OULUN AMMATTIKORKEAKOULU

Arto Drees

AUTOMATED DIAGNOSTICS OF ELECTRONIC SYSTEMS

AUTOMATED DIAGNOSTICS OF ELECTRONIC SYSTEMS

Arto Drees
Bachelor´s Thesis
Fall 2014
Information Technology and
Telecommunications
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Information Technology and Telecommunications, option of Wireless Devices

Author: Arto Drees
Title of thesis: Automated Diagnostics of Electronic Systems
Supervisors: Tuomo Tikkanen & Ville Nurkkala
Term and year of submission: Fall 2014 Pages: 38 + 2 appendices

This thesis was commissioned by Nokia Networks as a part of a wider ongoing quality project. The main objective for this thesis and the quality project was to improve diagnostic accuracy on a certain base station product by targeting the most misdiagnosed faults and to reduce unnecessary component replacement.

To achieve the objective, an early version of an automated diagnostic tool was developed. The tool was developed using Microsoft Visual Studio and C# programming language. The tool uses diagnostic databases to hold diagnostic information. The diagnostic databases were implemented using XML (Extensible Markup Language). The diagnostic databases implement fault models and rule based diagnostics to troubleshoot target products.

Two pilot programs were launched in order to verify the functionality of the tool and measure improvements on diagnostic accuracy. The results of the pilots were encouraging; the rate of failed diagnoses decreased by 88%.

The success of the pilot programs and the ease of use of the tool sparked interest to it, and it was decided to be taken into global use in system module repair of Nokia Networks base stations. Further development is, however, still needed to extend the coverage of the tool and to add new features.

Keywords:
diagnostics, automation, XML, C#

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, Langattomien laitteiden suuntautumisvaihtoehto

Tekijä: Arto Drees
Opinnäytetyön nimi: Automated Diagnostics of Electronic Systems
Työn ohjaajat: Tuomo Tikkanen & Ville Nurkkala
Työn valmistumislukukausi ja -vuosi: Syksy 2014 Sivumäärä: 38 + 2 liitettä

Tämän opinnäytetyön tilaajana toimi Nokia Networks. Työ tehtiin osaksi laajempaa laadunparantamisprojektia. Päätavoitteena oli parantaa diagnostiikkaa sekä vähentää väärien diagnoosien määrää tietyillä tukiasematuotteilla kohdentamalla työ useimmin väärin diagnosoiduille komponenteille.

Tavoitteet saavutettiin kehittämällä varhainen versio automatisoidusta diagnostiikkatyökalusta. Työkalu kehitettiin Microsoft Visual Studio-kehitysympäristössä C#-ohjelmointikielellä. Työkalu käyttää diagnostiikkatietokantaa, joka kehitettiin XML(Extensible Markup Language)-kielellä. Tietokannalla toteutetaan vikamallinnusta ja sääntöihin perustuvaa diagnostiikkaa juurisyyn löytämiseksi.

Kaksi pilottihanketta käynnistettiin varmistamaan työkalun toimivuus ja mittaamaan onnistuneiden diagnoosien määrän kehitystä. Pilottihankkeiden tulokset olivat rohkaisevia; väärien diagnoosien määrä laski 88 %.

Pilottihankkeiden hyvät tulokset ja työkalun helppokäyttöisyydestä saatu palaute johtivat siihen, että työkalu päätettiin ottaa käyttöön globaalisti tilaajan valmistamien systeemimoduulien korjauskäytössä. Työkalu vaatii silti vielä jatkokehitystä, erityisesti kattavuuden ja vaillinaisten ominaisuuksien osalta.

Asiasanat:
diagnostiikka, automaatio, XML, C#

CONTENTS

GLOSSARY

ANN	Artificial Neural Network
CAD	Computer Assisted Design
CVI	C for Virtual Instruments
DOM	Document Object Model
DUT	Device Under Test
GUI	Graphical User Interface
IEEE	The Institute of Electrical and Electronics Engineers
PC	Personal Computer
R&D	Research and Development
TSI	Troubleshooting Instruction(s)
W3C	The World Wide Web Consortium
XML	Extensible Markup Language

1 INTRODUCTION

This thesis was commissioned by Nokia Networks as a part of a wider ongoing quality project. Nokia Networks is a business unit of Nokia Corporation. Nokia Corporation was founded in 1871 and the global headquarters are located in Espoo, Finland. As of September 2014 Nokia Corporation employs 59035 employees of which 51980 (88%) work for Nokia Networks (1). The Oulu site of Nokia Networks serves as a R&D site and ramp-up factory for new products.

The main objective for this thesis was to improve diagnostic accuracy on a certain base station product by targeting some of the most misdiagnosed faults and to reduce unnecessary component replacement. To achieve the objective, an early version of an automated diagnostic tool was developed and a draft of a SRS (Software Requirement Specification) document was created to guide future development of the tool. The existing troubleshooting instructions were converted to a database and a set of diagnostic information which is accessed by the tool. The SRS follows guidelines set by IEEE document 830-1998: Recommended Practice for Software Requirements Specifications (2).

The tool solves some problems in the preceding troubleshooting procedure, mainly in the TSI (Troubleshooting Instructions) -document. Such a document can easily grow up to hundreds of pages and the relevant information for a single fault can be spread out to several sections. In such cases the operator can accidentally or intentionally skip some measurements. The tool hides all the irrelevant troubleshooting information and guides the operator trough only the relevant measurements. The repair operator can see the current measurement points at a glance and observe the flowchart of the higher level measurement. This encourages the operator to actually go through all the necessary steps.

The tool was developed using Microsoft Visual Studio and Visual C# -programming language. This development environment was chosen because it enables fast and easy Windows based GUI (Graphical User Interface) development while still maintaining interoperability with the more commonly used National Instruments LabWindows/CVI and TestStand -development environments and their libraries.

2 DIAGNOSTIC METHODS

A diagnostic process of any electronic system usually follows a set of three fundamental tasks. These tasks are:

- Hypothesis generation
- Hypothesis testing
- Hypothesis discrimination

Given that a system is undergoing a diagnostic process, it must have at least one symptom. Using this symptom as a starting point, a list of hypotheses can be generated. These hypotheses are then tested by checking if they alone can account for all (and only) observed symptoms. After the hypotheses have been tested, and usually some eliminated in the process, the initial observations cannot provide any additional information. At this stage it is necessary to start gathering new information that allows further discrimination of the remaining hypotheses. The last step is executed until only a single hypothesis is left. (3.)

Diagnostic systems can be categorized by their approach to execute the fundamental tasks. Some of these approaches are:

- Rule-based approaches
- Model-based approaches
- Machine learning approaches
- Hybrid approaches
- Others

(4.)

