_5) \wedge OK(c_6) \vee OK(c_9) \Rightarrow x = 1$$

The former notation is used in this section since it is more compact.

An entity with an empty supporting environment is called an ATMS-premise. This means that this assertion will always hold, it does not need any supporting environment. An assertion is called the *datum* and the needed supporting environment the *label* in ATMS terminology.

The local propagation uses the entities in the ATMS to step by step propagate the information fed into the algorithm. The propagation starts at the premises, the assertions without needed supporting environments. From there the information is propagated using related entities.

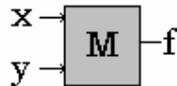


Figure 7. A schematic illustration of the multiplier system.

If we have a simple system consisting only of a multiplier, M , with two inputs and one output, as shown in Figure 7, this can be modelled by the entities,

$$\begin{aligned} \langle x = 1, \{\{\emptyset\}\} \rangle \\ \langle y = 4, \{\{\emptyset\}\} \rangle \\ \langle f = x * y, \{\{M\}\} \rangle \end{aligned}$$

The two first entities represent the premises of the known inputs. The last states that if the multiplier is functioning correctly f will be the product of x and y . The propagation will use the information in the given entities and derive a new one,

$$\langle f = 4, \{M\} \rangle$$

This new entity states that given that the multiplier is fault free, f will be 4. Although not obvious here, the needed supporting environment in the new entity is the union of the needed supporting environments of the entities used to derive the new entity.

Using the procedure outlined above the algorithm propagates through the information. When two contradictive assertions are found a so called nogood is created. A nogood is the union of the two needed supporting environments from the contradicting assertions. The nogoods are used to indicate that not all the members of the nogood set can function at the same time. If we expand our example system with a new component, I , the system will have the outline shown in Figure 8.

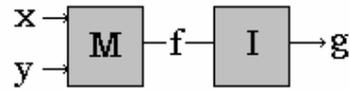


Figure 8. A schematic illustration of the extended multiplier system.

The function of the new component can be described by

$$I(f) = f^{-1} = g .$$

The new set of entities in the ATMS would be,

$$\begin{aligned} \langle x = 1, \{\{\emptyset\}\} \rangle \\ \langle y = 4, \{\{\emptyset\}\} \rangle \\ \langle f = x * y, \{\{M\}\} \rangle \\ \langle g = f^{-1}, \{\{I\}\} \rangle \end{aligned}$$

Measurement of the inputs and outputs of the system gives the premises

$$\begin{aligned} \langle x = 4, \{\{\emptyset\}\} \rangle \\ \langle y = 3, \{\{\emptyset\}\} \rangle \\ \langle g = 10^{-1}, \{\{\emptyset\}\} \rangle \end{aligned}$$

Having this information the GDE starts to propagate through the information deriving the new entities

$$\langle f = 4, \{\{M\}\} \rangle$$

$$\langle g = 4^{-1}, \{\{M, I\}\} \rangle$$

When adding these new tests the GDE will detect the conflicting assertions of g and create a nogood

$$\{M, I\}$$

This state that M and I both can not be functioning correctly. In this case it is fairly easy to see that the diagnosis, the set of mode assignments of the components that could explain the observations, are a fault in the multiplier or a fault in the inverter. In cases where there are several nogoods, it is harder to derive a diagnosis. How this is done is described further in [60].

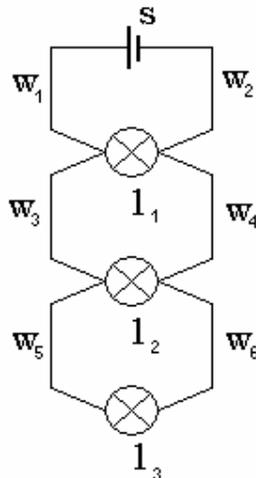


Figure 9. A simple system consisting of three lights, a battery and six wires.

Also worth mentioning is that there are some drawbacks using the GDE, these drawbacks are described in [26]. Consider the system shown in Figure 9. The system is built up of three lights, l_1, l_2, l_3 , a battery, s , and six wires, $w_1, w_2, w_3, w_4, w_5, w_6$, connecting the lights and battery. The observations tell us that only light 3 is lit. GDE would in this particular case generate about twenty possible diagnoses where $\{s, l_3\}$, $\{w_1, w_5\}$ and $\{l_1, l_2\}$ are some of them. In these sets the components included are merely indicated to be faulty, no information about the failure mode is available. The first two sets illustrate the weakness of GDE. In the first candidate the engine concluded that there is something wrong with the battery (it does not create a voltage and thus light one and two does not light) and there is also something wrong with light three since it is lit but there is no voltage. In the second candidate, wire one is

presumed to be cut off and this explains why light one and two are off. Wire five on the other hand is presumed to produce voltage and explains why light three is on. This is from a strictly logical point of view a sound explanation. It explains the observations that is light one and two are off and light three is on. Although it is not a sound explanation from a physical point of view. A light cannot be on without a voltage and a wire cannot produce voltage.

If the models are expanded with information on how the components behave when in a fault mode and we use an inference engine that can utilize from the new information, we can benefit from the knowledge on how a fault affects the component. The inference engine GDE+ is able to utilize this failure mode information. With it we can exclude a number of candidates and speed the diagnosis up.

2.2 XML

Extensible Markup Language, (XML), is a free standard that is used to structure data that can be spreadsheets, address lists, financial transactions, or blueprints. It is a framework for how to construct textual formats with which the data is structured. With XML data is easily generated and read and data structures are guaranteed to be univocal. XML is platform independent and it is extensible, which means that it allows for new formats to be created [12].

In XML there are elements, tags, and attributes. An element must be enclosed by an opening tag and a closing tag. The opening tag is preceded by the character '<' and followed by the character '>'. The same is true for the closing tag although the tag name is preceded by the character '/'. I.e the syntax looks like this [13]:

```
<TagName> ElementContent </TagName>
```

An element can also have attributes, which have the syntax: AttributeName="value". Elements can be nested so that they get a child/parent relationship [13].

A simple XML code example could look like this:

```
<carcollection>
  <car color="blue">
    <make>Toyota</make>
    <origin>Japan</origin>
  </car>
  <car color="red">
    <make>Ferrari</make>
    <origin>Italy</origin>
  </car>
</ carcollection >
```

XML is used for [15]:

- Information identification
- Information storage
- Information structuring
- Publishing
- Messaging and data transferring
- Web services

2.3 Document Type Definition

A Document Type Definition (DTD) decides the structure of a class [19] of XML documents by listing the allowed elements and attributes [18]. With a DTD an XML file carries with it a description of its format and different users can exchange data with this agreed format [18]. In order to give an example of a DTD we consider the following XML file, which is taken from [72].

```
<?xml version="1.0"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

This XML file describes a note that is sent from Tove to Jani, has the heading *Reminder*, and the message in the note is: *Don't forget me this weekend!*. A DTD for describing such an XML file can look as follows. This DTD is also taken from [72].

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

In this DTD file we can see that on the first line defines the *note* element to have the four child elements *to*, *from*, *heading*, and *body*. The remaining four lines define each of these four to be of the type *#PCDATA* [72].

2.4 XML Schema

The XML Schema is an alternative to DTD, which was described in the above section, for describing the structure of an XML document [70]. Schemas are, unlike DTDs, XML based, i.e. the schemas are themselves written in XML. A Schema defines what elements and attributes that are allowed to appear in an XML document and also defines data types for elements and attributes. In an XML Schema we can also specify numeric ranges for a certain element type or specify a list of possible values for the element type.

To show an example of an XML Schema we use the XML file *note* from Section 2.3. Also this example of an XML Schema is taken from [72]:

```
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In which we can see that the *note* element is a complex type because it contains other elements whereas the elements *to*, *from*, *heading*, and *body* are elements of what is called simple types as they do not contain other elements. We also can see in the example that the four simple type elements are specified to be of the data type string.

2.5 Relational Databases

A relational database can be described as a collection of tables that relate to one another. The horizontal rows are called records or tuples and the vertical columns are called fields or attributes [30]. In Figure 10 a simple example can be seen.

Poet			
Code	First Name	Surname	Age
1	Mongane	Afrika	62
2	Stephen	Serote	58
3	Tatumkhulu	Watson	29

Poem	
Title	Poet
Wakening Night	1
Thrones of Darkness	2
Once	3

Figure 10. Two tables in a relational database [30].

The two tables in Figure 10 relate to one another through the Code field in the Poet table and the Poet field in the Poem table.

2.6 AI-ESTATE Standard

Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE) is IEEE standard 1232. This standard deals with the application of artificial intelligence to system test and diagnosis [25]. It also describes how to exchange diagnostic information between diagnostic reasoners by defining a set of formal data and knowledge specifications consisting of the logical representation of devices, their constituents, the failure modes of those constituents, and tests of those constituents.

The specified goals in the standard are [25]:

- Incorporate domain specific terminology
- Facilitate portability of diagnostic knowledge
- Permit extensibility of diagnostic knowledge
- Enable the consistent exchange and integration of diagnostic capabilities

AI-ESTATE is extensible and allows for the user to include new diagnostic technology, which is not specified within AI-ESTATE, in a controlled manner. The intent of the standard is for diagnostic knowledge to be fully portable and not to be host computer dependent. Two different AI-ESTATE implementations can interchange diagnostic models by translating them into the AI-ESTATE interchange format. Although an implementation can use the interchange format for its own internal form, in which case the translation is not needed.

The diagnostic data and knowledge in an AI-ESTATE implementation is structured as can be seen in Figure 11.

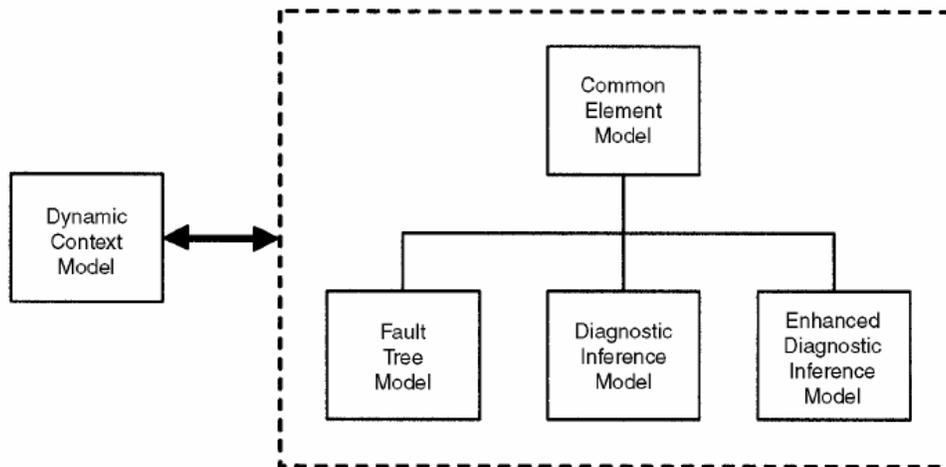


Figure 11. Hierarchical structure of AI-ESTATE models [25].

The top layer in Figure 11 is the Common Element Model that specifies elements common to the AI-ESTATE domain of equipment test and diagnosis e.g. diagnosis, repair_item, resource, and test [25]. The layer below consists of application specific data and knowledge formats i.e. the Diagnostic Inference Model, the Enhanced Diagnostic Inference Model, and the Fault Tree Model. In [27] one can read that AI-ESTATE will be extended to include a model to cover Bayesian diagnosis.

To the left in Figure 11 the Dynamic Context Model (DCM) can be seen. The DCM enables important functions in a diagnostic reasoner that is conformant to AI-ESTATE. These are functions that regard the state of the reasoning process and the data in the DCM are developed during a diagnostic session [25]. The models in AI-ESTATE, e.g. the Common Element Model in Figure 11, are defined with the EXPRESS data modelling language which is ISO standard 10303-11 [25].

3 Survey of Market for Diagnostic Tools

This chapter presents the result of a survey of the market for tools for diagnostic analysis and construction. The process of finding the tools that are to be evaluated is described. For clarifying what is searched for, features not directly connected to diagnostics are described in Section 3.1. In Section 3.2 features are listed, concerning diagnostics desired for a tool in this thesis. The features in 3.2 serve as a guide in the searching process; if a tool that is found possesses one or more of these features it is of interest for further studies. With the desired features and non-diagnostic desired features in mind during the search process, the search resulted in a list of tools which is presented Appendix B. The tools that are finally chosen for evaluation in this thesis are presented and motivations for made choices are explained. The chapter is rounded off with a presentation of a few tools that were close to being selected but did not make it.

3.1 *Non-diagnostic Desired Features*

When using a tool to design, develop, and maintain a diagnostic system there are some aspects needed to be considered besides the technical capabilities. The users will use the tool to formalize their knowledge about the system into a form specified by the tool. To do this, it is preferable if the tool is similar to operate and is compatible with the other tools they use. The tool is also going to be used by a number of persons probably during an extended time. This causes aspects of user friendliness, compatibility and the way to work with the tool to become important.

When dealing with a large system it is also preferable if the tool supports a modularized system construction. If the subsystem codes could be used as modules in the whole system, that would simplify the merging of different subsystems into a whole product.

3.2 *Basic Desired Features*

To point the search for tools in the right direction, twelve features that are desired were formulated. The tools should possess at least one of these features to qualify as an alternative for further studies. The features are motivated and described below.

- 1) A tool that is used for building a model based diagnostic system must include a representation, a model, of the system under consideration to be able to perform diagnostics on that system. The tool must therefore be able to represent a mechanical, electrical, logical or hydraulic system as a model.
- 2) To be able to design and modify the system eases the construction of a diagnostic system. Compared to being forced to import new system designs this saves a lot of time. It is therefore desirable if the tool also can be used to design a mechanical, electrical, logical or hydraulic system.
- 3) In a scenario where different programs are needed to complete the construction of a diagnostic system the need for programs able to convert between different formats may arise. Different formats may for instance be needed for creating the

diagnostic system and running the diagnostic system online; thus a tool that can be used to convert or transfer information between other programs which are considered as alternatives might be wanted.

- 4) The usability of the diagnostic system can be increased if faults can be simulated in the tool. Faults of great criticality or faults known to often occur can with this feature be extensively examined. To simulate faults in the system at hand using the tool is therefore desirable.
- 5) Tracing the effects of a fault; that is to be able to foresee what consequences a particular fault might have on the system behaviour and sensor readings is a necessity if one wishes to avoid testing every fault individually at its source. Also to be able to use an inference algorithm in a model based diagnostic system the ability to trace a fault is necessary. Thus automatically tracing the fault effects and describe them using the tool is a desired feature.
- 6) The different effects upon tests of faults and failure modes in the system must be mapped to be able to perform isolation of the fault. The program must therefore be able to generate fault detection and fault isolation logics from the information fed into the tool.
- 7) The diagnosability of a system can be increased in various amounts if more sensors are added. The amount of increase is affected by where the sensors are placed in the system. Thus it is preferable if the system could evaluate the degree of diagnosability, the detectability and the isolation ability of the system. To further increase the usability of the tool it is desired that the tool could suggest places suitable for new sensors which maximizes the diagnosability.
- 8) In cases where the tool also can generate test formulas and procedures those tests should be easily accessed. Thus a file which contains algorithms for calculating test values should be generated with the tool upon request.
- 9) A generation of a representation of the detection and isolation of faults would ease a transferral to another tool or an on board computer running diagnostics online. The tool thus benefits from being able to generate a code or other file which represents the fault detection and fault isolation logics.
- 10) To fully utilize the diagnostic system it is preferable to be able to attach information related to the different faults. The information could include limitations of operation modes or repair instructions related to the present fault. This information should not necessarily be limited to text but could also include e.g. video. Thus a desired feature is that the tool is able to generate a file which represents the information concerning a detected fault.
- 11) It is a nontrivial task to generate suitable test quantities suitable for the best diagnostic capabilities. Therefore it is desirable if the tool could automatically suggest combinations of inputs and sensor values that would be suitable to use as test quantities.

- 12) Using the same tool for improving the design of a system, running online and offline diagnosis eases the ability to reach fast results. Although the main objective here is to generate an as good diagnostic system as possible and if this requires separate software solutions the diagnostic capabilities are prioritized before speed. Thus the final desired feature is that at least one of online and offline diagnostics and the design of the diagnostic system are treated.

3.3 The Chosen Tools

The tools found in this thesis are listed in Appendix B. After the demands and wishes in Section 3.4 have been considered three tools were finally selected for further investigation. None of these three tools fulfilled all the desired features in Section 3.2 but no tool on the list in Appendix B did. All of the selected tools missed the desired features 8) and 11). This because none of the tools deals with modelling at such a physical level that these demands could have been met. This slightly limits the usability of the tool but it does also simplify modelling and thus compensates for the loss of usability. The fact that these tools all lacked the same features would also make them more comparable than other tools. The selected tools are briefly described in the following sections.

3.3.1 TEAMS from Qualtech Systems Inc

The American company Qualtech's product Testability, Engineering And Maintenance System (TEAMS) is a program package that consists of several software components compatible with each other and make up what they claim to be a complete health management solution.

One of the components is TEAMS Designer which is used to model failures dependencies in the system at hand, meaning what different failures affect which different components. The failures can also be linked to tests, repair procedures and troubleshooting steps. The model captures the interconnections between systems failures, built in tests, and components. The tool can after a model has been given be used to develop a diagnostic workflow optimized for e.g. fast isolation. The tool can also for example suggest test points better suitable for isolation [35].

Another component in the software suite is TEAMS Real Time (TEAMS RT). Fed with a simplified version of the model the TEAMS RT works on the systems on board computer and examines built in test results. This makes TEAMS RT provide online diagnostics for the system [36].

Also other components are available, such as TEAM Automatic Test Equipment (TEAMATE) and Remote Diagnostic Server (RDS). TEAMATE generates a dynamic response to data and test results entered by a service technician. The tool interact with the service technician to aid him or her to perform the fastest sequence of test for isolating the faulty component or components and it also provide test and repair instructions as the technician goes along [38].

RDS is used to remotely health manage a fleet of vehicles or in support of an aftermarket service contract. By sending sensor and test data to the server over the internet or a local network it is automatically processed in the server and the server returns a web page containing tests as help for isolating the fault and test procedures.

All the data and the process are stored in the server for future maintenance or further analysis [37].

TEAMS is analysed in Section 4.1 where it will be clear that it fulfils the criteria stated in Section 3.2 except 8) and 11) and also fits into the description of desired tools in Section 3.4.

3.3.2 eXpress from DSI

eXpress made by the American company DSI International is an off-the-shelf tool suite that encompasses diagnostics, testability, prognostics and systems engineering and is the result of more than 30 years of experience of software development for the company [40]. Rather than focusing on run-time aspects eXpress can be used in the earliest phases of designing a system [41]. An important advantage for eXpress, according to DSI, is that it uses Hybrid Diagnostic Modelling (HDM) [2]. HDM allows for representation of relationships between tests and functions as well as between tests and failure modes in the system.

The modelling is explained further in Section 4.2 together with other aspects of the program that will show that eXpress has all of the desired features discussed in Section 3.4 and the criteria in Section 3.2 except 8) and 11) were fulfilled.

3.3.3 Raz'r from OCC'M

Raz'r is a tool from the German company OCC'M for building diagnostics systems. They base their tool on model based diagnostics and use qualitative models. Raz'r provides a graphical interface for constructing the models and performing the analysis of the system. Capability for increasing the testability of the current system and generation of test trees for fault isolation is also offered. The tool also offers both on board and off board diagnosis, FMEA and detectability analysis [29].