2.1 Rule-based approaches

Rule-based approaches are the oldest and simplest form of system diagnosis. A rule-based diagnostic procedure takes information about the problem and applies a set of rules on the information. This generates more information or actions, upon which another set of rules is applied. The procedure is repeated iteratively until a solution is found. For most modern electronic systems the set of rules can quickly grow up to hundreds or thousands of rules. In addition, even small changes in the system can cause large deviation in the rule-set, which then needs to be rebuilt. However, because the process is simple and clearly defined, it is easy to follow. (4.)

The diagnostic process of a rule-based approach can be represented with a decision tree as in FIGURE 1. In the diagram; iterative steps are represented with different colors and each step applies its own rule set on the previous data.

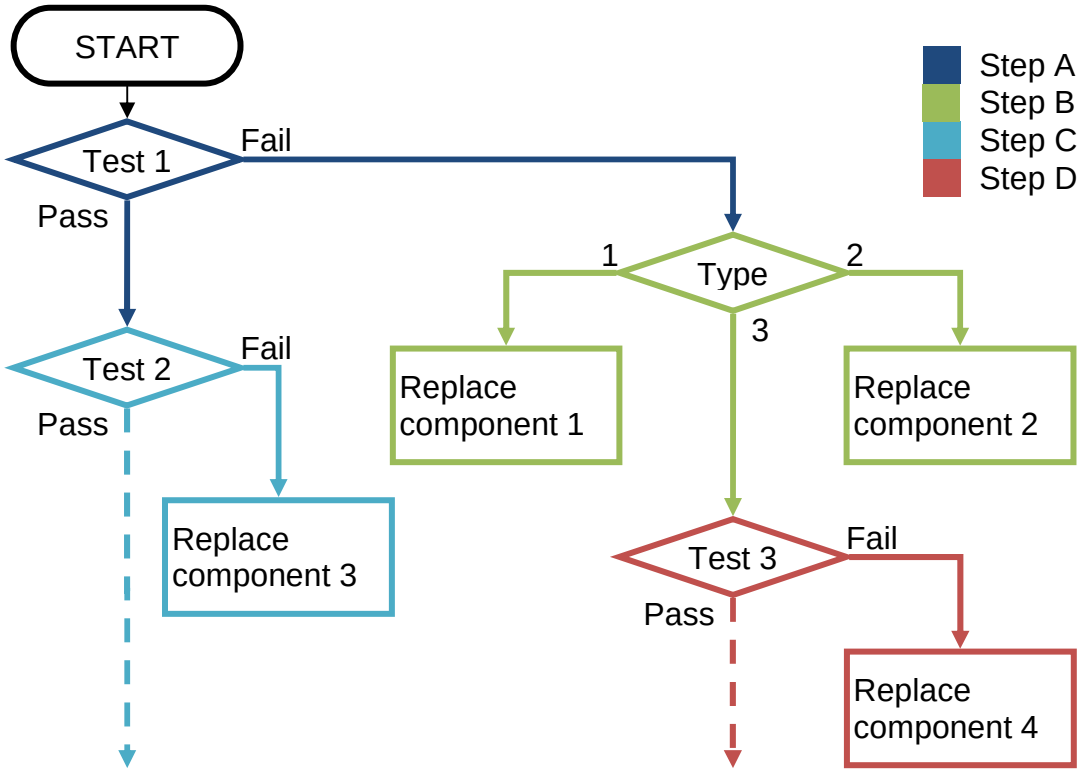

FIGURE 1. Simple decision tree for a rule-based diagnostic approach

2.2 Model-based approaches

Model-based approaches use a model to approximately represent the system. The model can be constructed in various ways, including fault models, structural and behavior models. It is usually necessary to construct the model hierarchically, so that one top level model contains several sub level models representing parts of the diagnosable system.

2.2.1 Fault Models

Fault models model the system by anticipating various types of faults and their resulting symptoms. This model performs the first two fundamental tasks; hypothesis generation and testing. When using a fault model, only the modeled faults can be diagnosed. The model does not contain any information about the actual system, but only shows what happens when a modeled fault occurs. Fault models can be very accurate when dealing with system level blocks and simple combinational logic. However, they do not work particularly well for complex circuits or when a vast quantity of different failures can occur. Example fault model is illustrated in TABLE 1.

TABLE 1. Example fault model

	Test 1 Fail type 1	**Test 1** Fail type 2	**Test 1** Fail type 3	**Test 2** Fail	**Test 3** Fail
Comp. 1 - Internal fault	X			X	X
Comp. 2 - Internal fault		X			X
Comp. 4 - No power			X		
Comp. 4 - Output shorted			X		
Comp. 4 - Internal fault			X		X
Wiring fault: Comp. 1 - 2	X				X
Wiring fault: Comp. 3 - 4				X	

2.2.2 Structural and behavioral models

A model based on structure and behavior consists of two different representations of the system, structural and behavioral. The structural representation is essentially a list of all the components or functional blocks and their connections. The behavioral representation describes the behavior of individual components or functional blocks. The advantage of these models is that they can be straightforwardly generated from CAD data. (4.)

Given a specific input, the model can be used to predict the output of the system. If a discrepancy is found between the predicted and observed behavior, the model can quickly narrow the hypothesis list to those components that affect the point of discrepancy. This hypothesis list can be further discriminated by using a so called guided probe -method. Essentially the method starts at the observed discrepancy and follows the causal chain of components following the discrepancies between observations and predictions. Once a discrepancy is no more found, taking one step back reveals the cause of the failure. The guided probe -method is just one of many possible methods, other methods might account for failure probabilities of different components, finding the optimal probing points to minimize measurement count or testing with different input values to deduce the point of failure. (4.)

An example of the guided probe method in action is illustrated in FIGURE 2. The behavior of the components is described by their names and corresponding simple logical operations. The discrepancy is first observed at *F*, and then followed backwards via *MAX, Z, ADD, Y* and *MIN*. First, *MAX* input *Z* is found incorrect, so it is followed to *ADD*, whose input *Y* is found incorrect. Finally, *MIN* inputs *C* & *X* are both found to be correct. Therefore the component *MIN* is found to be faulty, producing faulty output with correct inputs.