The analysis of Raz'r is found in Section 4.3 and will tell that the wanted attributes listed in Section 3.4 and Section 3.2 except 8) and 11) is present in the tool.

3.4 Additional Motivations for Choices of Tools

The three tools in Section 3.3 that advanced from the list of tools in Appendix B to the final evaluation did so based on a number of criteria that are listed below.

3.4.1 Scope of Use

The desired features in Section 3.2 are regarded for the final choices of tools. What are the tools in Appendix B really capable of? It is important that a tool that is going to be evaluated is capable of doing what is desired. Some of them turned out, at a second review, not to do at all what was thought at first.

Examples of tools that are disqualified because of this criterion are Dexter from Maceea that turned out to be specialised in marine use and Model Wizard from Integrated Systems Diagnostics that turned out to focus more on finance.

3.4.2 Software Based

As can be seen in the prerequisites, Section 1.3, only pure software is of interest in this thesis. Many systems that are found do what the sought after software is intended

to do, but are hardware with integrated software. These systems may be very interesting but are sifted out. This is because Saab is not interested in new systems with hardware. A hardware dependent solution would require a larger work load to integrate into the already existing products.

Two found systems that include hardware are: HUMS from GE and IHUMS from Meggit Avionics.

3.4.3 Available Information

One criterion is how easy it is to find information about the tool and if it is possible to get hold of the tool itself. It is very important that there is enough information that can serve as a good foundation for the evaluation.

For instance, SIDIS from Siemens did not fulfil this criterion.

3.4.4 Named Product

There also has to be a named product that is offered on the market. Some companies describe that they are capable of supplying services that could be of interest but there is not a specific tool to be evaluated. In order to be able to evaluate a tool it is important that there is a clear tool that can be the subject of the evaluation.

One found company, Foster Miller, does not mention a named product which is why it was sifted out.

3.4.5 Clear Overall Solution

It is desired that there is a clear overall solution to be evaluated. If a company provides too many small modules that are specialized in specific parts of diagnostic processes this company's solution did not make it to the final evaluation. In line with the previous section this is to make the evaluation feasible.

Impact Technologies, although mentioned in Section 3.5 below as a strong candidate, did not make it due to this criterion.

3.4.6 Theoretical Foundation

Since there are a number of different approaches solving the diagnostic problem there are also a number of different theories. The approach in this thesis is the model based one, as can be seen in the prerequisites, Section 1.3.

Spotlight from Casebank is an example of a tool that is not model based why it did not qualify for further exploration.

3.4.7 Saab's Current Knowledge

The aim of this thesis is to widen Saab's knowledge about available tools. If Saab already is familiar with a tool [28], that one in particular may be put aside in favour of another tool that Saab does not have as much knowledge about. Rodon from Sörman was not chosen due to this criterion.

3.5 Strong Candidates that did not make the Cut

This section lists some candidates that were close to being in the final evaluation but did not make it all the way. The tools listed here are regarded as strong candidates because they fulfil the desired features in Section 3.2 to basically the same extent as the chosen tools and are in the right scope of use. These tools could have been among the ones that in the end were chosen, although there are reasons for them not being so and those reasons are given for each individual tool below.

3.5.1 Rodon from Sörman

As shown in Section 1.3 Rodon is one tool that would be very interesting in this thesis if it had not been for reasons given in Section 3.4.7. There already is a thesis written on Rodon [28]. Therefore Rodon is not one of the tools that are researched further.

3.5.2 Numerous Different Tools from Impact Technologies

Impact Technologies is an interesting candidate but was not selected due to criteria given in Section 3.4.5. They offer a range of products and it would have been difficult to evaluate them as a whole solution. Impact Technologies' Reason Pro is mentioned though as a suitable complement to eXpress [1].

3.5.3 Diagnostic Profiler & Diagnostician from VSE

Diagnostic Profiler & Diagnostician from VSE was another strong candidate. Diagnostic Profiler & Diagnostician is a model based diagnostic reasoning tool set [34]. Diagnostic Profiler is a development tool used to develop a model base for the Diagnostician which is a dynamic run-time expert system. There are also prognostic features in a tool called Prognostic Framework. Diagnostic Profiler & Diagnostician did not make it all the way due to Section 3.4.3; there was not that much information to be found. However the tool have been mentioned in several papers e.g. [76].

3.5.4 Livingstone 2

Livingstone 2 is an implemented kernel for model based system. It is developed by NASA Ames Research Center. With this kernel faults are not only detected and isolated but are also automatically corrected. The kernel is given a model of the system and then monitors the disparities between from the model calculated behaviour and the observed behaviour of the actual system. From this comparison the Livingstone software calculates recommended actions to be taken to isolate and compensate for a failure in a system component if such is detected. Livingstone have been successfully tested on the International Space Station (ISS), X-34 propulsion system (PITEX) and participated in flight experiments on the Earth Observing Satellite. Although a promising alternative, Livingstone 2 is only a research project and not a product supported by a company. This means that no support or development is available which is important to Saab [39].

3.6 Summary

The search for suitable tools revealed a much greater flora of software dealing with diagnostics than expected. Although the approaches and areas of usage differed a lot from manufacturer to manufacturer they all seemed suitable for their respective intended tasks. The next step is to examine the three finally selected tools, TEAMS, Raz'r, and eXpress, further and more in depth to see if their respective solutions

concerning choices of model types, data formats and inference algorithms results in benefits or limitations to the diagnostic systems. In the next chapter the three individual tools are presented more thoroughly and in Chapter 5 they are compared.

4 Analysis of the Tools

In this chapter the three tools that in the previous chapter were chosen for evaluation are analyzed and these are: TEAMS from Qualtech Systems Inc (QSI), eXpress from DSI, and Raz'r from OCC'M.

Each tool is described in terms of eight sections which are the same for all three tools and the headings of these sections are Representation of knowledge, Methods for fault detection and isolation, Formats for storing data, Design for testability aids, Modifiability, Compatibleness, Libraries, and Disciplines of modeling.

Representation of knowledge describe in what way knowledge about the system is represented in models in the tools. In *Methods for fault detection and isolation* it is described what kind of diagnostic techniques for detecting and isolating faults that are used in the tools. *Formats for storing data* describe what kind of data formats that are used in the tools to store knowledge of the models in the systems. *Design for testability aids* describe if there are some kind of aids for evaluating and improving the testability of a system. For a tool to be efficient to use, it is of interest that models that are created in a tool can be changed and updated as easy as possible and this is described in *Modifiability*. In *Compatibleness* the compatibleness to other software and tools is discussed. What file formats can be imported and exported? With what other software can the tool interact? In *Libraries* it is discussed what kind of facilities there are for storing models. Do the tools come with some kind of basic library of components or preconstructed models? *Disciplines of modelling* reports in which disciplines modelling can be done in respective tool, i.e. if modelling can be done in electrics, mechanics, hydraulics etc.

TEAMS and eXpress have in addition to these eight sections a ninth section called *Test sequencing algorithms*. These sections describe options available in the tools that allows the user to influence in what order tests are to be performed, depending on what kind of results that are desired. There is not a chapter like this in the analysis of Raz'r because no information about this matter has been found in the case of Raz'r.

In addition to what has been described above, three sample systems have been assembled which were sent to the each of the three companies that are producing the tools. These systems, as they were sent to the companies, can be seen in Appendix C. The idea was that QSI, eXpress, and OCC'M would model these systems and show how they are diagnosed with respective tool. Only eXpress and OCC'M did this and what was received from them gave some insight to how eXpress and Raz'r work. Since the material received from eXpress and the material received from Raz'r are quite dissimilar and since no material was received from QSI no further analysis or comparison is made.

4.1 TEAMS

In this section an analysis of TEAMS is made in terms of the items in the list above. It is recommended to return to Section 3.3.1 for a short introduction of the tool.

4.1.1 Representation of Knowledge

In this section the multi signal model is introduced by modelling a known benchmark system called a polybox as a multi signal model. See Figure 12. The following section expands the model by introducing general and functional failures in an amplifier and filter circuit. The information in these sections originates from [16]. The complete multi signal model also contains information about probabilities of failures, cost and time for repair and replacement procedures. These parameters are left out in this description though since they do not add to the detectability or the isolation ability of the system. They will on the other hand be described more in detail in Section 4.1.3.

When QSI uses what they call multi signal models to represent the current knowledge of a system they use the word signal in a somewhat confusing way. They do not refer to signals in the usual sense but rather refer to properties of the system. As an example we consider a gearbox; a property that could be used as a signal (in the multi signal model sense) would be the exchange between momentum into the gearbox and out of the gearbox. In a voltage divider the ratio between input voltage and output voltage could, in QSI terminology, be a signal (again in the multi signal model sense). Thus properties or parameters would have been a more appropriate name than signal.

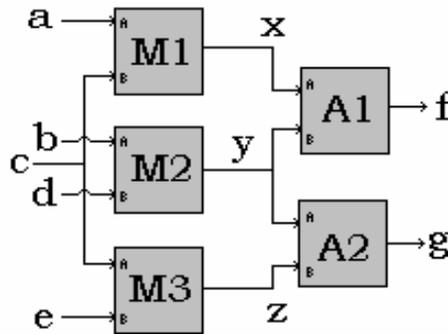


Figure 12. The polybox example.

The polybox system, see Figure 12, is first introduced and then modelled to introduce and illustrate the multi signal model type.

The polybox consists of three multipliers and two adders, connected as shown in Figure 12. The multipliers all function in the same way. They multiply the input from port “A”, the upper input port, and the input from port “B”, the lower port, and the result is passed on to the multipliers output port to the right in each multiplier. The outputs of the multipliers are the inputs to the adders. The adders have the function of adding the inputs from their input ports “A” and “B” and sending the result to the output port to the right in the adders.

The input to the system are the signals $a, b, c, d,$ and e and the outputs are f and g . In between there are three, to us unknown, internal variables, $x, y,$ and z .

To model the polybox as a multi signal model we first create a set

$$\mathbf{C} = \{m_1, m_2, m_3, a_1, a_2\}$$

In this set the instances are the adders and multipliers the polybox consists of. Also signals are defined; these are in this example rather abstract but can be listed as follows:

$$\begin{aligned} s_1 &= \text{the signal "correct calculations in } m_1 \text{"} \\ s_2 &= \text{the signal "correct calculations in } m_2 \text{"} \\ s_3 &= \text{the signal "correct calculations in } m_3 \text{"} \\ s_4 &= \text{the signal "correct calculations in } a_1 \text{"} \\ s_5 &= \text{the signal "correct calculations in } a_2 \text{"} \end{aligned}$$

The signals are collected in the set

$$\mathbf{S} = \{s_1, s_2, s_3, s_4, s_5\}.$$

The tests that are available in this example we can collect in the set

$$\mathbf{T} = \{t_1, t_2\},$$

where t_1 is the test if output f is correct (given the current input) and t_2 is the test if output g is correct. The tests can be expressed as,

$$t_1 : f = ac + bd \quad t_2 : g = bd + ce$$

Another test is also possible to construct,

$$t_3 : f - ac = g - ce$$

but it is not made available to us in this example.

The tests are grouped together in test points which represent a point in the system where the tests are performed. The test points are in turn all instances of the set

$$\mathbf{TP} = \{TP_f, TP_g\}.$$

In this example the test points contain the following tests,

$$\mathbf{SP}(TP_f) = \{t_1\}$$

and

$$\mathbf{SP}(TP_g) = \{t_2\}.$$

Each test check certain signals, these sets of signals are denoted:

$$\begin{aligned}\mathbf{ST}(t_1) &= \{s_1, s_2, s_4\} \\ \mathbf{ST}(t_2) &= \{s_2, s_3, s_5\}\end{aligned}\quad \mathbf{Eq\ 4-I}$$

The next step is to deduce which components that affect each signal. In this case it is fairly simple to see what component that affects which signal since each component is connected to one signal. Although the component also affects the signals connected to components after it. Thus one must follow the signal path through the system. From these connections a set for each component is constructed,

$$\begin{aligned}\mathbf{SC}(m_1) &= \{s_1, s_4\} \\ \mathbf{SC}(m_2) &= \{s_2, s_4, s_5\} \\ \mathbf{SC}(m_3) &= \{s_3, s_5\} \\ \mathbf{SC}(a_1) &= \{s_4\} \\ \mathbf{SC}(a_2) &= \{s_5\}\end{aligned}\quad \mathbf{Eq\ 4-II}$$

The interpretation of the first set is that the signals affected by component m_1 is s_1 and s_4 .

A bipartite graph is constructed to formalize the relations between the components and the test points,

$$\mathbf{BG} = \{\mathbf{C}, \mathbf{TP}, \mathbf{E}\}$$

The first two sets have been explained earlier. The new one, \mathbf{E} , consists of the directed edges specifying the structural connectivity of the system, i.e. for the example

$$\mathbf{E} = \{(m_1, TP_f), (m_2, TP_f), (m_2, TP_g), (m_3, TP_g), (a_1, TP_f), (a_2, TP_g)\}$$

The edges specify at what test point failures in different components can be detected. Thus each edge starts at a component and ends at a test point. Observe that failures in some components are detectable at several test points. A failure in m_2 for instance is observable at both TP_f and TP_g .

From the bipartite graph we can now collocate the information about dependencies in a dependency matrix, shown in Table 3. The dependency matrix is constructed from the sets

$$\mathbf{SC}(x) \quad x \in \mathbf{C}$$

where every row in the matrix corresponds to a component. If the component affects the signal corresponding to a column there will be a "1" in that point in the matrix, if not there will be a "0". We can also derive the relation back to certain test points.

Table 3. The dependency matrix for the polybox example.

	TP _f			TP _g		
	s ₁	s ₂	s ₄	s ₂	s ₃	s ₅
m ₁	1	0	1	0	0	0
m ₂	0	1	1	1	0	1
m ₃	0	0	0	0	1	1
a ₁	0	0	1	0	0	0
a ₂	0	0	0	0	0	1

After we have listed the components, the signals, the test points, and the tests and also described the relations between them, we now have the fault detection and isolation part of a multi signal model of the polybox [16].

To further exemplify the model type we now consider a less theoretical example, an electrical circuit consisting of a number of resistors, operational amplifiers, and a capacitor. The circuit can be divided into three parts, first an amplifier of gain 2 followed by a low pass filter and a buffer. See Figure 13 for a schematic picture of the system. The amplifier starts at the left and ends after the first operational amplifier. The resistor and capacitor make up the filter and the rest to the right is the buffer. Each of the three parts is made up of components, in this case resistors, a capacitor, and operational amplifiers.

To each of the components a number of signals (again in the multi signal model sense) can be associated as we did in the first example. Signals that carry information of different kind of how the component functions.

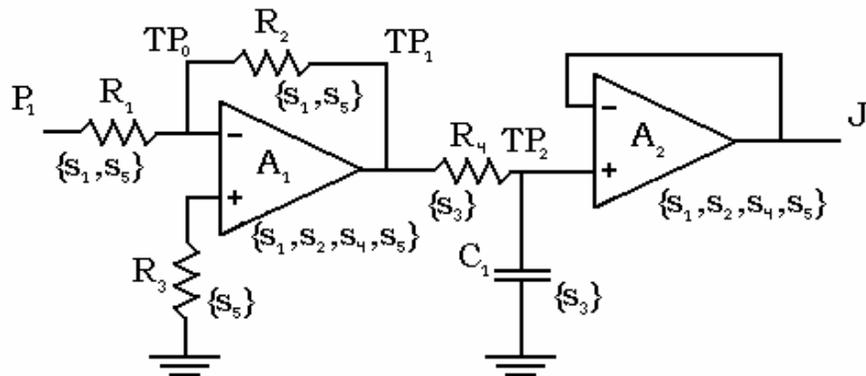


Figure 13. Schematic illustration of an amplifier/filter marked with test points and affected signals [16].

In this example the signals concerned by each component are listed close to that component. The signals are explained in Table 4, the components affecting them are also listed.

Table 4. Description of the signals and their dependencies.

Signal	Description	Concerned by components
s_1	Gain	R_1, R_2, A_1, A_2
s_2	Linearity	A_1, A_2
s_3	Cut off frequency	R_4, C_1
s_4	Slew rate	A_1, A_2
s_5	DC offset	R_1, R_2, R_3, A_1, A_2

The information in Table 4 can be translated into the same sets as we used in the polybox example:

$$\begin{aligned}
\mathbf{C} &= \{R_1, R_2, R_3, R_4, C_1, A_1, A_2\} \\
\mathbf{S} &= \{s_1, s_2, s_3, s_4, s_5\} \\
\mathbf{SC}(R_1) &= \{s_1, s_5\} \quad \mathbf{SC}(R_2) = \{s_1, s_5\} \quad \mathbf{SC}(R_3) = \{s_5\} \quad \mathbf{SC}(R_4) = \{s_3\} \\
\mathbf{SC}(C_1) &= \{s_3\} \quad \mathbf{SC}(A_1) = \{s_1, s_2, s_4, s_5\} \quad \mathbf{SC}(A_2) = \{s_1, s_2, s_4, s_5\}
\end{aligned}$$

For example the cut off frequency is set by the resistor R_4 and the capacitance C_1 and these two components do thus affect the signal s_3 . The other signal dependencies are deduced in the same manner. These methods for deriving tests are part of the physics of the system and are not treated in the tool other than that the test procedures, which are entered manually into the program, can be associated with the tests they describe. The test procedures will thus not be produced by the program.

To each of the test points certain tests are connected. The test points for the sample system are also marked in Figure 13. These points specify where in the system it is possible to perform tests and which test we can perform at that point.

The sets concerning tests and test points are listed below:

$$\begin{aligned}
\mathbf{TP} &= \{P_1, TP_0, TP_1, TP_2, J_1\} \\
\mathbf{T} &= \{t_1, t_2, t_3, t_4, t_5\} \\
\mathbf{ST}(t_x) &= \{s_x\} \quad \forall x \in \{1, 2, 3, 4, 5\}
\end{aligned}$$

The last row specifies the somewhat simplified relation that each signal corresponds only to one single test and that all signals are covered.

An example of a test is how to test s_5 ; apply a known signal at P_1 and measure the DC voltage at TP_1, TP_2 and J_1 . This gives a connection between s_5 and all the test points, thus test 1 is to be associated to all the test points. The same pass for the signal s_1 . To test the slew rate, s_4 , a high frequency and high amplitude sine signal, $S(t)$, have to be applied at P_1 and then we have to observe the derivative of sine when $S(t) = 0$. This test could be performed at all the test points.

For the sake of simplicity we test only s_3 at TP_2 and the other signals at TP_1 and J_1 . This gives the sets:

$$SP(TP_1) = \{t_1, t_2, t_4, t_5\} \quad SP(TP_2) = \{t_3\} \quad SP(J_1) = \{t_1, t_2, t_4, t_5\}$$

If we would like to change the specifications of the systems behaviour we merely add another signal specifying the new property and attach it to the concerned components and test points.

The connections between components, test points and signals are summarized in Table 5 where $a_{ij}=1$ symbolizes that there is a connection between component i and signal j and $a_{ij}=0$ means that there is not. P_1 and TP_0 have been left out since they are not associated with any tests.

Table 5. Dependency matrix for the sample system.

	TP ₁				TP ₂	J ₁			
	s ₁	s ₂	s ₄	s ₅	s ₃	s ₁	s ₂	s ₄	s ₅
R ₁ (G)	1	1	1	1	1	1	1	1	1
R ₁ (F)	1	0	0	1	0	1	0	0	1
R ₂ (G)	1	1	1	1	1	1	1	1	1
R ₂ (F)	1	0	0	1	0	1	0	0	1
R ₃ (G)	1	1	1	1	1	1	1	1	1
R ₃ (F)	0	0	0	1	0	0	0	0	1
A ₁ (G)	1	1	1	1	1	1	1	1	1
A ₁ (F)	1	1	1	1	0	1	1	1	1
R ₄ (G)	0	0	0	0	1	1	1	1	1
R ₄ (F)	0	0	0	0	1	0	0	0	0
C ₁ (G)	0	0	0	0	1	1	1	1	1
C ₁ (F)	0	0	0	0	1	0	0	0	0
A ₂ (G)	0	0	0	0	0	1	1	1	1
A ₂ (F)	0	0	0	0	0	1	1	1	1

This is also where we see how the general and functional failures come into use. The parenthesis after each of the components indicates whether the failure is a general, (G), or a functional, (F), one. A general failure of a component makes it stop performing its intended task completely whereas a functional failure limits the function of the component but does not cancel it completely. A cut off in a resistor would be considered as a general failure but a reduction of the resistance, e.g. bisection, would be a functional failure. A functional fault in R_4 would only affect s_3 while a general fault would affect all the signals affected by components behind R_4 .

General failures can be said to affect the whole system behind the failed component whilst a functional failure only affects the subset of signals affected by the failed component. The components after the component with the functional failure will still be able to perform their tasks although the input is faulty.

4.1.2 Methods for Fault Detection and Isolation

After modelling the system we want to be able to automatically detect and isolate the failed component. This is done with an inference algorithm. In this section we will go through the algorithm used in TEAMS step by step. The description here is based on information found in [17,74].

The algorithm has a number of inputs, the inputs are in this case test results.

$$\mathbf{T} = \mathbf{T}_f \cup \mathbf{T}_g$$

where \mathbf{T}_f are the set of failed tests and \mathbf{T}_g are the set of passed tests. All, n , tests can be written as

$$\mathbf{T} = \{t_1, \dots, t_i, \dots, t_n\}.$$

The tests must all have been executed during a period in which the system does not change its state. Otherwise the algorithm will not function correctly.

The fault sensitivity of t_i can be denoted as $\mathbf{T}s_i$ and is expressed as

$$\mathbf{T}s_i = \{c \in \mathbf{C} \mid \mathbf{SC}(c) \cap \mathbf{ST}(t_i) \neq \emptyset\}$$

The expression states that components affecting signals that are tested by test t_i are a part of the test sensitivity of that particular test.

Each of the components can be in one of four states: unknown, suspected, bad, or good. If a component is in state suspected it is suspected to be bad but there is not enough evidence that it is bad. If there is enough evidence the component is transferred to the bad state. Components known to be good are transferred to the good state. If there is no knowledge available about a specific component that component is in the unknown state.

The aim of the algorithm is to, from the given test results, $\mathbf{T}_f \cup \mathbf{T}_g$, conclude in which state each of the components are. Each of the components are in exactly one of the states and this can be represented by,

$$\mathbf{C} = \mathbf{U} \cup \mathbf{S} \cup \mathbf{B} \cup \mathbf{G}$$

In this representation the components in state unknown are members of the set \mathbf{U} , the ones in state suspected are members of set \mathbf{S} , the ones in state bad are members of the set \mathbf{B} , and the ones known to be good are members of the set \mathbf{G} .

Next we give a formal description of the fault isolation algorithm using the sets just described. In the algorithm the cardinality of a set A will be denoted $|A|$.

Algorithm I:**Inputs:** A set of failed tests, \mathbf{T}_f , a set of passed tests, \mathbf{T}_g **Outputs:** The sets $\mathbf{U}, \mathbf{S}, \mathbf{B}$, and \mathbf{G} .

1. Initialization: $\mathbf{U} = \mathbf{C}$, $\mathbf{G} = \emptyset$, $\mathbf{S} = \emptyset$, $\mathbf{B} = \emptyset$
2. Compute the good components covered by the passed tests,

$$\mathbf{G} = \bigcup_{t_i \in \mathbf{T}_g} \mathbf{T}s_i, \quad \mathbf{U} = \mathbf{U} \setminus \mathbf{G}$$
3. Compute the suspected components covered by the failed tests,

$$\mathbf{S} = \mathbf{U} \cap \left(\bigcup_{t_i \in \mathbf{T}_f} \mathbf{T}s_i \right), \quad \mathbf{U} = \mathbf{U} \setminus \mathbf{S}$$
4. Compute the bad components,

$$\mathbf{B} = \{c \in \mathbf{C} \mid \exists t_i \in \mathbf{T}_f : |\mathbf{T}s_i \cap \mathbf{S}| = 1\}, \quad \mathbf{S} = \mathbf{S} \setminus \mathbf{B}$$

The algorithm begins with setting all the components in the unknown state \mathbf{U} . It then finds all the components that are affecting test that have passed. Since those tests have passed, the found components must be good and are thus moved to \mathbf{G} . The next step is to find the components remaining in state \mathbf{U} that are affecting tests that have failed. These components are then moved to the suspected state \mathbf{S} . Finally the components moved to the bad state \mathbf{B} are found. If a test is affected by only one component in the suspected mode it is then moved to the bad state. The algorithm is repeated as new test results are available or when the system changes state.

The algorithm will be illustrated with an example using the polybox model derived in Section 4.1.1. Assume there is a fault in component m_3 . This will generate an indication (failure) in test t_2 . This test is associated with $\{s_2, s_3, s_5\}$, as seen in Eq 4-I, and thus $\{m_2, m_3, a_2\}$ will be indicated, but the algorithm also utilizes the information that test t_1 not has failed.

First all components are set to unknown in step 1 of the algorithm:

$$\mathbf{U} = \{m_1, m_2, m_3, a_1, a_2\}$$

and the rest of the sets are empty:

$$\mathbf{B} = \mathbf{S} = \mathbf{G} = \emptyset$$

To refresh our memory some sets belonging to the polybox model are listed:

$$\begin{aligned}\mathbf{ST}(t_1) &= \{s_1, s_2, s_4\} \\ \mathbf{ST}(t_2) &= \{s_2, s_3, s_5\}\end{aligned}\quad \text{Eq 4-I}$$

$$\begin{aligned}\mathbf{SC}(m_1) &= \{s_1, s_4\} \\ \mathbf{SC}(m_2) &= \{s_2, s_4, s_5\} \\ \mathbf{SC}(m_3) &= \{s_3, s_5\} \\ \mathbf{SC}(a_1) &= \{s_4\} \\ \mathbf{SC}(a_2) &= \{s_5\}\end{aligned}\quad \text{Eq 4-II}$$

The sets $\mathbf{T}s_1$ and $\mathbf{T}s_2$ are created:

$$\begin{aligned}\mathbf{T}s_1 &= \{m_1, m_2, a_1\} \\ \mathbf{T}s_2 &= \{m_2, m_3, a_2\}\end{aligned}$$

In this example $\mathbf{T}_f = \{t_1\}$ and $\mathbf{T}_g = \{t_2\}$

The next step, 2, is performed to find the components associated with the passed test, i.e. t_1 . The condition

$$c_x \in \mathbf{T}s_j \quad \text{if} \quad \mathbf{SC}(c_x) \cap \mathbf{ST}(t_j) \neq \emptyset \quad \left\{ \begin{array}{l} \forall c \in \mathbf{C} \\ \forall t \in \mathbf{T}_g \end{array} \right\}$$

gives

$$\mathbf{G} = \{m_1, m_2, a_1\}$$

from

$$\begin{aligned}\mathbf{SC}(m_1) \cap \mathbf{ST}(t_1) &= \{s_1, s_4\} \cap \{s_1, s_2, s_4\} = \{s_1, s_4\} \\ \mathbf{SC}(m_2) \cap \mathbf{ST}(t_1) &= \{s_2, s_4, s_5\} \cap \{s_1, s_2, s_4\} = \{s_2, s_4\} \\ \mathbf{SC}(m_3) \cap \mathbf{ST}(t_1) &= \{s_3, s_5\} \cap \{s_1, s_2, s_4\} = \emptyset \\ \mathbf{SC}(a_1) \cap \mathbf{ST}(t_1) &= \{s_4\} \cap \{s_1, s_2, s_4\} = \{s_4\} \\ \mathbf{SC}(a_2) \cap \mathbf{ST}(t_1) &= \{s_5\} \cap \{s_1, s_2, s_4\} = \emptyset\end{aligned}$$

where the sets can be seen in Eq 4-I and Eq 4-II.

With this new information we can update the sets:

$$\mathbf{G} = \{m_1, m_2, a_1\} \quad \mathbf{U} = \mathbf{U} \setminus \mathbf{G} = \{m_3, a_2\}$$

In step 3 both m_3 and a_2 are covered by the condition for transferral to \mathbf{S} by

$$\mathbf{S} = \mathbf{U} \cap \mathbf{T}s_2 = \{m_3, a_2\} \cap \{m_2, m_3, a_2\} = \{m_3, a_2\}$$

Here, if there would have been another component not covered by any test, that component would have remained in \mathbf{U} .

In the final step, step 4, the algorithm tries to identify bad components. Since a failure in any single one of the components in suspected mode will explain the failed test, none of them can be isolated. Using equations it appears as:

$$|\mathbf{S} \cap \mathbf{T}s_2| = |\{m_3, a_2\} \cap \{m_2, m_3, a_2\}| = |\{m_3, a_2\}| \neq 1$$

We see that the condition is not satisfied and no bad component can be found. This gives the following output from the algorithm,

$$\mathbf{U} = \emptyset, \quad \mathbf{G} = \{m_1, m_2, a_1\}, \quad \mathbf{S} = \{m_3, a_2\}, \quad \mathbf{B} = \emptyset$$

From this we see that the algorithm can not isolate the faulty component, it merely narrows the suspected components down to two. Although if only one failed component would have explained the failed test this component would have been included in the \mathbf{B} set and the following components that also affect tests that have failed would be included in the suspected set, \mathbf{S} . This case will be illustrated in the next section.

If we would like to upgrade the systems isolation ability we could add a test, measurement of z , and update the different matrices and sets accordingly. The addition of a sensor between m_3 and a_2 would increase the ability to isolate the fault in the example above. This due to the fact that it would bring a new test,

$$t_4 : z = c * e$$

The test would be performed at a new test point, TP_z , and would be sensitive to faults in m_3 but not to faults in a_2 . The new sets would be,

$$\begin{aligned} \mathbf{T} &= \{t_1, t_2, t_4\} \\ \mathbf{TP} &= \{TP_f, TP_g, TP_z\} \\ \mathbf{SP}(TP_z) &= t_4 \\ \mathbf{ST}(t_4) &= \{s_3\} \\ \mathbf{T}s_4 &= \{m_3\} \end{aligned}$$

$$\mathbf{E} = \{(m_1, TP_f), (m_2, TP_f), (m_2, TP_g), (m_3, TP_g), (m_3, TP_z), (a_1, TP_f), (a_2, TP_g)\}$$

All these new sets and information can be collapsed into a new dependency matrix shown in Table 6.

Table 6. The dependency matrix for the polybox with the additional test.

	TP _f			TP _z	TP _g		
	s ₁	s ₂	s ₄	s ₃	s ₂	s ₃	s ₅
m ₁	1	0	1	0	0	0	0
m ₂	0	1	1	0	1	0	1
m ₃	0	0	0	1	0	1	1
a ₁	0	0	1	0	0	0	0
a ₂	0	0	0	0	0	0	1

With this new test the last step in the algorithm is

$$|\mathbf{S} \cap \mathbf{T}_{s_2}| = |\{m_3, a_2\} \cap \{m_2, m_3, a_2\}| = |\{m_3, a_2\}| \neq 1$$

$$|\mathbf{S} \cap \mathbf{T}_{s_4}| = |\{m_3, a_2\} \cap \{m_3\}| = |\{m_3\}| = 1$$

Due to this the algorithm will set the state of m_3 to **B** and the output is

$$\mathbf{U} = \emptyset, \quad \mathbf{G} = \{m_1, m_2, a_1\}, \quad \mathbf{S} = \{a_2\}, \quad \mathbf{B} = \{m_3\}$$

The reason that a_2 still is suspected is that there is no test affected by a_2 that have passed, only one that have failed, t_2 .

From this the conclusion can be drawn that given the initial tests the algorithm suspects two components, of which one is the faulty one. If the test set is extended with a suitable test then the algorithm can isolate the faulty component.

4.1.3 Test Sequencing Algorithms

When performing diagnostics there might be different aspects that are considered more interesting than others, sometimes the repair cost might be the most important aspect; other times perhaps the time it takes to isolate the faulty component is the most important aspect. TEAMS uses a test sequencing algorithm to order the tests in a sequence where the different aspects of importance have been accounted for. The algorithm is shown in [16] and will in this section be briefly described.

The algorithm takes as input a number of sets;

$$\mathbf{A} = \{a_0, a_1, \dots, a_m\}$$

lists the m failure sources in the system. It does also include a dummy failure source, a_0 , that denotes the no fault state of the system. Related to this set is the vector

$$\mathbf{P} = (p(a_0), p(a_1), \dots, p(a_m))^T$$

which denotes the probabilities of each of the faults in \mathbf{A} . These probabilities are based on a priori knowledge of the fault occurrences under a single fault assumption

and are also related to the components Mean Time To Failure (MTTF). The set of tests is

$$\mathbf{T} = \{t_1, t_2, \dots, t_n\}$$

where each test is associated with a cost. These costs are listed in,

$$\mathbf{B} = (b_1, b_2, \dots, b_n)^T$$

where b_x states the cost of test t_x . The cost can be in money, estimated criticality of the faults for which the specific test tests, time to perform the test, manpower requirements, or some other factor specified by the user. These different costs can be used to aim the optimization according to the wishes of the users.

The repair costs of a failure source is listed in

$$\mathbf{F} = (f_1, f_2, \dots, f_m)$$

where each failure source a_x has the repair cost f_x . To sequence the tests there must also be connections between the tests and the failure sources. These connections are stated in the dependency matrix, in TEAMS called a diagnostic dictionary. The dictionary is denoted \mathbf{D} and have dimension $m \times n$.

The output is a sequence of ordered tests that minimizes the expected cost. For each set the following criteria is minimized,

$$J = \sum_{l=0}^m \left(f_l + \sum_{j=1}^{|P_l|} b_{P_l[j]} \right) p(a_l)$$

where P_l is a set of ordered indices representing the tests to be executed to isolate the fault a_l and $|P_l|$ is the cardinality of the set. The optimization is made over different sets of P , i.e. the different sets of ordered indices isolating the fault.

The problem of constructing the sets P is a binary identification problem. These problems arise in many applications but are not treated in this report.

4.1.4 Formats for Storing Data

This section describes what kind of formats that are used for storing models in TEAMS. For model storing TEAMS uses both XML files and relational databases. TEAMS-KB, of which we can read more in Section 4.1.8, is an Oracle based database application [63] which is used to manage relational databases. As mentioned, TEAMS also uses XML files for representing TEAMS models and these follow the TEAMS Document Type Definition [21] (DTD, see Section 2.3). To exemplify how such a file can look like, Figure 14 displays an XML file containing a dependency model of an aircraft engine that was modelled in the study in the paper [21]. In this XML file we

can see different faults of a subsystem called Actuator Fuel Pump, for example the fault “drive shaft shears”. There is information about these faults e.g. Mean Time To Failure, (MTTF) repair time, and repair cost that were mentioned in Section 4.1.3.

```

<!--
*****
* SUB   MODULES
*
*****
-->

<SUBMODULE NAME="Actuator_Fuel_Pump">
<NARRATIVE> Actuator_Fuel_Pump </NARRATIVE>
<FAULT NAME="A731510_FAAA" MTTF="5.26316e+007" REPAIRCOST="100" REPAIRTIME="100" RECTCOST="100.0" RECTTIME
<NARRATIVE><![CDATA[Actuator fuel pump -drive shaft shears, excessive drive spline wear, restricted spline
</FAULT>
<FAULT NAME="A731510_FABA" MTTF="5.39811e+006" REPAIRCOST="100" REPAIRTIME="100" RECTCOST="100.0" RECTTIME
<NARRATIVE><![CDATA[Actuator fuel pump -gear journal, face, gear tooth, or spline wear, gear bearing wear.
</FAULT>
<FAULT NAME="A731510_FABB" MTTF="5.39811e+006" REPAIRCOST="100" REPAIRTIME="100" RECTCOST="100.0" RECTTIME
<NARRATIVE><![CDATA[Actuator_Fuel_Pump - Gear Bearing Wear; or Bearing Preload Spring Fracture]]></NARRATI
</FAULT>
<FAULT NAME="A731510_FACA" MTTF="5.54017e+007" REPAIRCOST="100" REPAIRTIME="100" RECTCOST="100.0" RECTTIME
<NARRATIVE><![CDATA[Actuator fuel pump -impeller fractures, boost stage drive shaft shears or excessive sp
</FAULT>
<FAULT NAME="A731510_FADA" MTTF="4.55685e+006" REPAIRCOST="100" REPAIRTIME="100" RECTCOST="100.0" RECTTIME
<NARRATIVE><![CDATA[Actuator fuel pump -impeller worn, inboard thrust bearing worn, thrust balance seal hu
</FAULT>
<FAULT NAME="A731510_FAEA" MTTF="4.21053e+007" REPAIRCOST="100" REPAIRTIME="100" RECTCOST="100.0" RECTTIME
<NARRATIVE><![CDATA[Actuator fuel pump -gear stage relief valve poppet binding open or spring fractured;du
</FAULT>
<FAULT NAME="A731510_FAFa" MTTF="8.09717e+007" REPAIRCOST="100" REPAIRTIME="100" RECTCOST="100.0" RECTTIME
<NARRATIVE><![CDATA[Actuator fuel pump -gear stage relief valve poppet binding closed;due to contamination
</FAULT>
<FAULT NAME="A731510_FAGA" MTTF="9.07441e+006" REPAIRCOST="100" REPAIRTIME="100" RECTCOST="100.0" RECTTIME
<NARRATIVE><![CDATA[Actuator fuel pump -gear housing, volute housing, or relief valve housing fracture, or

```

Figure 14. XML file containing a dependency model in the study in the paper Validation of a COTS EHM Solution for the JSF Program [21].

4.1.5 Design for Testability Aids

TEAMS comes with a number of different aids for evaluating and improving the testability of a system. Tools for generating testability reports with information about mean ambiguity group size, detection rate and isolation rate calculated from the model and inference algorithm are incorporated into the tool. These reports can be used to increase the testability of the system.

There is a special feature to investigate feedback loops detected in the model from the dependencies and the loops affect on the testability. With this feature the tool can suggest where to break the loop, indicate where ambiguity groups are located and where it would be suitable to insert tests to decrease the size of the ambiguity groups and thus increase the testability of the system [35, 43].