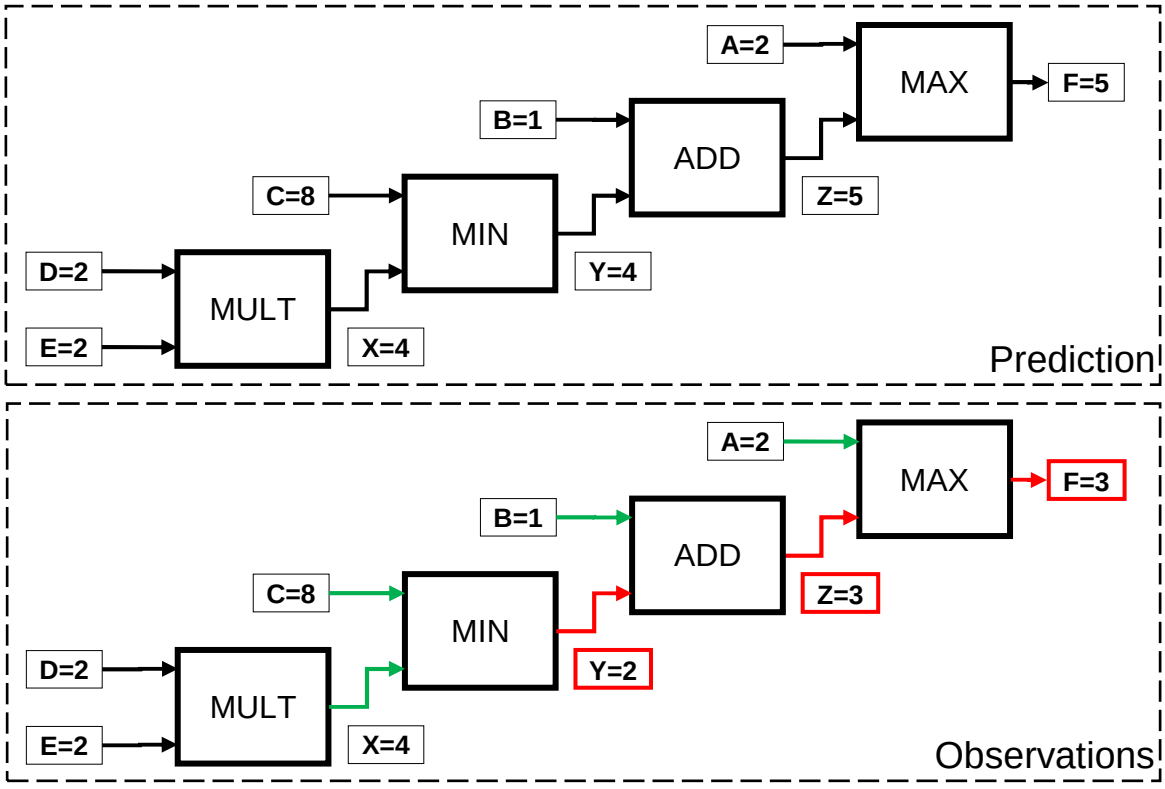

FIGURE 2. Example of a structural and behavioral model in action

2.3 Machine learning approaches

Previously discussed approaches have fixed performance and outcome after initial implementation. A diagnostic system using machine learning can improve itself using information on past or example solutions and their success or failure. This enables continuous improvement on both diagnostic accuracy and time to perform additional observations. The most widely used learning method is case-based reasoning. Other methods include explanation based learning (teaching) and learning from existing data. [4.]

Case based reasoning starts by identifying the symptoms and observations of the current problem. Using this information, a set of similar cases is retrieved from a database. These cases are then further compared and ranked to find the most matching case. The remaining retrieved case is then adapted to suit the new problem and a set of repair actions is proposed. After performing the actions and measuring their success the adapted case is revised if necessary and retained in the database as a new case. One drawback for this method is that the diagnostic system must encapsulate the entire repair process in order to measure the success of repair actions. It also struggles to find solutions for completely new symptoms or novel faults. FIGURE 3 illustrates the concept of case based reasoning.

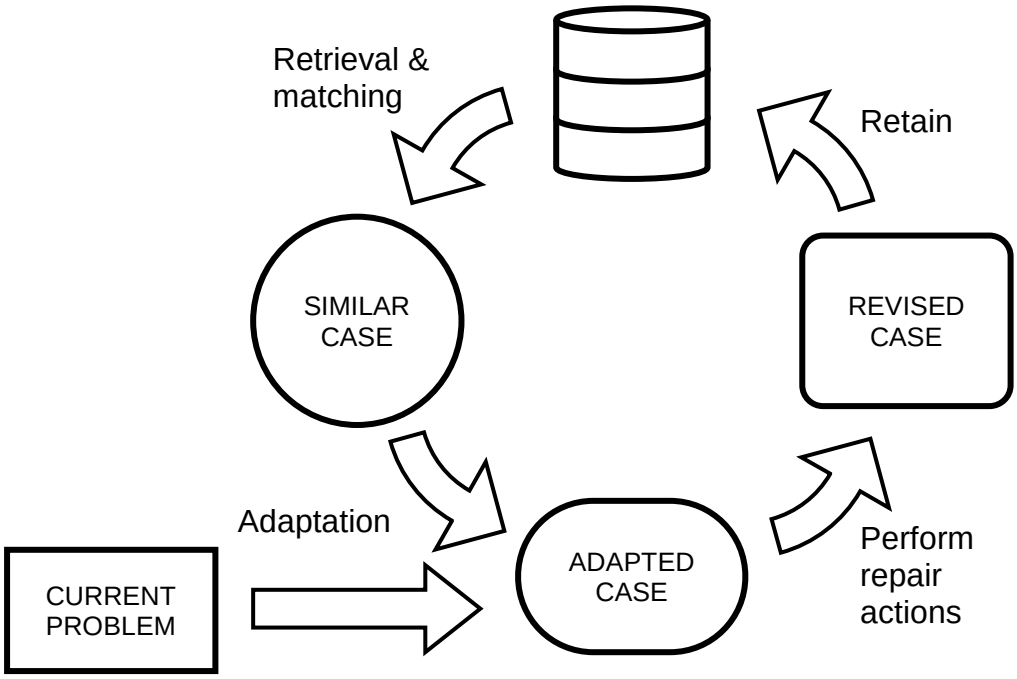

FIGURE 3 Concept of case based reasoning

2.4 Hybrid approaches

A hybrid approach can use any combination of the aforementioned approaches to complement the characteristics of each other. In general, any approach can be combined with machine learning to improve itself. In addition to the methods described earlier, there are many other methods that can be used to enhance diagnostics. These methods include, but are certainly not limited to: Fuzzy logic, Artificial Neural Networks (ANN) or Genetic algorithms. (4.)

An example of a hybrid approach is the diagnostic database described in detail later in this thesis. It uses a combination of a fault model and rule based diagnostics. A fault model is used at system level to generate the initial hypothesis list. This list is then further discriminated by diagnostic measurements. Each measurement is essentially a rule based diagnostic approach for a certain block. The rule based approaches are constructed using knowledge of structure and behavior of the block and implement a sort of fixed guided probe method.

3 XML

XML (Extensible Markup Language) is, as the full form suggests, a simple text-based format for representing structured information, documents, data, configurations and much more. XML is used for sharing structured information between programs, people and computers, both locally and across networks. (5.)

3.1 Structure and syntax

The structure of an XML document as described by a W3C standard; DOM (Document Objet Model), is a tree-like structure; it starts at “the root“, and branches all the way to “the leaves”. Between “the root” and the leaves are nodes. The DOM defines several node types that are shown in TABLE 2. An example xml tree is illustrated in FIGURE 4.

TABLE 2. XML DOM node types (6)

Node Type	Description	Children
Document	Represents the entire document (the root-node of the DOM tree)	Element (max. one), ProcessingInstruction, Comment, DocumentType
DocumentFragment	Represents a "lightweight" Document object, which can hold a portion of a document	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
DocumentType	Provides an interface to the entities defined for the document	None
ProcessingInstruction	Represents a processing instruction	None
EntityReference	Represents an entity reference	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference

Node Type	Description	Children
Element	Represents an element	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
Attribute	Represents an attribute	Text, EntityReference
Text	Represents textual content in an element or attribute	None
CDATASection	Represents a CDATA section in a document (text that will NOT be parsed by a parser)	None
Comment	Represents a comment	None
Entity	Represents an entity	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Notation	Represents a notation declared in the DTD	None

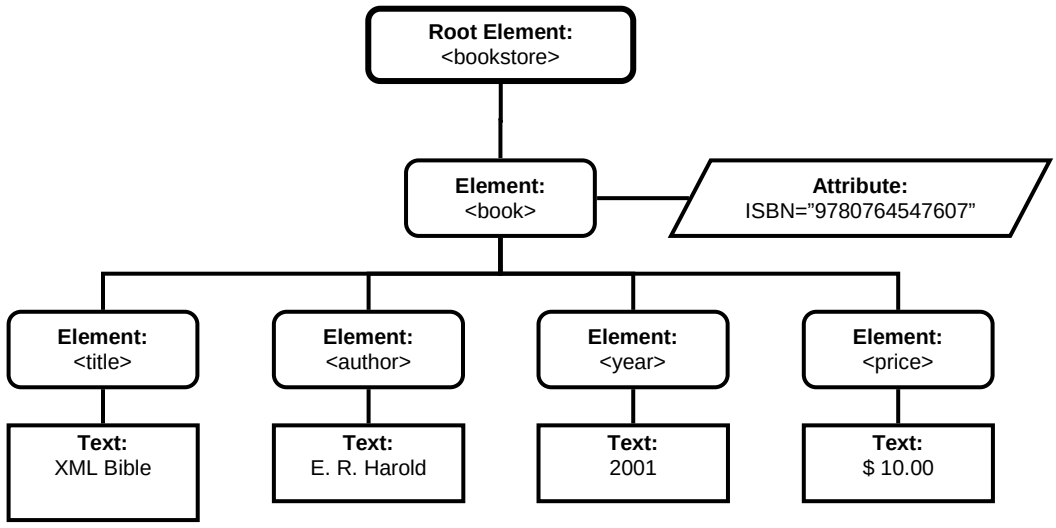

FIGURE 4. Example of XML tree-structure

The XML syntax is also maintained by The World Wide Web Consortium (W3C). There are only a handful of syntax rules in XML. If an xml document follows these rules it is said to be “well-formed”. According to one popular XML guide: www.w3schools.org (7), the list of XML syntax rules is as follows:

- All XML elements must have a closing tag
    - <Example>
      This is an example element
      </Example>
    - <EmptyExample /> ← “self closing” empty element
- XML tags:
    - are case sensitive
    - cannot start with numbers, punctuation or word “xml”
    - (“ **-** “ , “ **.** ” and “ **:** ” should be avoided)
- XML Elements must be properly nested
    - An element must be opened and closed within the same parent
- XML Document must have one root -element
    - The root-element is the parent of all other elements
- Attribute values must be quoted
    - Examples:
      <Element id=”123”>; <Vehicle type=”train”>, <Note date=”30/12/14”>
- Reserved characters are to be replaced with entity references
    - < → < , less than
    - > → > , greater than
    - & → & , ampersand
    - ‘ → ' , aposthrope
    - “ → " , quotation mark
- Comments are expressed within comment tags

- o **<!--** , opens a comment
- o a comment can contain any text, including reserved characters
- o **-->** , closes the comment

- New line -character is stored as LF (ASCII: 10)

The tree structure of the earlier example in FIGURE can be represented in XML as illustrated in FIGURE 5. The first line opens the root element: “bookstore”. In addition to the earlier example this example also lists another book: “Beginning XML Databases”.

```
<bookstore>
    <book ISBN="9780764547607">
        <title >XML Bible</title>
        <author>E. R. Harold</author>
        <year>2005</year>
        <price>30.00</price>
    </book>

    <book ISBN ="9780471791201">
        <title>Beginning XML Databases</title>
        <author>Gavin Powell</author>
        <year>2007</year>
        <price>25.00</price></book>
</bookstore>
```

FIGURE 5. Example of XML representation

3.2 XPath

Often when working with xml-documents, it is necessary to select nodes or elements with certain path, properties, values or relations. XML offers a way to do this by using a special addressing language: XPath. It allows selecting virtually any node or node set from an XML-document.

An XPath expression can consist of one or more of the following: a path expression, predicate(s), wildcards and operators. The path expression determines the initial scope of the selection. This selection can be refined by using predicates or wildcards, or expanded by using the | operator. Predicates are enclosed within square brackets (“[ ]”). TABLE 3 shows some of the most useful XPath expressions using the earlier bookstore -example. The complete and up-to-date syntax for XPath can be found from W3C website (8).

TABLE 3. Useful XPath expressions

Expression	Description
Path expressions	
/	Select from root.
//	Select from anywhere in the document.
.	Select current context node.
..	Select parent of current context node.
nodename	Selects node(s) “nodename” from current context.
//Nodename/nodename2	Selects node(s) “nodename2” which are children of any “nodename” anywhere in the document.