4.1.6 Modifiability

To fully benefit from a diagnostic system it must be simple and intuitive to update the models used. QSI described in [45] that they were in the process of developing their Reusable Test and Model Library (RTML) which now is known as TEAMS-KB which will be further described in Section 4.1.8. In this RTML, it is described in [45], entities of a model such as components are represented as objects and are thus reusable. Such reusability reduces cost of model development since new components

will easily be modeled starting at already modeled similar components. As maintenance and repair data become available TEAMS-KB provides capabilities to update component reliability, component failure rate, repair costs, and repair times [42].

4.1.7 Compatibleness

In [21] it is described how data from a Microsoft Excel spreadsheet is automatically imported into an XML file using a data import tool with a Graphical User Interface (GUI). In this import tool the data elements in the Excel spreadsheet are mapped to TEAMS data elements. The XML file complies with the TEAMS Document Type Definition and an example can be seen in Figure 14. It is stated on the homepage of QSI that for importing models they “have an XML import format and have experience importing Matlab/Simulink data” [35]. Another feature is that structural models can be automatically generated in TEAMS from structural models or netlists in VHDL and EDIF [45]. EDIF stands for Electronic Design Interchange Format and is a neutral format for storing and interchanging electronic netlists and schematics. VHDL stands for VHSIC Hardware Description Language and is a language for construction and simulation of electronic systems [71]. VHSIC is an abbreviation of Very High Speed Integrated Circuits.

Compatibleness concerns exporting as well and besides TEAMATE and TEAMS-RT models from TEAMS can also be exported to any database that is ODBC-compliant [64]. ODBC is an abbreviation of Open DataBase Connectivity and is a standardized method for accessing databases regardless of what database management system and operating system is used. Furthermore, according to the TEAMS 9.x user manual TEAMS can generate four kinds of different formatted text files which contain the diagnostic strategy in form of a diagnostic tree and these are [24]:

- Diagnostic strategy in XML format
- Diagnostic strategy in HTML format
- Diagnostic strategy in PDF format
- AI-ESTATE fault tree model.

The AI-ESTATE standard in the list above, which is presented in Section 2.6, is mentioned in papers written by QSI. In it is described that “the AI-ESTATE 1232.1 standard has been in full use and the 1232.2 standard will be a full use in the near future” [23]. This paper was written in 2001 and AI-ESTATE 1232.1 was a trial version, i.e. not the finished standard. Information about to what extent AI-ESTATE is used today has not been found other than, as mentioned above, the fact that an AI-ESTATE fault tree model can be exported from TEAMS.

Figure 15 below displays an example of how a diagnostic tree can look like in TEAMS. The following description of the tree explains how it works when handled in TEAMS.

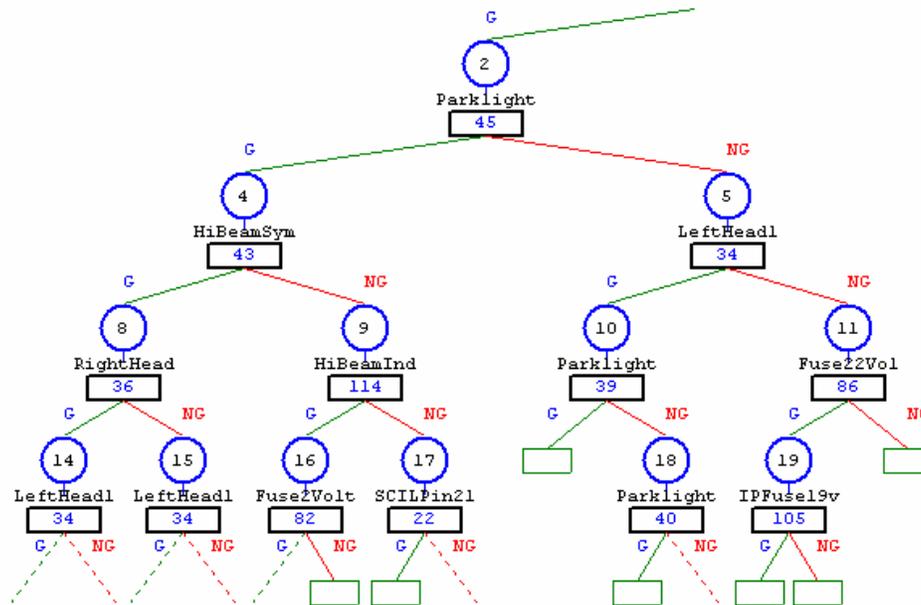


Figure 15. Part of a diagnostic tree in TEAMS [68].

In Figure 15 the circles are nodes, which are the points at which decisions are made in the diagnostic tree. Within the circles are the numbers that are assigned to the nodes, numbers that are listed in the text reports. The rectangles below circles are tests and enclosed within the rectangles are the numbers of the tests. Tests were discussed in Sections 4.1.1, 4.1.2, and 4.1.3. The text between a circle and a rectangle, e.g. *HiBeamSym* between node 4 and test 43, is a shortened version of the name of the test. The G and NG above the circles stand for Go and No Go respectively, i.e. G stands for a passed test at the previous node and NG stands for a failed test at the previous node, i.e. previous in the sense that we traverse the tree downwards. The smaller rectangles at the bottom of the tree that have no numbers inside them represent isolated faults. A question mark within one of these rectangles would represent an ambiguity group, i.e. that there are two or more possible failures. Ambiguity groups were discussed in Section 4.1.5. The dotted lines at the bottom row indicate that the tree is continued and if for example node number 14 is clicked upon there will be another display with another part of the tree in which node number 14 is the root, i.e. at the top of the displayed tree.

There is also a detailed version of the diagnostic tree and an example of this is displayed in Figure 16. This version also includes descriptions of the tests.

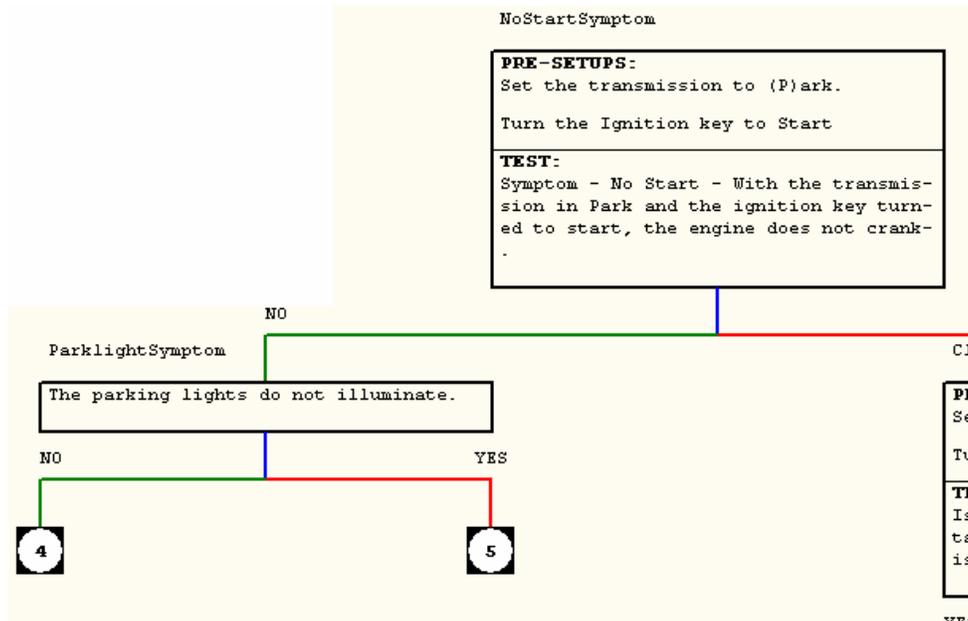


Figure 16. Detailed version of a diagnostic tree in TEAMS [68].

4.1.8 Libraries

One part of the TEAMS software suite is TEAMS-KB (Knowledge Base) [22] which is an Oracle Enterprise database of TEAMS models. System models and their components can be created, modified and stored here along with parameters of the model such as cost, time and failure rate [22]. It also stores media and text of repair procedures, results of diagnostic sessions, and other information that is used by the system to generate procedures and reports. The media mentioned can be photos, videos, drawings etc. TEAMS-KB also stores fault lists and logistic data [42].

In TEAMS-KB online diagnostic and maintenance information that is collected by TEAMATE can specify which components that have been repaired, cost and required time for repair procedures and tests [54].

TEAMS also comes with an example model of an automobile.

4.1.9 Disciplines of Modelling

In TEAMS 9.X Manual it is described that modules can be labelled by different technologies and the ones listed there are: electrical, mechanical, hydraulic and chemical [44].

4.2 eXpress

This chapter deals with eXpress which is analysed in terms of the topics listed in the introduction to Chapter 4. A quick introduction to the tool was given in 3.3.2.

4.2.1 Representation of Knowledge

In eXpress the models are made up by sets of components where each component is constituted by a number of failure modes. A schematic illustration is given in Figure 17. These failure modes represent the known failure modes of that specific component. In each failure mode certain properties are assigned, such as what functions that are affected in the specific failure mode and in what way the functions are affected. Since eXpress does not handle physical modelling, only dependency modelling, the impact of the failure mode on the functions is only modelled as either “does not affect the function”, “does sometimes affect the function”, or “does always affect the function”. The functions are often the starting point in the modelling work. The system has a number of functions that are critical to the system and thus are failure in these extra important to be able to detect and isolate. These functions are specified and the components needed to provide the function as well. The functions and the components are then connected through failure modes. This kind of dependency models with failure modes and functions DSI calls hybrid models and are described in [2].

In the later stages of the system development procedure, failure probabilities might become available and these should also be incorporated into the failure mode to enable calculations of e.g. mean time to isolation or mean time to failure. Where mean time to isolation is the calculated mean time to isolate a fault based on the time it has taken other times when the fault has occurred.

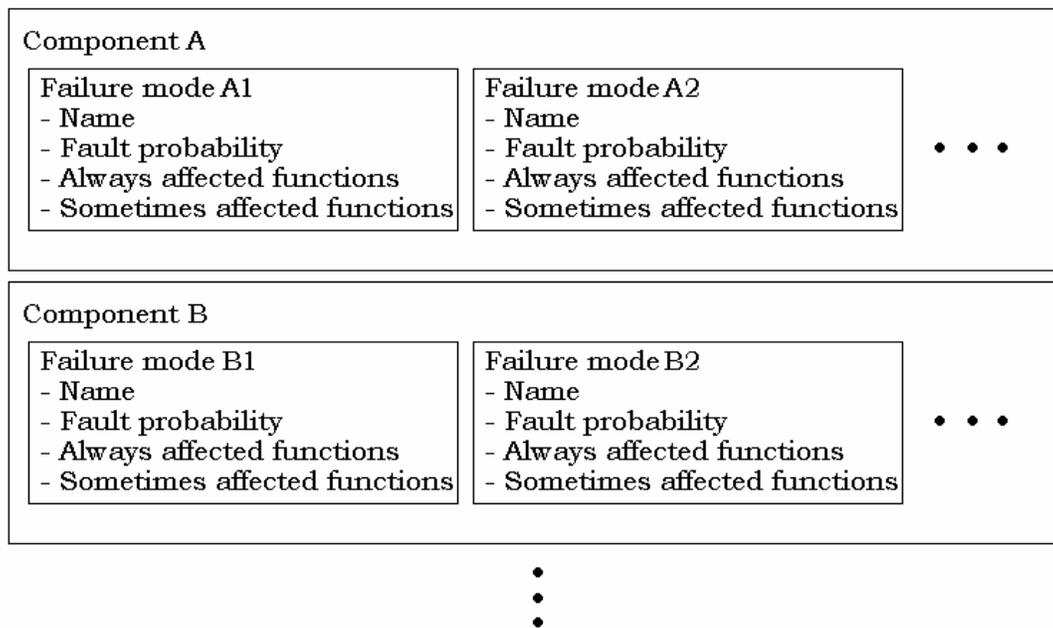


Figure 17. A schematic picture of a hybrid model.

Given the above description we can try to formalize the model by introducing a number of sets representing the entities in the model. Let \mathbf{C} represent the number of components in the model. Each component, $c \in \mathbf{C}$, is associated with its respective set of failure modes, FM_{c_j} . These failure modes are uniquely related to their respective components;

$$FM_{c_j} = \{fm_1, fm_2, \dots, fm_k\}$$

where FM_{c_j} represents the set of all failure modes of component c_j . Each failure mode, fm , defines two sets of functions. One set,

$$\mathbf{AA}_{fm} = \{f_{a1}, f_{a2}, \dots, f_{ai}\}$$

whose functions are *always affected* by that fm . The other set of functions,

$$\mathbf{SA}_{fm} = \{f_{s1}, f_{s2}, \dots, f_{sn}\}$$

contains the functions that are *sometimes affected* by the fm . This is illustrated in Figure 18. In the figure the solid lines represents the always affects relation and the dashed lines represents the sometimes affects relation. The function at the bottom is a so called dummy function. A dummy function is a function designed to test for a certain failure mode.

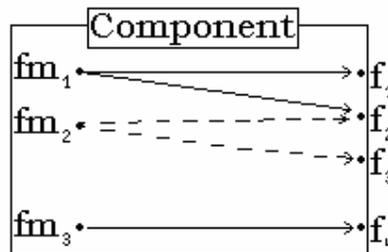


Figure 18. A schematic representation of a component and the relations between failure modes and functions.

A function can be a member of sets belonging to different failure modes and different components but can only belong to either \mathbf{AA} or \mathbf{SA} in each fm . The dependencies can be expressed through the matrix in Table 7. The table represents both \mathbf{AA}_{fm} and \mathbf{SA}_{fm} . A “1” at row x and column y indicates that failure mode y always affects function x . A “0” and an “X” represent never affects and sometimes affects respectively. Thus the ones in the table represents \mathbf{AA} and the X’s represents \mathbf{SA} .

Table 7. A matrix depicting the relations between failure modes and functions.

	FM _{c1}				FM _{c2}				...	
	fm _{cj1}	fm _{cj2}	...	fm _{cjn}	fm _{ck1}	fm _{ck2}	...	fm _{ckm}
f ₁	0	0		1	1	X		0		
f ₂	1	X		0	1	1		1		
f ₃	0	1		1	0	0		1		
.										
.										
.										
f _n	0	0		1	0	0		1		

The tests in the model are of two types, one that tests the functions in the system and one that tests the failure modes of the system. Although the failure modes are probably tested through dummy functions, i.e. functions designed specifically to indicate if a component is in a certain failure mode. The dummy functions are then treated as tests for the failure mode.

Tests focusing on functions are usually added in early stages of development when the different failure modes of the components are unknown. Later on when test and failure data are available, failure mode tests can be added.

There is also a matrix expressing the relationship between tests, functions, and failure modes. Even though there is a relationship between functions and failure modes both functions and failure modes must be represented in this matrix. This is because tests can test both functions and failure modes. An example of this sort of matrix is shown in Table 8. From the test results the mode of operation of the components can be derived using relationships in the matrices. How this is done is described in Section 2.1.3.

Table 8. A matrix showing the dependencies between tests, functions and failure modes.

	f ₁	f ₂	...	f _n	fm ₁	fm ₂	...	Fm ₁
t ₁	1	0		0	0	0		0
t ₂	0	X		0	0	0		0
.								
.								
.								
t _m	0	0		0	0	1		1

To clarify the model structure the polybox system, see Figure 12, will be modeled again but this time using a hybrid model. The list of components will naturally be the same, $\mathbf{C} = \{m_1, m_2, m_3, a_1, a_2\}$. Each of the components is now attached to a number of failure modes. To better be able to illustrate the workings of the model the fault space is extended to include for each component three different possible failures. sa_c is the failure mode when there is a shortcut between input A, i.e. the upper input in each component, and output. A shortcut between input B, i.e. the lower input in each component and output, is called sb_c . Finally an intermittent fault is added by the failure mode loose component, l_c . The index refers to the affected component.

We add some constraints to the inputs. It is assumed that each input to the system is greater than one. This is because otherwise all the faults would be categorized as sometimes affecting. Having only sometimes affecting fault would be a poor example of the model type.

Due to the constraints of the input the loose component failure differs from the other two failures since it is only sometimes affecting its function. The failure information can thus be encapsulated into,

$$\mathbf{FM}_c = \{sa_c, sb_c, l_c\}$$

The functions of the system can be divided into two different functions, one generating the output f and one generating the output g. There will also be a so called dummy function, sensitive to only one failure mode, l_{m3} . These functions will be affected by the different failure modes according to,

$$\left\{ \begin{array}{l} \mathbf{AA}_{sa_{m1}} = \{f\} \\ \mathbf{AA}_{sb_{m1}} = \{f\} \\ \mathbf{SA}_{l_{m1}} = \{f\} \end{array} \right\} \quad \left\{ \begin{array}{l} \mathbf{AA}_{sa_{m2}} = \{f, g\} \\ \mathbf{AA}_{sb_{m2}} = \{f, g\} \\ \mathbf{SA}_{l_{m2}} = \{f, g\} \end{array} \right\} \quad \left\{ \begin{array}{l} \mathbf{AA}_{sa_{m3}} = \{g\} \\ \mathbf{AA}_{sb_{m3}} = \{g\} \\ \mathbf{AA}_{l_{m3}} = \{z\} \\ \mathbf{SA}_{l_{m3}} = \{g\} \end{array} \right\}$$

$$\left\{ \begin{array}{l} \mathbf{AA}_{sa_{a1}} = \{f\} \\ \mathbf{AA}_{sb_{a1}} = \{f\} \\ \mathbf{SA}_{l_{a1}} = \{f\} \end{array} \right\} \quad \left\{ \begin{array}{l} \mathbf{AA}_{sa_{a2}} = \{g\} \\ \mathbf{AA}_{sb_{a2}} = \{g\} \\ \mathbf{SA}_{l_{a2}} = \{g\} \end{array} \right\}$$

The first set, for example, means that the failure mode shortcut between input A and the output of the component always will affect function f. The failure mode shortcut between input B and the output of the component will also always affect function f and the loose failure mode will sometimes affect the function f. In the description above the empty sets such as the sometimes affected functions of sa_{m1} , $\mathbf{SA}_{sa_{m1}}$, have been left out. The dependency information can be presented also in a matrix form which makes it more comprehensible, see Table 9.

Table 9. A matrix depicting the relations between failure modes and the two functions in the polybox example

	FM _{m1}			FM _{m2}			FM _{m3}			FM _{a1}			FM _{a2}		
	sa _{m1}	sb _{m1}	l _{m1}	sa _{m2}	sb _{m2}	l _{m2}	sa _{m3}	sb _{m3}	l _{m3}	sa _{a1}	sb _{a1}	l _{a1}	sa _{a2}	sb _{a2}	l _{a2}
f	1	1	X	1	1	X	0	0	0	1	1	X	0	0	0
g	0	0	0	1	1	X	1	1	X	0	0	0	1	1	X
z	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

In the model there must be a connection between the tests, the failure modes and the functions. In this example we test two functions, function f and function g , and one test testing the failure mode l_{m3} , i.e function z . The tests can be written as

$$\begin{aligned} t_1 &: ac + bd = f \\ t_2 &: bd + ce = g \\ t_3 &: \min\{(c(t)e(t) - z(t))^2, (c(t) - z(t))^2, (e(t) - z(t))^2\} \neq 0 \end{aligned}$$

In Table 10 the direct connections between the tests and their respective functions are depicted.

Table 10. The matrix relating the tests to their directly affected failure modes and functions.

	f	g	z
t ₁	1	0	0
t ₂	0	1	0
t ₃	0	0	1

There are also a number of parameters in the hybrid model such as component price, component failure rate, and component repair time that should be entered into the model if they are available. These parameters are used to generate different detection or isolation procedures, optimized with different focuses such as “Detect Malfunctions with Fewest Test” or “Prove Maximum Operation Before Detecting Malfunction” which will be further described in Section 4.2.3. These parameters have been left out in this theoretical example since they do not affect the capabilities of detection and isolation. In the next section the detection and isolation algorithm will be described.

4.2.2 Methods for Fault Detection and Isolation

In this section we have tried to describe an algorithm for detection and isolation that mirrors the rules used in eXpress. The algorithm is first explained and then illustrated with an example. The section is based on information found in [2] and from [1].

The algorithm used by DSI in eXpress is based on five rules and the input to the algorithm is a number of test results. These test results are used to derive the state of the system by the use of the five rules to conclude what mode each component is in. The output of the algorithm consists of the possible states of each and everyone of the components.

Each component c is associated with a set, \mathbf{Ps}_c , listing the possible failure modes of each component. From this set failure modes will be withdrawn when the algorithm allows us to exclude them. All the failure modes no longer suspected are listed in \mathbf{FM}_c . Since all the excluded failure modes of all the components are collected in \mathbf{FM}_c all the failure mode names must be unique.

The test results, $\mathbf{T} = \mathbf{T}_f \cup \mathbf{T}_g$, where \mathbf{T}_f is the set of failed tests and \mathbf{T}_g is the set of passed tests, indicate whether the functions are functioning correctly or not. In the

case of dummy functions, the test examines whether the component is in the particular failure mode that the dummy function tests for. The information contained in the test results can be translated using $\mathbf{T}s$ into two sets of functions, good functions \mathbf{F}_g which consists of functions that are suspected to be fault free by passed tests, and bad functions \mathbf{F}_f which consists of functions that are proven none functioning by the failed tests. From \mathbf{F}_f and \mathbf{F}_g the rules are used to derive the current failure mode of the components. $\Delta\mathbf{F}_g$ contains good functions and is only used to easily see if the fifth step generated any new good functions when doing the comparison in step six.

Algorithm II:

Inputs: A set of failed tests, \mathbf{T}_f , a set of passed tests, \mathbf{T}_g

Outputs: The set of $c_i(fm) \quad \forall i \in \{1, \dots, n\}$.

1. Initialization:

$$\mathbf{P}_{S_c} = \mathbf{FM}_c \quad \forall c$$

2. Derive the good and bad functions from the test results:

$$\mathbf{F}_g = \{f \mid f \in \cup_{t_j \in \mathbf{T}_g} \mathbf{T}s_j\}$$

and

$$\mathbf{F}_f = \{f \mid f \in \cup_{t_k \in \mathbf{T}_f} \mathbf{T}s_k \setminus \mathbf{F}_g\}$$

3. Compute which fm that can be excluded due to passed tests for each component c :

$$\mathbf{FM}_e = \{fm \in \mathbf{P}_{S_c} \mid \mathbf{AA}_{fm} \cap \mathbf{F}_g \neq \emptyset\}$$

$$\mathbf{P}_{S_c} = \mathbf{P}_{S_c} \setminus \mathbf{FM}_e$$

4. When all functions that are sometimes affected by a common failure mode are determined to be good that failure mode should be eliminated from suspicion:

$$\mathbf{FM}_e = \mathbf{FM}_e \cup \{fm \notin \mathbf{FM}_e \mid \mathbf{SA}_{fm} \subseteq \mathbf{F}_g\}$$

5. When all failure modes affecting a function are inferred to be good, that function is inferred to be good as well:

$$\Delta\mathbf{F}_g = \mathbf{F}_g \cup \{f \mid \forall fm. \{(\mathbf{AA}_{fm} \cup \mathbf{SA}_{fm}) \cap \{f\} \neq \emptyset \rightarrow fm \in \mathbf{FM}_e\}\}$$

6. If $\mathbf{F}_g \neq \Delta\mathbf{F}_g$ go back to step 3.

7. The output of the algorithm is $\mathbf{P}s$

The first step initialises the algorithm by setting all known failure modes to suspected by including them into the \mathbf{P}_{S_c} set. The second step derives which functions that can

be inferred as good and what functions that can be inferred as suspected bad by comparing the test signatures of passed tests and failed tests respectively. If a test is included in the test signature of both a passed and a failed test the function will be inferred as good. The functions inferred as good are all listed in \mathbf{F}_g and the bad functions in \mathbf{F}_f . The third step examines if any always affected function of a failure mode is included in the set of good functions. If so, that failure mode is excluded from the set of possible failure modes. The fourth step eliminates the failure modes whose all sometimes affected functions are inferred to be good. The fifth step concludes that a function is a new good if all the failure modes affecting that function are excluded from suspicion.

Now to illustrate this algorithm we are going to use the polybox example constructed in the previous section. We again insert a fault in the system at multiplier three, and the fault is of the type “short between input A and output”, sa_{m3} . This fault makes t_2 react since function g becomes faulty.

$$\mathbf{T}_g = \{t_1, t_3\} \quad \mathbf{T}_f = \{t_2\}$$

The initialisation of the algorithm sets the possible failure modes as follows,

$$\begin{aligned} \mathbf{P}_{s_{m_1}} &= \{sa_{m_1}, sb_{m_1}, l_{m_1}\}, & \mathbf{P}_{s_{m_2}} &= \{sa_{m_2}, sb_{m_2}, l_{m_2}\}, & \mathbf{P}_{s_{m_3}} &= \{sa_{m_3}, sb_{m_3}, l_{m_3}\}, \\ \mathbf{P}_{s_{a_1}} &= \{sa_{a_1}, sb_{a_1}, l_{a_1}\}, & \mathbf{P}_{s_{a_2}} &= \{sa_{a_2}, sb_{a_2}, l_{a_2}\} \end{aligned}$$

The second step of the algorithm will create the sets

$$\mathbf{F}_g = \{f, z\} \quad \mathbf{F}_f = \{g\}$$

which is used in the third step of the algorithm when concluding which failure modes that can be eliminated,

$$\mathbf{FM}_e = \{sa_{m_1}, sb_{m_1}, sa_{m_2}, sb_{m_2}, sa_{a_1}, sb_{a_1}, l_{m_3}\}$$

Using \mathbf{FM}_e to update \mathbf{P}_s gives

$$\begin{aligned} \mathbf{P}_{s_{m_1}} &= \{l_{m_1}\}, & \mathbf{P}_{s_{m_2}} &= \{l_{m_2}\}, & \mathbf{P}_{s_{m_3}} &= \{sa_{m_3}, sb_{m_3}\}, \\ \mathbf{P}_{s_{a_1}} &= \{l_{a_1}\}, & \mathbf{P}_{s_{a_2}} &= \{sa_{a_2}, sb_{a_2}, l_{a_2}\} \end{aligned}$$

When all functions that are sometimes affected by a failure mode are inferred to be good, that failure mode is excluded from the suspected failure modes. This applies to l_{m_1} and l_{a_1} since $\mathbf{SA}_{l_{m_1}} = \{f\}$ and $\mathbf{SA}_{l_{a_1}} = \{f\}$ where f is known to be good.

Propagating in the algorithm, the next step is to find functions that now can be inferred as good since all the failure modes affecting it are excluded. Although in this example the excluded failure modes do not infer any new good functions.

The output of the algorithm is thus

$$\begin{aligned} \mathbf{P}S_{m_1} &= \emptyset, & \mathbf{P}S_{m_2} &= \{l_{m_2}\}, & \mathbf{P}S_{m_3} &= \{sa_{m_3}, sb_{m_3}\}, \\ \mathbf{P}S_{a_1} &= \emptyset, & \mathbf{P}S_{a_2} &= \{sa_{a_2}, sb_{a_2}, l_{a_2}\} \end{aligned}$$

Which is to be interpreted as component m_1 and a_1 are fault free, and the other components are suspected.

The inference used when eliminating failure modes connected to m_1 and a_1 , that is using the fact that tests have passed, can also be used in a hierarchical manner. Consider the whole polybox as a system and the components as subsystems. If it is known by some global test that the whole polybox is good, then each individual subsystem, i.e. component, do not have to be tested since the whole system already has passed. This can reduce the number of needed tests in systems where a lot of the failure modes always affect functions.

Table 11. The updated matrix relating both the old tests and the new test to their directly affected failure modes and functions.

	f	g	z	g'
t ₁	1	0	0	0
t ₂	1	1	0	0
t ₃	0	0	1	0
t ₄	0	0	0	1

If the system is expanded with a sensor measuring y , another test t_4 , can be added. Observe that in the equation describing the test, z is not the function z but the signal z , i.e. the output of m_3 .

$$t_4 : y + z = g$$

This test makes it possible to monitor a_2 and the table specifying the test sensitivity must be updated to Table 11.

With this new test also failure modes connected to component a_2 can be removed from the set of suspected failure modes; this gives the remaining set:

$$\begin{aligned} \mathbf{P}S_{m_1} &= \emptyset, & \mathbf{P}S_{m_2} &= \{l_{m_2}\}, & \mathbf{P}S_{m_3} &= \{sa_{m_3}, sb_{m_3}\}, \\ \mathbf{P}S_{a_1} &= \emptyset, & \mathbf{P}S_{a_2} &= \emptyset \end{aligned}$$

Thus not even with this new test the failure could be isolated to component m_3 which is where we introduced the fault. Although, three of the four suspected fault modes are in m_3 so it would probably be the first component to examine manually.

4.2.3 Test Sequencing Algorithms

In eXpress there are a number of options available for which kind of algorithms to use for fault detection and fault isolation [69]. There is something called test weightings that define the order in which tests should be performed and depending on the choice of algorithms these weightings are chosen differently. There are seven algorithms concerning fault detection and seven algorithms concerning fault isolation and they are all described below.

4.2.3.1 Fault Detection Algorithms

Detect Malfunctions with Fewest Tests is the algorithm that is used when there is a desire to detect a fault with as few tests as possible. The test weightings are set so that the tests that are more probable to fail are performed first. Also non-intrusive tests are performed before intrusive ones. An intrusive test is explained as a test that is more disruptive than a non-intrusive one.

The *Detect Probable Malfunctions* algorithm is almost identical to the *Detect Malfunctions with Fewest Tests* algorithm only that the order of the non-intrusive tests is different.

The *Detect Critical Malfunctions* algorithm focuses on finding the most critical failures.

In the *Prove Operation with Fewest Tests* the tests that can prove the most number of functions as good are prioritized.

Prove Maximum Operation Before Detecting Malfunction is an algorithm that will prove as much as possible of the design as good before it detects a failure.

The *Minimize Switches in Monitored Stimuli* algorithm is used when there is a desire to minimize wear and tear on test equipment, e.g. that positions of switches are changed as little as possible. This algorithm can be combined with the other algorithms.

The *Detect Using Fault Codes* algorithm prioritizes tests that usually can detect a fault when it is present but that can not draw any conclusion about the eventual presence of the fault if it is not observed.

4.2.3.2 Fault Isolation Algorithms

The *Multiple Fault: Half-Split Failure Probs. (refinement postponed)* algorithm prioritizes tests that are certain to decrease the number of suspected functions regardless of whether the test passes or fails over tests that would only decrease the number of suspected functions for one of the outcomes pass or fail. In this algorithm refinement tests, tests that only reduces ambiguity when they pass, are postponed until after all isolation tests.

The *Multiple Fault: Half-Split Failure Probs. (refine where appropriate)* algorithm differ from the last one only in the way that refinement tests are not postponed.

The *Multiple Fault: Half-Split Failure Probs. (no refinement)* algorithm is the same as Multiple Fault: Half-Split Failure Probs. (refinement postponed) except that there are no refinement tests.

The algorithm *Multiple Fault: Static Health Monitoring (Operational refinement not postponed)* is a mix of Multiple Fault: Half-Split Failure Probs. (refinement postponed) and Multiple Fault: Half-Split Failure Probs. (refine where appropriate).

In the *Multiple Fault: Maximize Functions Proven by Refinement* algorithm the order in which are performed are decided by the weighting criteria of the algorithm. These weightings try to reduce the number of refinement by first performing other tests.

The *Common Cause: Half-Split Failure Probs.* algorithm uses common cause, which means that there is an assumption that there can only be a single fault present. The weightings of this algorithm prioritizes tests that, when they pass, halve the number of suspected functions.

The *Common Cause: Half-Split Failure Probs. (Max. Depth = 10)* algorithm is the same as *Common Cause: Half-Split Failure Probs.* only that no more than ten tests can be performed in each isolation path.

4.2.4 Formats for Storing Data

In this section the format in which design data, maintenance data, test data, and diagnostic data are stored in eXpress is presented. This format is called Diagnostics Markup Language (Diag-ML) and is an XML schema that is used for capturing and transferring diagnostic knowledge and was created in 2001 by a consortium of companies. Today this consortium consists of two companies; DSI International and TYX Corporation [5]. Being XML-based, Diag-ML is extensible and the user can extend the format to meet the requirements of his or her specific application [9]. Diag-ML is an open and non-proprietary language and companies/tools that support it are:

- Sörman's Rodon, see 3.5.1, can import Diag-ML from eXpress for static or run-time diagnostics via an IETM (Interactive Electronic Technical Manual) [4]
- AIMSS' Raytheon can import Diag-ML from eXpress for static or run-time diagnostics via an IETM [4]
- Impact Technologies' ReasonPro (see 3.5.2) can import Diag-ML from eXpress for Prognostic Health Management Reasoning [4]
- Tyx' TestBase uses Diag-ML natively [4]
- NASA's Jet Propulsion Laboratory (JPL) used Diag-ML when they required a diagnostic transfer format between eXpress and JPL's BEAM/SHINE tool [4]
- VSE Corporation's Diagnostician (See 3.5.3) [10]
- Boeing Phantom Works' IVHM Test Bench [10]
- DSI's OSS [10]

The last three are only listed as supporters of Diag-ML on DSI's homepage [10] why no information about how they support Diag-ML is provided.

A Diag-ML file contains the four main sections DesignData, MaintenanceData, TestData, and DiagnosticData [46] which are described below. Figure 19 displays the basic structure of a Diag-ML file.

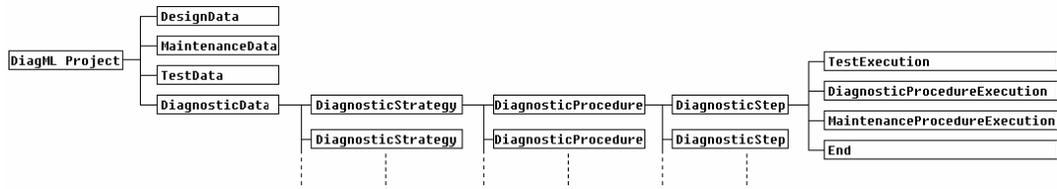


Figure 19. Basic structure of a Diag-ML file.

In the DesignData section information is stored that relates to the model of the Unit Under Test (UUT) which include UUT components, functions, and failure modes, which were described in Section 4.2.1. In the polybox example in Section 4.1.1 a multiplier is an example of a component, an example of a function is that f is correctly calculated given inputs in the polybox, and a shortcut between one of the inputs and the output of a multiplier is an example of a failure mode. MaintenanceData describes what should be adjusted or replaced depending on what diagnostic results that are reached. In the polybox example, maintenance data could be that a multiplier should be replaced if it is found faulty. In the TestData section is where the tests are defined as to the location of the tests and test parameters. The location can be specified by a numbered position in the model. In the DiagnosticData section one or more DiagnosticStrategy elements can be contained. A diagnostic strategy is an executable entity that is used to diagnose a specific UUT. In each diagnostic strategy there are one or more DiagnosticProcedure elements. Diagnostic procedures can invoke other diagnostic procedures and each of them contains a number of DiagnosticStep elements which in its turn is one of the operations TestExecution, DiagnosticProcedureExecution, MaintenanceProcedureExecution, or End. These four operations are described below.

TestExecution is the execution of a test which is defined in the TestData section. The DiagnosticProcedureExecution operation calls to another diagnostic procedure which is defined within the same diagnostic strategy. A MaintenanceProcedureExecution represents the execution of a maintenance procedure which is defined in the MaintenanceData section and finally End is the operation that terminates the diagnostic procedure

An example of an application of Diag-ML is described in [46]. In collaboration between DSI and TYX a simple diagnostic application consisting of an electrical circuit that models the operation of an amplifier is developed. The process from designing the model in eXpress to testing the design in TYX' software tool TestBase is described as follows: First the model is built in eXpress for the UUT by defining the ports of the UUT, its components, the interconnection between the components, the components failure modes, and the functions that model the propagation of faults through interconnections. Then a set of tests is defined in eXpress. For each test, attributes are specified, and examples of these attributes are high and low limits for

measured voltage in certain points. The attributes are transferred via Diag-ML to TestBase. The experiment proceeded with generating a diagnostic strategy in eXpress and evaluating the quality of the fault isolation which means that the size of the ambiguity groups are checked not to be too large. The diagnostic strategy is then exported to a Diag-ML file. After TestBase test procedures are developed the Diag-ML file is imported in TestBase as a test strategy (corresponding to the diagnostic strategy in eXpress). The test strategy is then compiled to be ready for run time execution by the TestBase Diagnostic Controller.

4.2.5 Design for Testability Aids

This section describes what kind of aids that is present in eXpress that can help evaluate and improve the testability of a system.

Reports that can be generated from eXpress can inform about the effectiveness of current sensor placements and also the best placement for additional sensors can be suggested [55]. There are also features that provide information for helping to make decisions about system design alternatives, redundancies, and maintenance strategies [56].

By adding expected or measured fault probabilities the software can calculate the expected isolation and detection rate of the system given the current sensor placements and other inputs. Also the average ambiguity group size can be calculated. The number of tests needed to isolate and detect failures is also calculated. All these features can be used to modify the system for better testability.

4.2.6 Modifiability

About modifiability of eXpress models the following can be said: As will be described in Section 4.2.7 model topology can be exported from eXpress to a Microsoft Excel spreadsheet with Spreadsheet Topology Export and the topology can also be imported from an Excel spreadsheet into eXpress with Spreadsheet Topology Import. If the user would like to change the model data it is preferable that this is done in the Excel spreadsheet rather than manipulating the data directly in eXpress [48]. The reason of this being that it will save a lot of time since the database has to be updated for each individual change if they are performed within eXpress. For a large model it is consequently better to do the changes via an Excel spreadsheet and thus all updates to the database are performed in one single update.

4.2.7 Compatibleness

The compatibleness of eXpress is here described and we will see that Microsoft Excel spreadsheets as well as the XML schema Diag-ML (See Section 4.2.4) are two important means to import and export data and communicate with other software. Finally Computer Aided Design (CAD) tools and some other formats that are supported by eXpress are listed.