Expression	Description
Wildcards	
*	Wildcard for elements.
@	Wildcard for attributes.
node()	Wildcard for any nodes.
Predicate examples	
//book[1]	Selects the first “book” -element from anywhere in the document.
/book[last()-1]	Selects the second to last book -element. (only direct children of the root -element)
//title	Selects all “title” -elements
//title[../year=”2005”]	Selects all “title” elements with a “year” sibling of value “2005”. (“../” = parent’s child)
//book[price<”29”]	Selects all “book” elements with a “price” of value “2005”.
//book[@ISBN="9780471791201"]	Selects “book” elements with attribute “ISBN” of “9780471791201”.
//book[price<26]	Selects all “book” elements with “price” less than 26.
//book[year>”2006”] **|** //book[price<=25]	Selects all “book” elements with “year” greater than 2006 **OR** “price” less than 25.

4 IMPLEMENTATION

The system implementation started with planning the structure of diagnostic database, how it could be used to help diagnostics and first sketches of the GUI. Microsoft Visual Studio was chosen as the main development environment. Microsoft XML Notepad 2007 was used to create and edit the diagnostic database.

Because the diagnostic procedures were available from existing and partly improved TSI documents, it was decided that the structure of those documents was not to be altered too much. That decision lead to a functional structure in which the first failed production test determined additional diagnostic measurements to be executed. These measurements where further divided into signals, which could be either real physical signals or logical conclusions of the observed behavior of the system.

The GUI (Graphical User Interface) was developed using Microsoft Visual Studio and C# -language. This combination usually leads to event driven programming style, where the primary function is reacting to events rather than running a loop or a linear flow of actions.

The schedule for the implementation stage was quite tight. Initial planning stage was roughly two weeks, followed by one week of GUI design and four weeks of intense programming. At that point the tool had reached version number 0.1 and was ready for piloting. Two pilot programs were started to verify the effectiveness of the tool.

4.1 Tool GUI

The main graphical user interface (GUI) design was started early in the development process. The goal was to develop a simple GUI which displays all relevant information at a glance and enables easy input of measured values. In case of an error or failed measurement, the tool should display additional information and action suggestions.

A total of 4 different forms were created; main view, database link tool and forms for adding new measurements and signals. These forms and their operation are described in detail in chapters 4.1.1 and 4.1.2. Additionally, some system provided forms, such as file- and message dialogs, were added to the tool. The main data containers used in the program were tree views and picture boxes. Various other system provided controls such as text boxes, numerical controls, labels, and buttons were also used.

4.1.1 Main view and measurement execution

The first action required by the user is to load a test plan and a diagnostic database. This is done via the “File”-menu. The tool remembers last loaded test plan and database files for the user. After the necessary files have been loaded, the user must enter a serial number for the DUT. The serial number is only used for logging purposes. After the text in the serial number textbox (top left in FIGURE 6) is of required length, the tests -tree view (left in FIGURE 6) is enabled.

After selecting and loading a test from the tests -tree view, a list of additional measurements is show in the measurements -list (top center in FIGURE 6). These measurements can be executed automatically; by clicking the “Run all” -button, or manually; by selecting a single measurement from the list. Once a measurement is selected, a description of the measurement is shown in the “Description” -textbox (top right in FIGURE 6). The contents of the description describes the measurement and provides some diagnostics aid. More information can be attained from a flowchart of the measurement or measurement related attachments; accessible by respective buttons near “Description” -textbox.

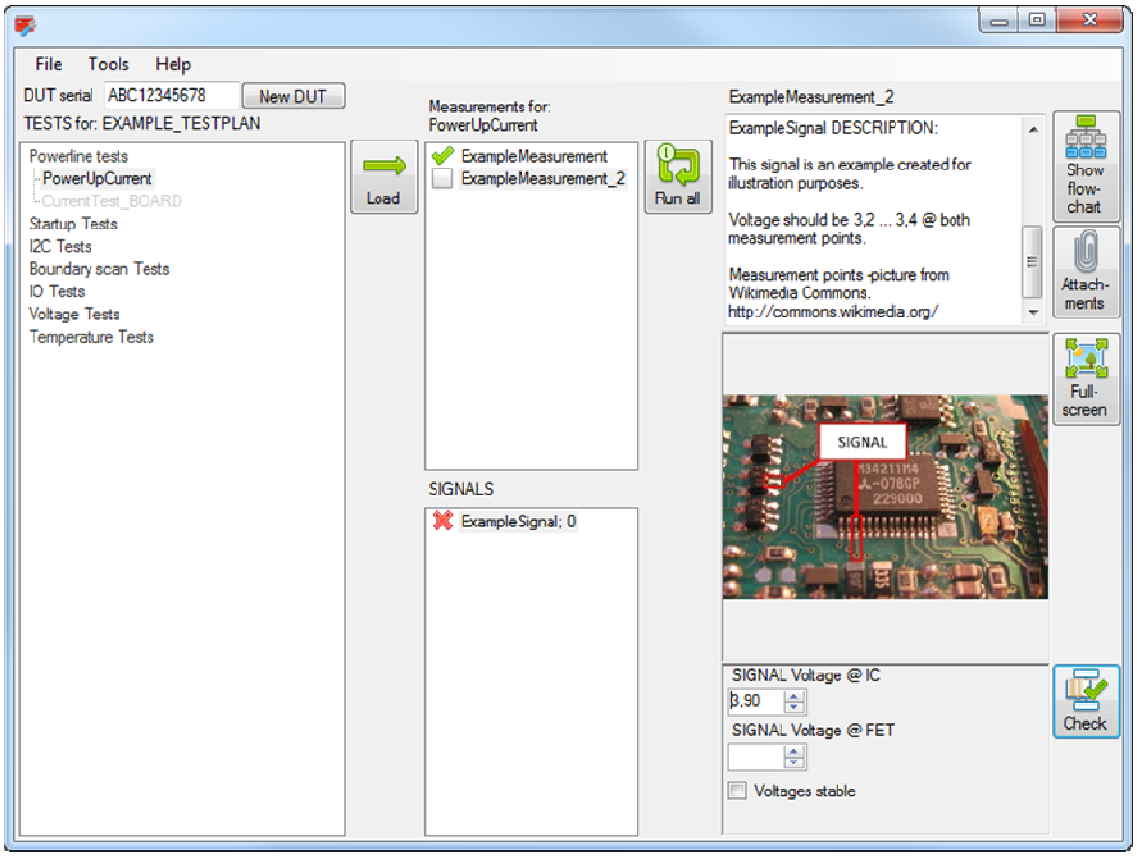

FIGURE 6. Main view of the tool

Once familiarization with the measurement has been established, the actual execution of the measurement can begin. This is done by selecting a signal from signals-list (bottom center in FIGURE 6), measuring requested signal at various points and providing input in the input-panel (bottom right in FIGURE 6). Measurement points or logical help can be found from the measurement point -picture (middle right in FIGURE 6). The picture can be enlarged by clicking the “Full screen”-button near the picture. Once the inputs have been given, they can be evaluated by clicking the “Check” -button. Signal status -picture changes accordingly and if all inputs are correct, the next signal is selected. If any of the inputs are incorrect, a message -popup is shown and the user is prompted to see additional information in flowchart and attachments; as illustrated in FIGURE 7.