Importing data into eXpress can be done from Microsoft Excel spreadsheets [47]. The spreadsheet that is to be imported has to conform to a certain format in order to be imported successfully. There can only be one table in each spreadsheet and the data should start from row 2, column A. Two correctly formatted spreadsheets can be seen

in Figure 20 and Figure 21. The different types of spreadsheet imports that eXpress supports are [47]:

- *Spreadsheet Attribute Import* is used for importing characteristics for e.g. an object such as those that can be seen in Figure 20, which are Object Name, Cost, and Failure Rate.
- *Spreadsheet Failure Mode Import* is used for importing failure modes of objects. Figure 21 displays a spreadsheet that can be imported with Spreadsheet Failure Mode Import and this spreadsheet displays the columns Object, Failure Mode, Percentage, and Functions. The Object column gives us information about which object that is concerned. The Failure Mode column describes what failure modes there are for respective object. The information in the Percentage column tells us how large share of the total number of failures of the object that is constituted by the failure mode at hand. In the Functions column functions that are affected by the failure mode at hand are listed.
- *Spreadsheet Failure Effect Import* is useful when FMECA data, such as probabilities and consequences of failure modes, are imported from spreadsheets. The FMECA features of eXpress were mentioned in Section 4.2.1.
- *Spreadsheet Test Import* is used for importing tests. Tests appeared both in Section 4.2.1 and Section 4.2.2 and a test definition in eXpress describes the diagnostic conclusions that can be reached after a passed or failed test [67].
- The *Spreadsheet Topology Import* option is used when model topology is to be imported. The following entity types can be imported or updated [49]:
 - *Dependencies* describe for each output signal of an object which of the input signals the output signal depends on. For instance, if an object has the inputs A, B, and C and the outputs D and E a dependency can describe that output D depends on A and B.
 - *Components* are the most basic type of design objects. In these relationships between inputs and outputs are established in internal dependencies. Components are non-hierarchical objects that can fail e.g. a valve in a hydraulic system.
 - *Ports* are the input and output points of an object e.g. the two points where an electrical resistor is connected to the wire. The type of a port defines the directional flow of the signal travelling through that port.
 - *Output functions* specify the dependencies, described above, between a given output port of an object and one or more of its inputs. If an object has multiple capabilities that are possible to test separately, these capabilities are represented with multiple output functions which all have to be proven good by diagnostics for the object to be considered as correctly functioning.
 - *Object states* are groupings of output functions that are used to further define an object.
 - *Nets* are used to connect objects. Examples of signal paths that can be represented by nets are wires in an electrical design and pipes in a hydraulic design.
 - An *I/O flag* is an object that marks the linking point where a signal goes into or out of an entire design. An I/O flag is not a possible source of failure.

- *Annotations* are non-functional objects that only provide information for example a title on a design sheet or a box around a number of objects that indicates a group

	A	B	C
1	Object Name	Cost	Failure Rate
2	U1	\$200.00	212.766
3	U2	\$12.00	169.492
4	R1	\$0.50	62.893
5	R2	\$0.50	63.291
6	R3	\$1.90	63.291

Figure 20. A correctly formatted spreadsheet for importing attributes [49].

	A	B	C	D
1	Object	Failure Mode	Percentage	Functions
2	R1	Open	80	Fctn R1
3	R1	Short	5	Fctn R1
4	R1	Change in R	15	Fctn R1
5	SW1	Stuck Open	50	Fctn SW-Pos1
6	SW1	Stuck Closed	50	Fctn SW-Pos2

Figure 21. A correctly formatted spreadsheet for importing failure modes [49].

As well as importing data from an Excel spreadsheet eXpress can also export to an Excel spreadsheet. In Spreadsheet Topology Export [48] the same kind data that was described in Spreadsheet Topology Import can be exported to an Excel spreadsheet. This can be used for passing topology information from eXpress to another tool. The user can also export to Excel, modify the file to update the existing model and import it again using the Spreadsheet Topology Import [48].

Diag-ML, described in 4.2.4, or XML can be used as export formats when exporting eXpress models to run-time tools from eXpress [1]. Companies/tools that have experience in exchanging diagnostic information with eXpress via Diag-ML are listed in Section 4.2.4.

eXpress has a new feature that is called EDA Import Module which utilizes a new standard called EDAXML [53]. EDAXML is an XML form of EDIF. EDIF can also be imported. The CAD tools [53] that are supported by eXpress are Cadence, Mentor DesignArchitect, Viewlogic/Innoveda, PADS, P-CAD, and OrCAD. Report formats that can be generated from eXpress include: Pdf, Doc, Rich Text Format (RTF) [65], and XML [66] Other formats that are listed as supported by eXpress are Tab-delimited ASCII and Common Object Model (COM) Interface [52].

4.2.8 Libraries

Regarding libraries within eXpress there are a number of example designs [52] that accompany eXpress and these include a 1553 data bus, an altimeter, an amplifier, and a hydraulic flow example.

eXpress does not come with a library of components and this is motivated by the large diversity in component usage within DSI's customer base; e.g. the component failure rates vary significantly based on their usage [1]. Failure rates are imported into eXpress using XML or spreadsheets instead. As new data become available these

failure rates must continually be updated for helping improve the correctness and usability of the diagnostic system [2]. Although lacking conventional component library there is a graphical library that assists in the development of model topology [1].

4.2.9 Disciplines of Modelling

The disciplines, in which eXpress can be used for modelling, are electrical, mechanical, hydraulic, software and also processes such as a chemical process [58].

4.3 Raz'r

In this section Raz'r from OCC'M is examined. For a short introduction see section 3.3.3. This section differs from the sections describing the other two tools since it lacks the Test Sequencing algorithm part, this is because no information has been found concerning this matter in Raz'r.

4.3.1 Representation of Knowledge

Raz'r uses a model type called Qualitative models, see Section 2.1.2. This model type will be described and exemplified in this section.

A benefit of using qualitative models is that the model can be based on the models used for controlling the system. This makes the transition and integration between the design process and the creation of a diagnosis system easier. The basic concept for the integration is shown in Figure 22.

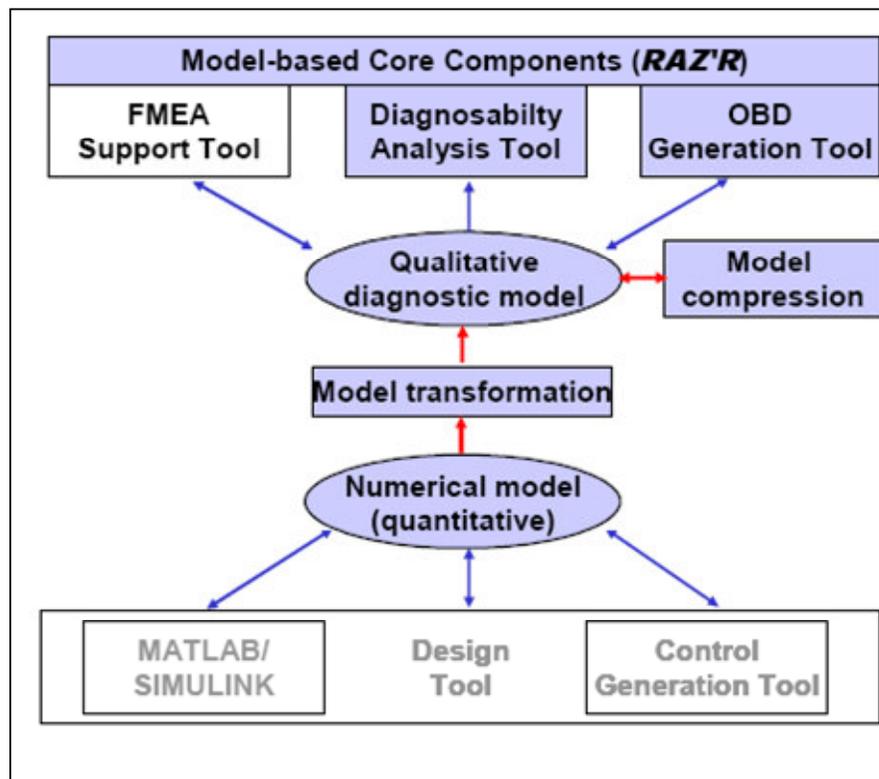


Figure 22. An integration of model based tools based on model transformation [32].

The quantification makes qualitative models less computationally demanding than its quantitative counterpart. A gain is also a more compact model. The quantification also infers a loss of information which could degrade the diagnosability of the system if the granularity is poorly chosen. There is currently research being done to optimize and automate the choice of granularity [8].

There are a number of different qualitative models which represents a system in different ways. A thorough description can be found in Chapter 2. The models used in Raz'r are a more sophisticated form of naïve physics models. They are not constrained to merely three states but can be chosen to contain a suitable number of states depending on what granularity is deemed to bring the highest usability of the diagnostic system.

The model includes a set of components, representations of the connections between them, the domains of the variables used in the model, and definitions for each of the types of components present in the model. The connections are defined through the terminals of the components that are the models connection points. The terminals are often defined as belonging to a specific physical area, such as electrical or pneumatic, and infers a number of associated constraints in the form of equations. If the terminals are said to belong to the electric area then all the terminals must fulfil Kirchhoff current law and the voltage into the terminal must equal the voltage out of it.

Also a number of modes of operation for the component are defined. The modes are specific to each kind of component and define a number of equations and relations between the component terminals that needs to be fulfilled for the component to be in that mode. Together with the equations parameters and state variables are connected to the components.

To model the polybox, illustrated in Figure 12, with this model type one needs to start by listing the components and define the different component types. All the components are listed in the set,

$$\mathbf{C} = \{m_1, m_2, m_3, a_1, a_2\}$$

The definitions for the multipliers, starts with the set of terminals,

$$\mathbf{T} = \{A, B, C\}$$

where A and B are the input ports and C is the output port. None of the terminals are specified to a certain physical field in this theoretical example. They are merely submitted to one simple equation,

$$T_{in} = T_{out}$$

meaning that the value coming out of the terminal must be same as the one coming in. If the terminal was of some other kind, e.g. electrical, then Kirchhoff's current law would have been applied.

The modes of operation, MO , in the multiplier are, the first mode, OK , which represents when the component is functioning correctly, the second mode, AS , and the third mode, BS , represent when there is a short between the input A and the output C and, the input B and the output C respectively. When the component is functioning in an unknown way, the fourth mode, UK , is used. All the three last modes are considered to be fault modes.

$$MO \in \mathbf{MO}_m = \{OK, UK, AS, BS\}$$

Where each mode is described by the following equations,

$$\left\{ \begin{array}{l} MO_m = OK \rightarrow C = A \times B \\ MO_m = AS \rightarrow C = A \\ MO_m = BS \rightarrow C = B \\ \text{else} \\ MO_m = UK \end{array} \right.$$

The adder type of component can be described in a similar way; the terminals function in the exact same way and the modes are defined by,

$$\mathbf{MO}_a = \{OK, UK, AS, BS\}$$

These modes function in corresponding way as the modes for the multipliers although the OK mode naturally adds instead of multiplies. The modes of operations are described by

$$\left\{ \begin{array}{l} MO_a = OK \rightarrow C = A + B \\ MO_a = AS \rightarrow C = A \\ MO_a = BS \rightarrow C = B \\ \text{else} \\ MO_a = UK \end{array} \right.$$

There is no need for parameters or state variables in the adder component either.

Using qualitative models the signals need to be quantified. In this model the signals are quantified in the somewhat untypical domain, all integers between zero and fifteen, giving the set,

$$\mathbf{D} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$$

When studying this theoretical example the importance of choosing the sets and granularities of the signals correctly quickly becomes apparent if all the input signals are chosen as 10. Already after the multipliers the chosen domains are incapable to represent the system. A physical system comes with a number of built in constraints; the maximum speed for a car or the maximum resolution of a sampled signal, both

gives limits to the span and the granularity of the set representing the signal. To illustrate this more thoroughly another system will be modelled.

This time an electrical circuit is modelled. The circuit is illustrated in Figure 23. The components in this system are, a voltage source, s , three lights, l_1 , l_2 and l_3 , and six wires connecting them, w_1, w_2, w_3, w_4, w_5 and w_6 . They are all collected in the set of components,

$$C = \{s, l_1, l_2, l_3, w_1, w_2, w_3, w_4, w_5, w_6\}$$

In the component set, observe that also the wires are modelled; this is due to the fact that they as well can be in a failed state and thus can be the source of a failure.

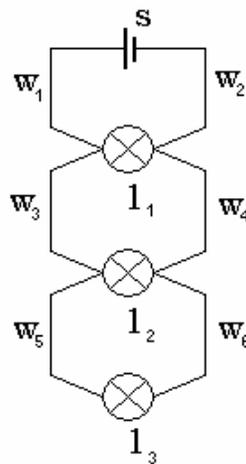


Figure 23. A simple system with three lights connected to a battery through parallel connections [26].

In the model the signals must be specified into domains; in this case only two domains for each signal are needed.

$$D_{voltage} = \{0,1\} \quad D_{current} = \{0,1\} \quad D_{light} = \{0,1\}$$

The first signal represents the voltage in the circuit which can be either zero or higher. The second signal states that also the current in the circuit is either zero or higher. The lights are either on, 1, or off, 0. Assuming that the limit is selected so that zero current is equivalent to no light and current is equivalent to light.

All the components has two terminals of the electrical type, IN and OUT where the first is the component input port and the later is the output port. In each terminal there is a port for each signal. For instance, if there is a voltage coming into a wire or light that would be expressed as $IN_{voltage} = 1$.

The different modes of operation for the source are

$$\mathbf{MO}_s = \{OK, D\}$$

The *OK* mode is when the source is feeding the circuit with a voltage, and the *D* mode is when the source is drained and thus does not feed the circuit with a voltage. The equations describing this,

$$\left\{ \begin{array}{l} MO_s = OK \leftrightarrow OUT_{voltage} = 1 \\ MO_s = D \leftrightarrow OUT_{voltage} = 0 \end{array} \right\}$$

The lights can be in either the *OK* mode and thus lit if a current is present, or they can be broken, *B*, not lit at all.

$$\begin{array}{l} \mathbf{MO}_l = \{OK, B\} \\ \left\{ \begin{array}{l} MO_l = OK \rightarrow \left\{ \begin{array}{l} Light = 1 \quad OUT_{current} = 1 \quad \text{if } IN_{current} = 1 \\ Light = 0 \quad OUT_{current} = 0 \quad \text{if } IN_{current} = 0 \end{array} \right. \\ MO_l = B \rightarrow Light = 0 \quad OUT_{current} = 0 \quad OUT_{voltage} = 0 \end{array} \right. \end{array}$$

The wires also have two modes of operation,

$$\mathbf{MO}_w = \{OK, C\}$$

If in the *OK* mode the output voltage is the same as the input voltage, if in the *C* mode, the wire is cut and there will be no output voltage.

$$\left\{ \begin{array}{l} MO_w = OK \leftrightarrow IN_{voltage} = OUT_{voltage} \quad IN_{current} = OUT_{current} \\ MO_w = C \leftrightarrow OUT_{voltage} = 0 \quad OUT_{current} = 0 \end{array} \right\}$$

The tests in this case are only whether the lights are on or not.

4.3.2 Methods for Fault Detection and Isolation

OCC'M uses a modified version of the extended General Diagnostic Engine, GDE+, see Section 2.1.3.2, to perform their fault detection and isolation. How the modifications are done or what the consequences of them are is unavailable to us since OCC'M do not publish information about their software components [51]. In the following section a description of how the GDE+ creates a diagnosis is shown. Hopefully this is an accurate enough description to show the general workings of the Raz'r algorithm.

GDE+ differs from GDE by also utilizing fault mode behaviour. This makes it possible to get better fault isolation since we not only can detect when a component isn't working properly but also when it is functioning according to a predefined fault mode behaviour. Usually we can make an educated guess about the fault mode behaviours of a component, e.g. a wire can have a fault mode behaviour that can be described as an infinite resistance, representing a cut off in the wire.

One major drawback of the GDE+ is that it is in many cases hard or even impossible to predict the faulty behaviour of a component and if we can not predict the behaviour we can not model it. There is one way around this called focusing. This approach solves the problem but limit the benefits of GDE+ severely, the algorithm then behaves as GDE when no known fault mode can describe the behaviour [26]. It functions by adding an unknown fault mode to the fault free mode and the fault modes. First the focused algorithm assumes all components to be fault free. If this can not explain the observations, known fault modes are tried as well. If also this fails, the algorithm examine if any unknown fault mode could explain the observations.

To exemplify the GDE+, the polybox will be treated. We introduce the same fault as in the other examples, i.e. the fault $MO_{m_3} = AS$ in m_3 . In the polybox the observations are the outputs in f and g , and the inputs a, b, c, d , and e . Assume that the inputs are,

$$(a \ b \ c \ d \ e)^T = (2 \ 2 \ 3 \ 2 \ 2)^T$$

This would generate, in the fault free case,

$$\begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} 10 \\ 10 \end{pmatrix}$$

Instead the output is

$$\begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} 10 \\ 7 \end{pmatrix}$$

The behaviour modes in which all components are functioning correctly,

$$\{MO_{m_1} = OK, \ MO_{m_2} = OK, \ MO_{m_3} = OK, \ MO_{a_1} = OK, \ MO_{a_2} = OK\}$$

does not explain the observations and thus a fault has been detected. The discovered nogoods are thus,

$$\{MO_{m_3} = OK \ MO_{m_2} = OK \ MO_{a_2} = OK\}$$

This in turn generates calls, m_2, m_3 and a_2 into suspicion. The combinations of the failure modes for these components are called candidates. There are six minimal candidates, i.e. candidates that not is a subset of another candidate. The minimal candidates are listed in Eq 4-III.

When we also use the information contained in the failure modes of the components the following nogoods are detected

$$\begin{aligned}
 & \{MO_{m_2} = AS, MO_{m_3} = OK, MO_{a_2} = OK\} \\
 & \{MO_{m_2} = BS, MO_{m_3} = OK, MO_{a_2} = OK\} \\
 & \{MO_{m_2} = OK, MO_{m_3} = AS, MO_{a_2} = OK\} \\
 & \{MO_{m_2} = OK, MO_{m_3} = BS, MO_{a_2} = OK\} \\
 & \{MO_{m_2} = OK, MO_{m_3} = OK, MO_{a_2} = AS\} \\
 & \{MO_{m_2} = OK, MO_{m_3} = OK, MO_{a_2} = BS\}
 \end{aligned}
 \tag{Eq 4-III}$$

Since all of the minimal conflicts but the third in Eq 4-III are nogoods the focused inference algorithm concludes that the set

$$\{MO_{m_1} = OK, MO_{m_2} = OK, MO_{m_3} = AS, MO_{a_1} = OK, MO_{a_2} = OK\}$$

not generates any conflicts. The faulty component have thus been isolated and we have also isolated in what fault mode it is.

4.3.3 Formats for Storing Data

In Raz'r the qualitative models, described Section 4.3.1, are stored in XML or relational databases [29]. In the same section modelling concepts such as components, their behaviour, and the interactions between them were described and for representing this there is an XML Schema [32].

4.3.4 Design for Testability Aids

Raz'r has incorporated tools for help designing for testability (DFT) and automatic generation of FMEA. The tools for DFT include a detectability analysis tool with which the detectability of the failed model components can be evaluated and a discriminability tool used to evaluate if a component, in case of a fault, can be pointed out as the faulty component, i.e. if it is part of an ambiguity group. These tools can be used together with a set of models where sensors are placed in different places among the models to sieve out the model with the best accordance with the wanted detectability and discriminability. In such a report, pairs of modes are listed along with the probability that we can determine which of the two modes in respective pair that is the current mode.

How the FMECA is generated is described more in detail in [50]. There it is also said that the automatic generation of the failure mode effects and analysis in the program is, under certain circumstances, more reliable then a traditional hand made generation.

4.3.5 Modifiability

Regarding modifiability in Raz'r it can be said that the fact that Raz'r uses qualitative models affects the modifiability of the software models. The initial model is fairly difficult to develop due to the importance of choosing a correct granularity and the different fault modes which demand a comprehensive knowledge of the system, both when it is working correctly and when it is working in some of its fault modes. Once the different models have been created and the functionality of the systems is captured

in the model it is no big issue to update the system. The update is made automatically from the structural information of the model [31].

4.3.6 Compatibleness

This section deals with what kind of formats that can be imported to and exported from Raz'r. Raz'r can take as input any qualitative model as long as it obeys the defined XML schema and further semantic constraints that is checked by the software [59]. The arrows in Figure 22 represent information flow in form of XML files. In this figure one can see that Raz'r can interact with Matlab/Simulink. Under certain circumstances Matlab/Simulink models can be abstracted to qualitative models and be automatically imported into Raz'r. The circumstances are that where the Simulink model contains an integration block the model has to be modified in the way that can be seen in Figure 24.

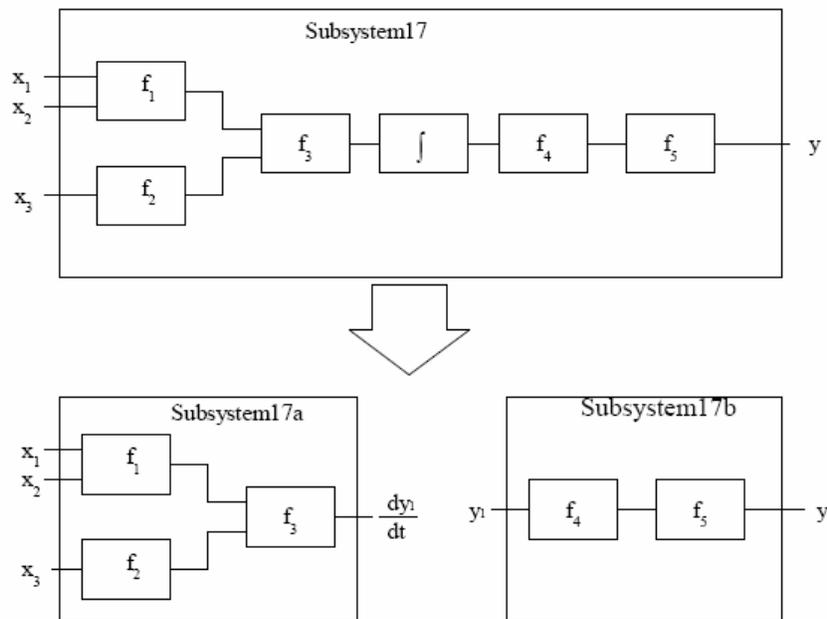


Figure 24. Modification that is made to Matlab/Simulink model to make it importable for Raz'r [8].

An integration block is eliminated from the subsystem at hand and its input, that is the derivative, becomes an output of the first part of the modified subsystem and its output, the integrated value, becomes an input to the rest of the subsystem [8].

In [6] it is described how Raz'r is capable of automatically generating C code based on, in this case, signals from a CAN bus. The compiled C code was then run on an Electronic Control Unit (ECU) that controlled the electric side windows and mirrors of a car. In this case the generated C code, consisting of the signal pre-processing, the compressed system model, and the diagnosis algorithm, when compiled required only 25 kB of memory.

Fault trees can be generated from Raz'r as well and the possible output formats are [33]:

- XML
- HTML
- ID3 (An algorithm for generating decision trees)
- C-code.

There is also an import function in Raz'r that makes it possible to import net lists [29], that describe components and connectivity between them in an electronic design.

4.3.7 Libraries

A library in Raz'r consists of four different concepts: *component types*, *constraint types*, *domain types*, and *terminal types*. These are described as follows [33]:

- *Component types* describe classes of physical objects. Objects within the same component type are very similar in the way that they behave and are constructed. An example is a resistor component type of which different instances only differ in the parameter resistance. Each component type has five functions:
 - *Local effects* describe the consequences a fault in the component will have at the same level in the hierarchy as the component itself.
 - A *Mode* defines a behaviour that is common to every instance of the component type. In the case of diagnosis the mode can be either ok or faulty in some way. A default mode should be set in each component type.
 - A *Parameter* is a constant that describe the behaviour of an instance of a component type. To exemplify by a resistor, the parameter would be the resistance.
 - *State variables* are similar to parameters but they can vary over time and are dependent of previous values of their selves. An example is the charge of a capacitor.
 - *Terminals* are the entities through which a component interacts with the rest of the model and as described in Section 4.3.1 each terminal belong to a physical area such as electrical or pneumatic. A resistor would have two electrical terminals transferring both potential and current.
- A *Constraint type* defines a restriction on a set of variables and parameters. There are predefined constraint types such as Ohm's law but the user can define his or her own.

- A *domain type* specifies the possible values for a variable or parameter. Raz'r, being based on qualitative modelling, uses finite domains. The predefined domains in Raz'r are:
 - Signs {-, 0, +}
 - Positive {0, +}
 - Boolean {True, False}
 - Position {On, Off}
 - Connectivity {0, Pos, Inf}
 - Vqual {Ground, Between, Src}

Thus are the possible values in e.g. the domain of Booleans True and False.

- *Terminal types* describe what happens at the interaction points. A terminal type specifies variables that concern not just one component. An electrical terminal for example can have the variables *delta_voltage* and *delta_current* which describe electrical connections.

OCC'M does have various component libraries developed in previous projects and they are reused on a case by case basis. The software does not automatically come with any basic library as it is not normally sold without consulting [31].

4.3.8 Disciplines of Modelling

In an e-mail conversation with a co-founder of OCC'M he explained that they have experience from modelling in and have component libraries in the disciplines electrics, pneumatics, and hydraulics [59].

5 Comparison of tools

In this chapter we will compare the different tools presented in Chapter 4. The effort to model a system is discussed and what pros and cons this does infer. A model that requires a lot of information and knowledge to be available takes longer to complete. This could also infer a better diagnostic capability in the form of better chances of detection and isolation.

Furthermore, it is discussed what inference algorithm that are used in the tools. The type of inference algorithm is of importance since this affects the detectability and isolation capabilities of the tools. It also affects the amount of computing power needed and the amount of time it takes to perform a search for one or more faulty components.

The different formats that are used for storing data in the different tools are compared as well as what has been presented in the Design for testability aids sections for the respective tools. Furthermore comparisons are made for the sections Modifiability, Compatibleness, Libraries, and Disciplines of modelling.

5.1 Modelling Effort

The tools all have different approaches to how the models are supposed to be constructed and to what they are supposed to contain. What in TEAMS is called a signal is very similar to what in eXpress is called a function. The tools both share a common difficulty, that there can be some difficulties to identify what a function/signal is. For example, in the polybox it was initially not obvious if the whole path of the different inputs to the respective outputs should be a signal/function or, if each single component was to be modelled as having its own signal/function. Perhaps this becomes clearer when the experience of using the tools grow.

The biggest effort to create a model is demanded if we want to create a qualitative model. This is due to the need for all relations to not only be formulated and physically modelled but also the need to choose the granularity of the signals correctly. A task that is not easily performed. There is currently research being done to try to automate the process of choosing the granularity to preserve the diagnostic capabilities without losing the benefits of the lower computational requirements compared to quantitative modelling [8]. To be able to choose the correct granularity, extensive knowledge is required about the system, not only about the system when it is functioning as it should but also when a component is faulty. For example if a sensor is fed with an insufficient voltage, will the output of this sensor be zero, outside a possibly correct interval, inside such an interval, or something totally random? This kind of knowledge must be available to be able to fully reap the gains of the model type [62].

When comparing the different modelling examples in Chapter 4 attention must be paid to the fact that although it is the polybox system modelled with each tool there are some differences in the system fault modes in the different sections. In TEAMS there are only two modes, one fault free and one failure mode in each of the components of the system. In eXpress and Raz'r the components in the polybox have

three respective two failure modes. Since we could not model the intermittent fault in Raz'r this failure mode was included in the unknown fault mode.

As Raz'r uses physical models of the system the feature of being able to define and use state variables is useful. This enables for dynamic systems to be modelled in the tool. Also in this matter, Raz'r is alone amongst the three tools to be capable of as far as what has been found in this examination.

5.2 Fault detection and Isolation Capabilities

Regarding the different fault detection and isolation capabilities of the tools there are some differences. The types of models that the tools work with open for different approaches. Once again the solutions presented in eXpress and TEAMS are quite similar while the solution used in Raz'r is more advanced.

Although the same information was available to us in the polybox example the tools reached different degrees of isolation. This is because the tools have differences in how well suited they are to model the knowledge of the different fault modes. TEAMS could not conclude whether m_3 or a_2 were faulty, eXpress could not isolate between

$$l_{m_2}, sa_{m_3}, sb_{m_3}, l_{m_3}, sa_{a_2}, sb_{a_2}, l_{a_2}$$

and Raz'r came to the conclusion that the faulty component was m_3 , and that it was now functioning according to the description given for fault state AS. This is where the benefits from the more extensive modelling come in. Since the fault states also have been modelled it is easy to test if any of those modes of operation can explain the sensor readings and behaviour of the system.

Raz'r also has the benefit of being able to use state variables. In the sample systems sent to the companies the only one who returned a usable model of the dynamic air intake system was Raz'r.

Another advantage with Raz'r is that there is no need to define the tests. The inference works with the observations of the system. These observations can be obtained from sensors. In both eXpress and TEAMS the tests must be constructed before we are able to use the diagnostic systems created with these tools. Constructing the tests is not a simple task since the tools assume that the tests are perfect. This means that if a test has not reacted it is guaranteed that none of the behavioural modes the test is sensitive to correctly describes the behaviour of the system. This is very hard to accomplish due to noise in sensor values and model errors [60]. A test reacts if a test quantity exceeds a certain limit value. The test quantity is calculated from sensor values and these values always include some amount of noise. If the limit is set too close to the normal value of the variable, i.e. the expected value in the fault free mode, the risk of the noise making the value exceed the limit is big. If the limit is set too far away from the normal value there is a risk that there is a fault present in the system but the test has not reacted.

Since eXpress and TEAMS use quite similar model types and inference algorithms there are probably more comparisons that we could make to further examine the differences and the pros and cons that those differences infer on the diagnostic systems created with the tools. For example better examine the use of fault modes in the tools.

A questionable aspect of the eXpress algorithm is whether it really is correct to exclude a failure mode, known to only sometimes affect a function, just because it at the present do not affect any function. This makes the design of the tests very important for the quality of the diagnostic system. The tests must be sensitive enough to always detect a fault and insensitive enough to not generate false alarms.

Regarding aspects such as computational speed and requirements of the inference algorithm no investigation has been done since comparable data have not been accessible. Due to the lack of access to the tools, test data could not be generated either.

5.3 Formats for Storing Data

In all the three tools XML files are used for representing models and in TEAMS as well as in Raz'r also relational databases are used. Raz'r and eXpress use XML Schema for setting the rules to which their XML files must conform while TEAMS on the other hand uses Document Type Definition (DTD). DTD is generally considered out of date and the XML Schema is favoured over the DTD. In DTD we can only specify the element structure, i.e. the allowed names of elements and attributes in an XML file, whereas in the XML Schema we can also specify numeric ranges for a certain element type or specify a list of possible values for the element type to have. Even though QSI, the producer of TEAMS, writes in [63] from 2001 that they will move on from DTD to the then newly developed XML Schema, we were not able to find any trace of it today. In [21] from 2006 we can read that the Document Type Definition is still used.

Of the three solutions for using XML files for model storing the one that we have been able to find a somewhat widespread support for in the industry is eXpress' Diag-ML. The other two solutions seem more to be natively used formats.

5.4 Test Design Capabilities

When it comes to Design For Testability (DFT) it is hard to get comparable data and hard to evaluate the quality of the services provided. Even if the different tools claim to have the same services such as detection rate and FMEA generation, the quality of these services is hard to evaluate. If a sample system was modelled with all the tools and the tools themselves were available a comparison could have been made of e.g. what the calculated isolation ability, mean time to failure and maximum ambiguity group size would have been calculated to in the different tools. Also a comparison of where new sensors are recommended to be placed and the contents of the automatically generated FMEAs could be performed. Now only which aids that are available can be compared and not the quality of them.

The aids described above are provided by all the tools. However, one aid that has only been found in TEAMS is the feedback loop, described in Section 4.1.5.

5.5 Compatibleness

A comparison of the compatibleness of the three tools is here presented in Table 12. The table lists items that one or more tools are compatible with in the topmost row and the three tools are found in the first column. An X in a box means that the tool at hand offer support. An empty box represents that no information has been found regarding the support in that case.

Table 12. Compliance for the three tools

	XML	Diagnostic reports	Matlab/Simulink	Excel	CAD tools	OBDC	C code	Net lists
TEAMS	X	X	X	X		X		X
eXpress	X	X		X	X			X
Raz'r	X	X	X				X	X

As can be seen in Table 12 all three tools use XML for information exchange. XML support is important since XML is a commonly used format for information storage and information exchange. All three tools are also able to generate diagnostic reports and the formats in which diagnostic reports can be generated in respective tool can be seen in Table 13.

Table 13. Formats in which diagnostic reports can be generated for the three tools

	XML	HTML	PDF	AI-ESTATE	RTF	Doc	ID3	C code
TEAMS	X	X	X	X				
eXpress	X		X		X	X		
Raz'r	X	X					X	X

Further on in Table 12 we can see that both QSI (TEAMS) and OCC'M (Raz'r) have capability of importing models from Matlab/Simulink. This capability is good since the use of Simulink is widespread in the engineering community. Another format, which especially eXpress use as the key types of external data sources, is the Microsoft Excel spreadsheet. The use of Excel is also widespread. In Section 4.1.7 it was described that Excel spreadsheets were imported into TEAMS using an import tool in a study.

eXpress uses the standard EDAXML and the following CAD tools are listed as compatible with eXpress: Cadence, Mentor DesignArchitect, Viewlogic/Innoveda, PADS, P-CAD, and OrCAD. The support of a CAD tool is important if there are existing designs that preferably could be imported into the tool of interest.

TEAMS models can be exported to databases that are ODBC-compliant. This feature makes it possible to access data bases regardless of what data base management system or operating system that is used. Raz'r can generate diagnosis algorithms in C code which is a plus if there is a desire to run C code on an on board computer. Finally in Table 12 we can see that all three tools have import functions for importing netlists. An import function like this would ease the design process if there are such netlists available and hence they would not have to be remodelled.

5.6 Libraries

eXpress and TEAMS come with some preconstructed example models which can provide information and tips about how to use these tools. In TEAMS there is an automobile that has been modelled and in eXpress the examples include a 1553 data bus, an altimeter, an amplifier, and a hydraulic flow example. Raz'r does not automatically come with any basic library as it is not normally sold without consulting, however OCC'M does have various component libraries developed in previous projects and they are reused on a case by case basis [31].

5.7 Disciplines of Modelling

The disciplines of modelling that are supported by each tool are displayed in Table 14.

Table 14. Disciplines in which can be modelled in the three tools

	Electrical	Mechanical	Hydraulic	Chemical processes	Software	Pneumatics
TEAMS	X	X	X	X		
eXpress	X	X	X	X	X	
Raz'r	X		X			X

In this table we can see that all three tools support electrical and hydraulic modelling. eXpress is the tool that supports the most disciplines.

5.8 Modifiability

Modification of a model in eXpress is done by exporting the model topology to Excel spreadsheets, modify the model there, and then importing it into eXpress again. This seems to be a good solution for making updates of models effective. Updates in Raz'r are, according to the producer, made automatically from structural information stored in the tool. In TEAMS, TEAMS-KB provides capabilities to update component reliability, component failure rate, repair costs, and repair times as maintenance and repair data become available.

6 Conclusions

In this chapter the work presented in the preceding chapters are summarized and the conclusions are stated together with ideas for future work.

6.1 Discussion

The aim of this thesis was to find and evaluate diagnostic tools. In Chapter 3 it was described how the search for such tools was done. The search revealed that no tool on the market had a dominating position. From the found tools three of them were selected for further evaluation. These were selected since they were expected to provide the diagnostic systems that best utilize the available information about the system. In Chapter 4 these three tools were analysed. The analysis can be seen as constituted by two different aspects, one focusing on the diagnostic methods with which each tool creates diagnostic systems, the other focusing on practical details that determine the usability of each tool. Chapter 5 provides a comparison of the three tools in terms of the topics analysed in Chapter 4.

The analysis revealed both similarities and differences amongst the tools. For example TEAMS and eXpress use a dependency based model type, whilst Raz'r uses qualitative modelling. Also the inference algorithms offer differences where once again Raz'r differs from the other two. The inference algorithm in Raz'r has the capability to utilize information from models of failure modes. The other two utilize only information about dependencies between components for each failure mode. eXpress here benefits from having the capability to define several fault modes while TEAMS have only two.

The different programs each have its strengths. TEAMS seems built on a solid theoretical foundation and papers from QSI often appear in conference proceedings. QSI also offer a number of different software to cover large parts of the diagnostic needs. DSI often emphasise the possibilities to use eXpress in an early stage of the development of the system to be diagnosed. The advantage of this is to consider diagnostic aspects when designing the system and by this enabling for better diagnostic capabilities. OCC'M also has contributed with several papers in conferences. Raz'r benefits from the capabilities to handle state variables and using the information available about failure modes to further enhance the isolation.

A similarity is that all three tools use XML for storing models although the methods used for defining the allowed elements and attributes of the XML files differ. eXpress and Raz'r use XML Schema while TEAMS uses DTD. The DTD format only specifies the names of elements and attributes whereas in the XML Schema we can also control values of attributes. This extra control makes the XML Schema more powerful. The fact that XML Schemas are written themselves in XML, unlike DTDs, eases work since another language is not needed. The XML Schema used in eXpress, Diag-ML, is a somewhat widespread standard for diagnostics capture and has some support within the industry. Rodon for example supports Diag-ML.

On a note on customer relations for the companies providing the evaluated tools we can conclude that the provider of eXpress, DSI, as well as the provider of Raz'r,

OCC'M, have been of great assistance to this project. They both have answered questions and provided us with information which has been very useful.

A final conclusion that can be drawn is that the information available concerning the system that is to be diagnosed should guide the choice of tool. Raz'r has better fault isolation capability if there is knowledge available about the behaviour of a system when it is in a failure mode. TEAMS and eXpress on the other hand require less knowledge of the system and thus the creation of the diagnostic system can begin earlier. They are also more intended to handle information gained from built in self tests of components. From this information the tool recommended to use is TEAMS or eXpress for systems built up mainly of components with built in self tests or Raz'r for systems where the components do not have built in self tests.