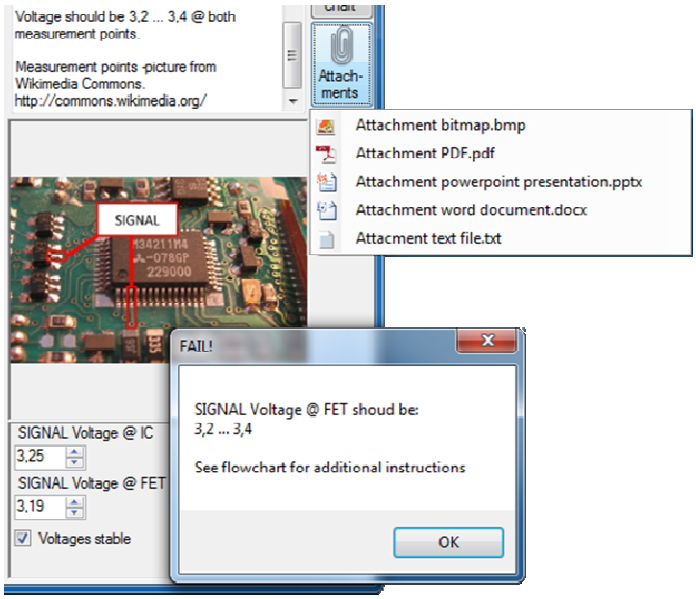

FIGURE 7. Failing input and measurement attachments

4.1.2 Database Manipulation

In order to ease the extension and manipulation of the database, a database link tool was implemented in the program. The tool can be used to create measurements and signals, linking and unlinking tests, measurements and signals to and from each others. In the “DBLinkTool”-window, the tests are shown on the left, measurements in the middle and signals on the right.

Upon selecting any item from any list, the other two lists are updated to show linked items. For example, the last selected item in FIGURE 8 is “ExampleMeasurement” and it is linked to “PowerUpCurrent” -test and “ExampleSignal; Group: 0”; illustrated by link icons. When two items from adjacent lists are selected, they can be linked by clicking the “Link” -button between the respective lists. Similarly, when the two items are already linked, the link can be broken by clicking the “Unlink” -button. It is also possible to add and delete measurements and signals. These actions can be started by clicking their respective buttons below measurement and signal -lists.

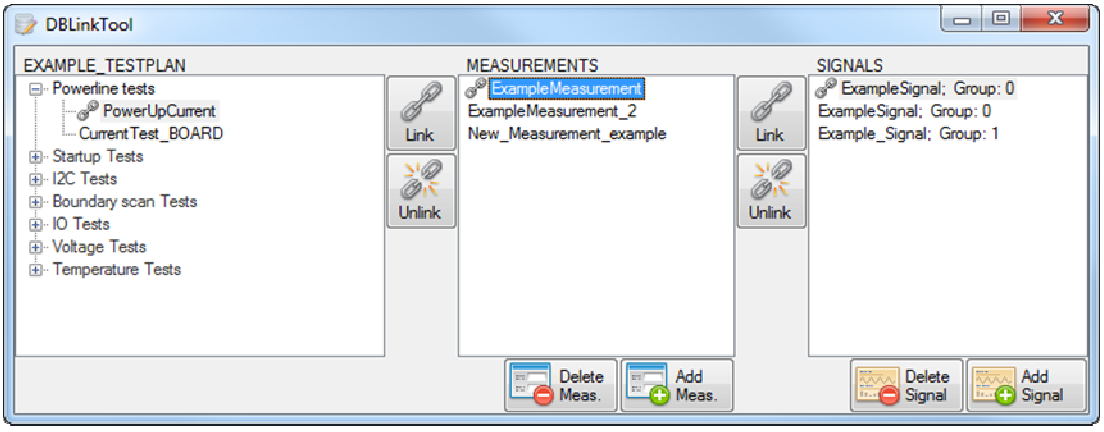

FIGURE 8. Database linking tool

New measurements are created using the “Add Measurement” -form. The function for each button and the overall operation of the form is quite self-explanatory once the name and function of the fields are known. From top to bottom in FIGURE 9 the fields are:

- Measurement name,
- Measurement description
- Object -file path and type (usually a flowchart)
- Attachment file path
- List of attached files

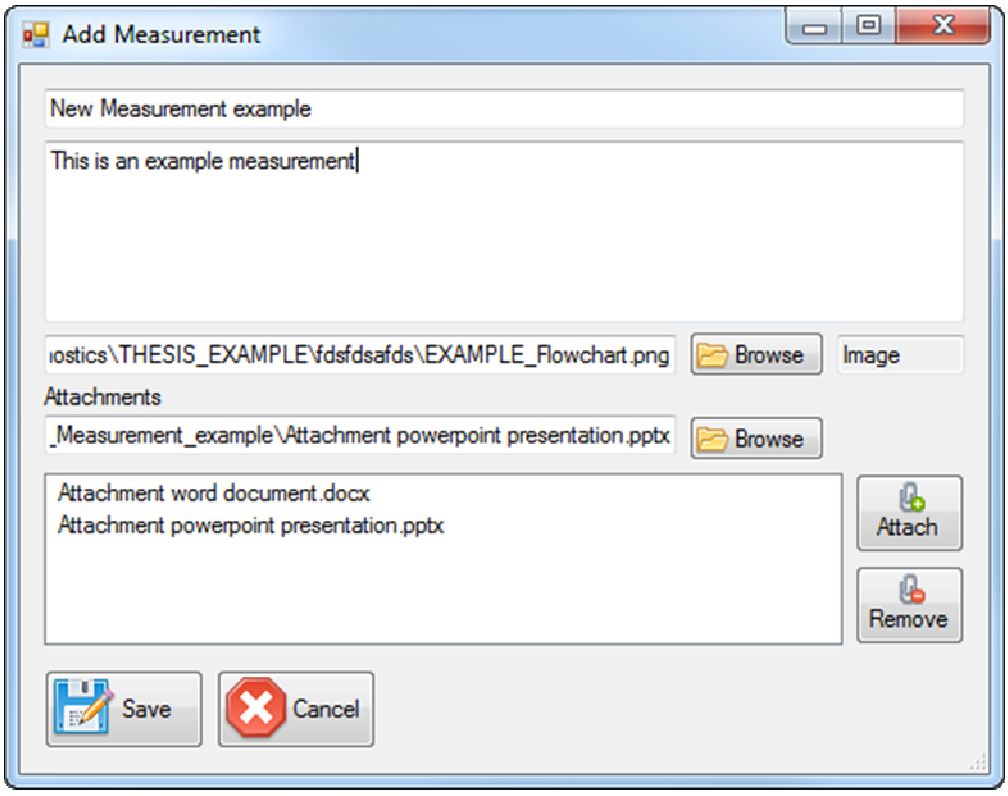

FIGURE 9. New measurement form

New signals can be added by using the “Add Signal” -form, illustrated in FIGURE 10. This form is similar to the “Add Measurements” -form, with the following differences:

- Signal group can be chosen
- Instead of attachments, inputs can be added

Each input type has special values associated to it. Each tab page also has a short description or help text to guide the creation of inputs. Once the special values have been given, the input can be added to the signal by clicking the “Add input” -button.

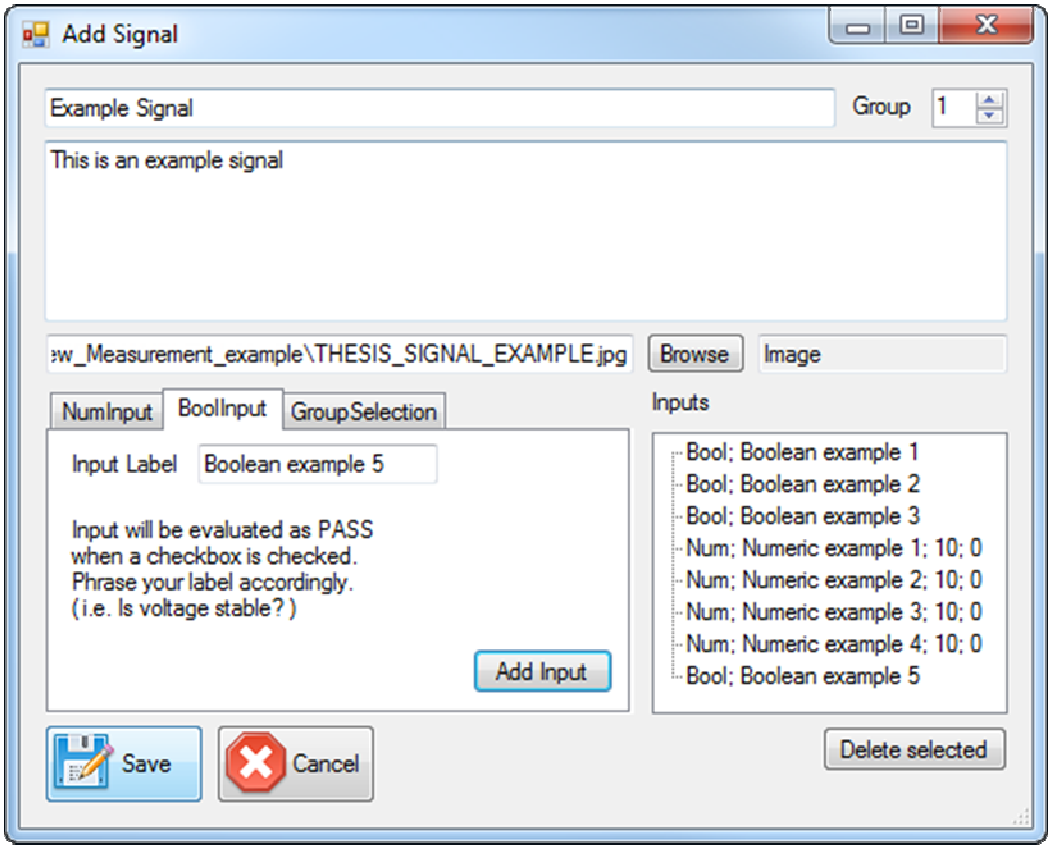

FIGURE 10. New signal form

4.1.3 DUT Log

In order to follow the usage and effectiveness of the tool, a logging feature was implemented in the tool. A log file is created for each unique serial number. It contains entries on five different levels; DUT, test, measurement, signal and input.

When a serial is entered in the DUT serial textbox, a log entry “DUT LOADED” is written. Loading a test writes an entry with test- and test plan name in the log file. Similarly loading measurements and evaluating signals create their own entries with details of the operation.

Inputs are the lowest level of information written to the log. An input entry is written for each passed input evaluation and, if one exists, the first failed evaluation. An Input entry contains: input label, input value (for numerical inputs) and evaluation result; “PASS” or “FAIL”. An example log is illustrated in FIGURE 11.

```
5.12.2014 15:43:17: DUT LOADED
5.12.2014 15:43:24:  TEST              10: PowerUpCurrent(EXAMPLE_TESTPLAN) LOADED
5.12.2014 15:43:26:     MEASUREMENT    ExampleMeasurement LOADED
5.12.2014 15:43:50:        SIGNAL      ExampleSignal; 0 being evaluated:
5.12.2014 15:43:50:           INPUT    SIGNAL Voltage @ IC  3,3   PASS
5.12.2014 15:43:50:           INPUT    SIGNAL Voltage @ FET 3,3   PASS
5.12.2014 15:43:50:           INPUT    Voltages stable   PASS
5.12.2014 15:43:50:     MEASUREMENT    ExampleMeasurement   PASS
5.12.2014 15:43:50:     MEASUREMENT    ExampleMeasurement_2 LOADED
5.12.2014 15:43:57:        SIGNAL      ExampleSignal; 0 being evaluated:
5.12.2014 15:43:57:           INPUT    SIGNAL Voltage @ IC  3,3   PASS
5.12.2014 15:43:57:           INPUT    SIGNAL Voltage @ FET 3  FAIL
5.12.2014 15:44:03:        SIGNAL      ExampleSignal; 0 being evaluated:
5.12.2014 15:44:03:           INPUT    SIGNAL Voltage @ IC  3,3   PASS
5.12.2014 15:44:03:           INPUT    SIGNAL Voltage @ FET 3,30  PASS
5.12.2014 15:44:03:           INPUT    Voltages stable   PASS
5.12.2014 15:44:03:     MEASUREMENT    ExampleMeasurement_2 PASS
...
```

FIGURE 11. Log example

4.2 Diagnostic database

In order for the tool to work on different products and product lines, all diagnostic information was decided to be stored in an external file; a diagnostic database. This database would serve as the diagnostic model of the product or products. Main elements and relations in the database were to be different diagnostic measurements, signals within those measurements, their relation to production tests and any additional diagnostic information for aforementioned elements.