6.2 Future Work

Future work following after this thesis might be to further explore one or more of the three evaluated tools that is of further interest. A comparison of TEAMS and eXpress could be useful since these tools seem to be more similar to each other compared to Raz'r. The effect of the limited number of failure modes in TEAMS compared to the larger number of failure modes in eXpress could be studied. In such a study a more practical approach can be used where a system is modelled and studied diagnostically in the tool. The same study could be made with Raz'r and Rodon since they seem more similar. A system that is to be modelled in such a study could be of the kind that would be representative of what the tool would be used for in an everyday use. Such a study, concentrating at only one tool at a time, might give a more complete picture of how the tool works, since this has been somewhat of an introduction to the three tools TEAMS, eXpress, and Raz'r.

Another suggestion for possible future work could lie in examining the strong candidates that did not make the cut described in Section 3.5.

In the problem description it was also included to map standards within the diagnostic field. The study revealed no widely accepted standard within the diagnostic field. One standard that was found in TEAMS is AI-ESTATE which was presented in Section 2.6. AI-ESTATE is a standard from IEEE that is used for storing and exchanging diagnostic knowledge and could certainly be of interest for further exploration.

Yet another interesting issue to further explore is how the computational requirements correlate to the size of a system for different modelling types and inference algorithms.

Finally one important aspect when choosing a tool is to secure that the tool has the ability to meet future requirements. This makes it important to insure that the company providing the tool has the prerequisites to develop the tool so it meets those demands. This could be done in a corresponding study.

7 References

- [1] Personal reference at DSI
- [2] Eric Gould, *Modelling it both ways: Hybrid diagnostic modelling and its application to hierarchical system designs*, DSI International, 2004 IEEE
- [3] B Long et al, *System Testability Analysis for Complex Electronic Devices Based on Multisignal Model*, Journal of Physics: Conference Series 48, 2006
- [4] <http://www.diag-ml.com/>, Accessible: 2007-11-13
- [5] <http://www.diag-ml.com/Detail.aspx?Detail=Consortium>, Accessible: 2007-11-13
- [6] Oskar Dressler, Peter Struss, *Generating instead of programming Diagnostics*, <http://www.occm.de/GeneratingOnBoardDiagnostics.pdf>, Accessible: 2007-11-13
- [7] P. Struss, A. Malik, M. Sachenbacher, *Qualitative modelling is the key to automated diagnosis*, 6th International Workshop on Principles of Diagnosis (DX-95), Goslar, Germany, pp. 99-106, 1995
- [8] P. Struss, *Automated Abstraction of Numerical Simulation Models- Theory and Practical Experience*, Proceedings of the Workshop on Algebraic Models of Reasoning, Ki-2003 Hamburg, Germany, Sept. 16, 2003
- [9] <http://www.diag-ml.com/Detail.aspx?Detail=Extensible>, Accessible: 2007-11-13
- [10] DSI Homepage, [http://www.dsiintl.com/WebLogic/Products.aspx?Page=Interoperability/Dia gML.ascx](http://www.dsiintl.com/WebLogic/Products.aspx?Page=Interoperability/Dia%20gML.ascx), Accessible: 2007-11-19
- [11] www.ne.se, [XML.Nationalencyklopedin](http://www.ne.se/XML.Nationalencyklopedin), 2007, Nationalencyklopedins internetjänst. http://www.ne.se.jsp/search/article.jsp?i_art_id=348461 (In Swedish) Accessible: 2007-11-19
- [12] http://www.w3c.se/resources/office/translations/XML-in-10-points_sw.html, Accessible: 2007-11-19
- [13] http://www.w3schools.com/xml/xml_syntax.asp
- [14] Jianhui Luo, Haiying Tu, Krishna Pattipati, Liu Qiao, Shunsuke Chigusa, *Graphical models for diagnosis knowledge representation and inference*, Autotestcon, 2005. IEEE, 26-29 Sept. 2005, Page(s): 483- 489
- [15] <http://xml.silmaril.ie/basics/whatfor/>, Accessible: 2007-11-22

- [16] Somnath Deb, Krishna R. Pattipati, Vijay Raghavan, Mojdeh Shakeri, Roshan Shrestha, *Multi-Signal Flow Graphs: A Novel Approach for System Testability Analysis and Fault Diagnosis*, AUTOTESTCON '94. IEEE Systems Readiness Technology Conference. 'Cost Effective Support Into the Next Century', Conference Proceedings, 20-22 Sep 1994, Page(s): 361-373
- [17] S. Deb, A. Mathur, P.K. Willett, K.R. Pattipati, *Decentralized real-time monitoring and diagnosis*, Systems, Man, and Cybernetics, 1998 IEEE International Conference on, Volume: 3, 11-14 Oct 1998, Page(s): 2998-3003 vol.3
- [18] <http://www.w3schools.com/dtd/default.asp>, Accessible: 2007-11-26
- [19] <http://www.w3.org/TR/2000/REC-xml-20001006>, Accessible: 2007-11-26
- [20] François E. Cellier, *Qualitative modelling and simulation: promise or illusion* Winter Simulation Conference archive, Proceedings of the 23rd conference on, Phoenix, Arizona, United States, Pages: 1086 – 1090, Year of Publication: 1991, ISBN:0-7803-0181-1
- [21] Somnath Deb, Venkata N. Malepati, Michel D. Paquet, Baban Baliga, *Validation of a COTS EHM Solution for the JSF Program*, Aerospace Conference, IEEE 2006, ISBN: 0-7803-9546-8
- [22] TEAMS Homepage, <http://www.teamqsi.com/KB.html>, Accessible: 2007-11-28
- [23] Kevin Cavanaugh, *An Integrated Diagnostics Virtual Test Bench for Life Cycle Support*, Proceedings of IEEE Aerospace Conference, Big Sky, Montana, 2001
- [24] TEAMS 9.x Manual, Section 2.3.3
- [25] IEEE Std 1232-2002, IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE), 2002
- [26] Peter Struss, Oskar Dressler, *"Physical negation"- Integrating Fault Models into the General Diagnostic Engine*, in *Readings in model-based diagnosis*, editors Walter Hamscher, Luca Console, Johan de Kleer, pages 153-158, 1992
- [27] John W. Sheppard, Timothy J. Wilmering, *Recent Advances in IEEE Standards for Diagnosis and Diagnostic Maturation*, 2006 IEEE Aerospace Conference, 2006
- [28] Daniel Andersson, Patrik Sköld, *Evaluation of a diagnostic tool for use during system development and operation*, 2007 Linköping University, Department of Electrical Engineering, Master thesis number: 2007-3916
- [29] E-mail conversation with Oskar Dressler, OCC'M, 2007-10-16

- [30] Ian Gilfillan, *Introduction to Relational Databases*, 2002 DatabaseJournal, http://www.databasejournal.com/sql/etc/article.php/26861_1469521_1, Accessible 2007-01-14
- [31] E-mail conversation with Oskar Dressler, OCC'M 2007-10-24
- [32] Peter Struss, Oskar Dressler, *A toolbox integrating model-based diagnosability analysis and automated generation of diagnostics*, Proceedings of the 14th International Workshop on Principles of Diagnosis (DX'03) 2003
- [33] Srđan Bjelic, Sofie Lindberg, *Model Based Reasoning from a Vehicle Perspective - Evaluation of the Software Tools Raz'r and Rodon*, 2007 Chalmers University of Technology, Department of Signals and Systems, Master thesis number: Ex007/2007
- [34] <http://diagnostician.vsecorp.com/VSE%20Corporation%20-%20Diagnostician%20Introduction.htm>
Accessible 2008-01-21
- [35] TEAMS Homepage, <http://www.teamqsi.com/TEAMS.html>, Accessible 2008-01-22
- [36] TEAMS Homepage, <http://www.teamqsi.com/RT.html>, Accessible 2008-01-22
- [37] TEAMS Homepage, <http://www.teamqsi.com/RDS.html>, Accessible 2008-01-22
- [38] TEAMS Homepage, <http://www.teamqsi.com/TEAMATE.html>, Accessible 2008-01-22
- [39] <http://ic.arc.nasa.gov/projects/L2/doc//>, Accessible 2008-01-22
- [40] DSI Homepage,
<http://www.dsiintl.com/WebLogic/Products.aspx?Page=eXpress/WhatIsExpress.ascx>, Accessible 2008-01-23
- [41] DSI Homepage,
<http://www.dsiintl.com/WebLogic/Products.aspx?Page=eXpress/HowDoesItWork.ascx>, Accessible 2008-01-23

- [42] <http://www.aerospaceonline.com/Content/ProductShowcase/product.asp?DocID=c97261c5-d115-11d4-8c88-009027de0829&VNETCOOKIE=NO>, Accessible 2008-01-25
- [43] Animated tutorial: DFT Feedback on Model, http://www.teamqsi.com/teams/demopage/DFT_Feedback_On_Model_viewlet/DFT_Feedback_On_Model_viewlet_swf.html Accessible 2008-01-25.
- [44] TEAMS 9.X Manual, Section 2.2.1
- [45] Somnath Deb, Sudipto Ghoshal, Amit Mathur, Roshan Shrestha and Krishna R. Pattipati, *Multisignal Modelling for Diagnosis, FMECA, and Reliability*, Proceedings of IEEE SMC 1998
- [46] Eric Gould, Danver Hartop, Erick Lee, Ion A. Neag, Mark Wilson, *DiagML - An Interoperability Platform for Test and Diagnostics Software*, AUTOTESTCON Proceedings, 2002
- [47] eXpress Online Help, Section: Importing: Overview
- [48] eXpress Online Help, Section: Design Reports – Spreadsheet Topology Export
- [49] eXpress Online Help, Section: Data Source Administration - Spreadsheet Topology Import
- [50] Claudia Picardi, Luca Console, Frederic Berger, Jan Breeman, Tony Kanakis, Jeroen Moelands,, Stephan Collas, Emmanuel Arbaretier, Nino De Domenico, Ermanno Girardelli, Oskar Dressler, Peter Struss, Benjamin Zilbermann, *AUTAS: a tool for supporting FMECA generation in aeronautic systems*, Proceeding of the 16th European Conference on Artificial Intelligence 2004
- [51] E-mail conversation with Oskar Dressler, OCC’M, 2007-12-03
- [52] DSI Homepage, <http://www.dsiintl.com/kb/default.aspx>, Accessible 2008-01-31
- [53] DSI Homepage, http://www.dsiintl.com/WebLogic/Products.aspx?Page=EDAImportModule_new.aspx, Accessible 2008-01-31
- [54] Amit Mathur, Sudipto Ghoshal, Deepak Haste, Charles Domagala, Roshan Shrestha, Venkatesulu Malepati, Krishna Pattipati, *An Integrated Support System for Rotorcraft Health Management and Maintenance*, IEEE Aerospace Conference Proceedings 2000

-
- [55] DSI Homepage,
<http://www.dsiintl.com/WebLogic/Products.aspx?Page=eXpress/SensorOptimization.ascx>, Accessible 2008-02-10
- [56] DSI Homepage,
<http://www.dsiintl.com/WebLogic/Products.aspx?Page=DiagnosticModule.ascx>, Accessible 2008-02-10
- [57] Personal reference at DSI
- [58] DSI Homepage,
<http://www.dsiintl.com/WebLogic/Products.aspx?Page=eXpress/MultiDisciplineSupport.ascx>, Accessible 2008-02-10
- [59] Personal reference at OCC'M
- [60] Mattias Nyberg, Erik Frisk, *Model Based Diagnosis of Technical Processes*, Linköping, 2006
- [61] E-mail conversation with Oskar Dressler, 2007-10-22
- [62] E-mail conversation with Oskar Dressler, 2007-11-13
- [63] Sudipto Ghoshal, Somnath Deb, *Implementation Strategy for AI-ESTATE compliance in the Remote Diagnosis Server (RDS) framework*, AUTOTESTCON Proceedings, 2001. IEEE Systems Readiness Technology Conference, 2001
- [64] TEAMS 9.x Manual, Section 1.4
- [65] <http://www.dsiintl.com/Resources/Brochures/eXpressFeatures.pdf>, Accessible 2008-02-18
- [66] eXpress Online Help, Section: Fault Isolation Report : Options Dialog
- [67] eXpress Online Help, Section: Tests - Overview
- [68] QSI Homepage,
http://www.teamqsi.com/teams/demopage/Diagnostic_Tree_viewlet/Diagnostic_Tree_viewlet_swf.html, Accessible 2008-02-22
- [69] eXpress Online Help, Section: Diagnostic Test Selection Criteria - Diagnostic Algorithms
- [70] <http://www.w3schools.com/schema/default.asp>, Accessible 2008-02-25
- [71] www.ne.se, VHDL. Nationalencyklopedin. 2007. Nationalencyklopedins internetjänst, http://www.ne.se/jsp/search/article.jsp?i_art_id=342037 (In Swedish) Accessible: 2008-02-27

- [72] http://www.w3schools.com/schema/schema_howto.asp, Accessible 2008-03-05
- [73] Military Standard, *Procedures for Performing a Failure Mode, Effects and Criticality Analysis*, MIL-STD-1629A, 24 NOVEMBER 1980
- [74] Amit Mathur, Somnath Deb, and Krishna R Pattipati, *Modeling and Real-Time Diagnostics in TEAMS-RT*, Proceedings of the American Control Conference, Philadelphia, Pennsylvania, June 1998
- [75] Lennart Ljung, Torkel Glad, *Modellbygge och simulering*, Lund 2004
- [76] Nolan, M., Giordano, J.P., *Use of adaptive model-based reasoning for embedded diagnostics and redundancy management for fault tolerant systems*, AUTOTESTCON '97 IEEE Autotestcon Proceedings, 1997

Appendix A - Search word list

To ease and structure the search for suitable alternatives a list of search words was constructed:

AI
aircraft
aviation
Bayesian networks
Diagnostics
failure
fault
FDI (Fault detection and isolation)
FMEA
FMECA
health management system
health monitoring
isolation
logic
Model based diagnostics
Model Based Reasoning
on-board diagnostics
prognostics
real-time diagnostics
Software
Tool

Appendix B - Found tools

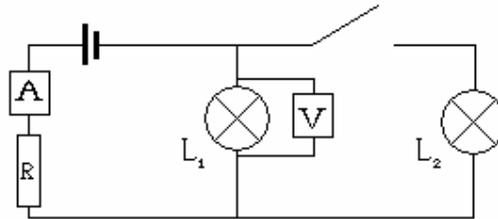
Here follows a list of the tools found in the search. First the name of the product is listed, then the name of the company supplying the software, next the foremost demand in 3.4.X which was not met and last a short comment of the tools is supplied.

Product name	Company name	Foremost demand not met 3.4.X	Comment
ACAMS	Arinc	3	Not enough information.
Adams	MCS Software	1	No direct affiliation with diagnostics.
AirMan	Airbus	1	Focused on air lines operating airbus planes
Autosoft	Mecel	3	Did not answer e-mail enquiry.
BayesiaLab	Bayesia	1	Not specialized in diagnostics.
CDG	Boeing	1	Not a diagnostics tool
Dexter	Macsea	1	Specialized in marine use.
Diagnostic Profiler/Diagnostician	VSE	3	Not enough information.
DSI Express	DSI	Chosen	
DTOOL/CNETS	Adnan Darwiche/Rockwell	3	Not enough information. The tools had been sold and their whereabouts were not known by previous owner.
FOCA	Simauthor	3	Program to simulate flights from recorded data. Mostly usable to evaluate pilots.
G2	Gensym	5	No clear overall solution.
Goodrich	Vehicle Health Management System	3	Demands hardware and other software from Goodrich to operate
Hugin Expert	Hugin Expert	1	Not specialized in diagnostics.
HUMS	GE	2	Includes hardware.
IHUMS	Meggitt Avionics	2	Hardware based
i-Trend / i.Predict / ICEMS	Scientific Monitoring inc	1	Provides only web based tools for diagnostics.
Livingstone 2	NASA	4	A research project rather than a product.
Maintenix	Mxi Technologies	3	Handles the logistics connected to maintenance.
Matlab/Simulink	Mathworks	1	Not specialized in diagnostics.
Model Wizard	Integrated diagnostics systems	1	Deals with finance.
Numerous products	Vector	4	Do not mention a specific product.
Numerous products	Impact technology	5	No clear overall solution.

Numerous products	IAC		2	Includes hardware.
PM2TM	Prime Photonics		3	Not enough information.
Raz'r	OCC'M		Chosen	
Rodon	Sörman		7	There already is a report written on Rodon.
SCORE	DDC-I		1	Is a software development tool rather than a diagnostic tool.
Sentient Awareness	Situation Management Sciences Inc.		3	Shortage of information and they did also have some hardware.
SFIM & AGS	ACMS GSE Sagem Avionics		3	Not enough information
Shield mm	Smartsignal		1	Deals with data mining rather than diagnostics
SIDIS	Siemens		3	Did not reply to e-mail enquiry.
Spotlight	Casebank		6	Stores cases and searches through a data base of them to recognize symptoms.
TEAMS	Qualtech Systems		Chosen	
Unspecified name	MTC Technologies		4	Do not mention a specific product.
Unspecified name	Foster Miller		4	Do not mention a specific product.
Unspecified name	Interface & Control Systems Inc		5	No clear overall solution.
Unspecified name	TechnoSciences		4	Works with predictive maintenance operations scheduling, modelling, and control design of engineering systems but does not mention a specific product.
VPS-Micro	VEXTEC		1	Diagnostics for fatigue in materials.

Appendix C - Sample Systems

Electric circuit



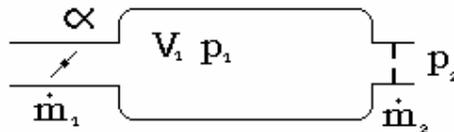
A light circuit with the observations:
 Light 1 is lit, light 2 is lit, measured voltage, measured current, and wanted switch position.

Possible faults:

- One or both lights are broken
- Bias fault in the voltage meter
- The resistance is short-circuited or cut
- Switch stuck open or closed

Hidden faults (e.g. one can't tell if L_2 is broken when the switch is stuck open) should be indicated if possible. Multiple faults should be detected and isolated if possible.

Air intake : Continuous system with feedback

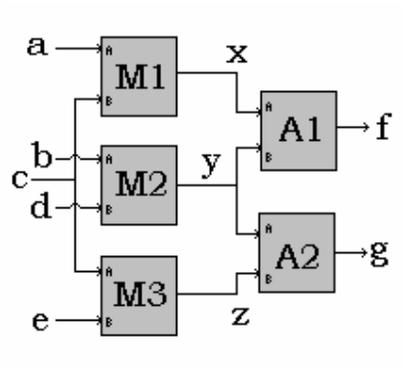


Modified model of an air intake where one measures p_2 and controls alpha. We want to control p_1 through the feedback $\dot{m}_1 = F(\alpha)$ where $F(\alpha)$ is known and $\alpha \in \left[0, \frac{\pi}{2}\right]$ where $\alpha = 0$ represents a completely closed intake. The sensor has a white noise disturbance.

Possible faults:

- Leakage in V_1 , $\dot{m}_1 = K(p_1 - p_2)$ where \dot{m}_1 is the mass flow through the leakage hole
- Sensor fault, $y = p_2 + f_y$ where f_y is a constant
- Actuator fault, the throttle can get stuck in some position

Polybox



"Classic" polybox, should be modelled both with c known and c unknown. a, b, d, e, f and g are known. Faults can occur in the multipliers and in the adders but are either working or A-short-circuit (output = port A of the inputs) or B-short-circuit (output = port B of the inputs). Multiple faults should be handled if possible.