The database was chosen to be implemented as an xml file. This decision was made mainly for two reasons. Firstly, the database could in the later phases of development be rather easily extended in contrast to a SQL based database which would have needed to be recompiled each time even a minor change was made. And secondly, the used development environment, Visual Studio with C#, provided good built-in libraries for easy and straightforward XML manipulation and querying. In addition, although an xml file is quite inefficient in storing data, the file would be stored locally on each PC, and the size of the additional diagnostic information, namely measurement point pictures and flowcharts, would greatly exceed the size of the database file. FIGURE 12 illustrates the high level structure of the database and relation to a test plan.

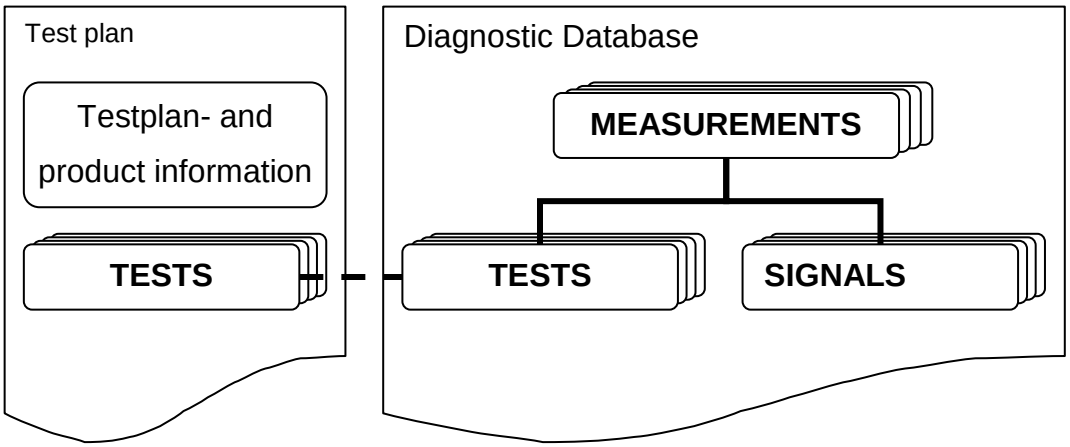

FIGURE 12. Diagnostic database high level structure

The following diagrams (FIGURE 13, FIGURE 14, FIGURE 15 and FIGURE 16) illustrate the structure of the diagnostic database in more detail. An element with **bolded** text represents a set (≥1) of elements.

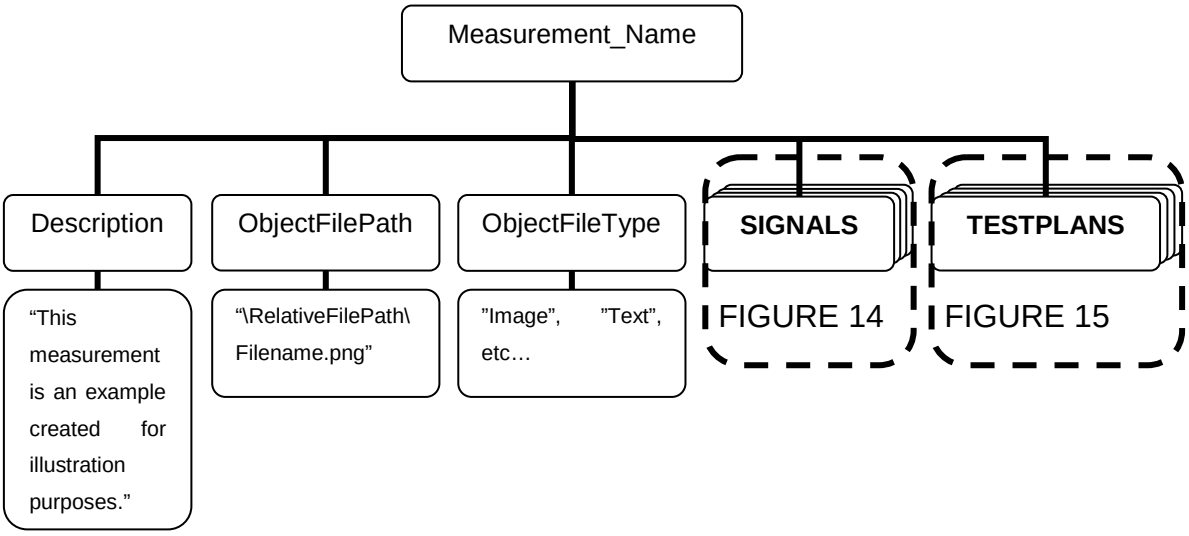

FIGURE 13. Measurement element and direct children

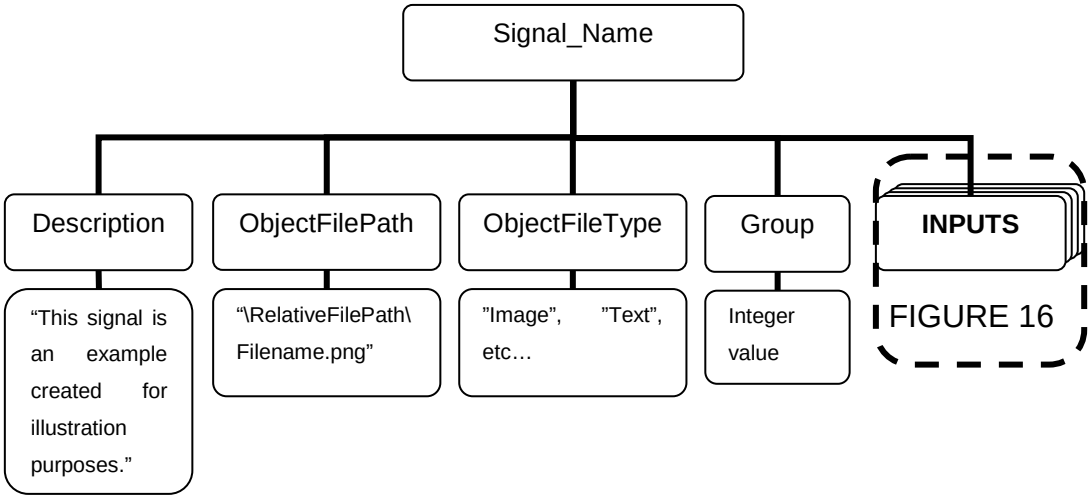

FIGURE 14. Signals-element and direct children

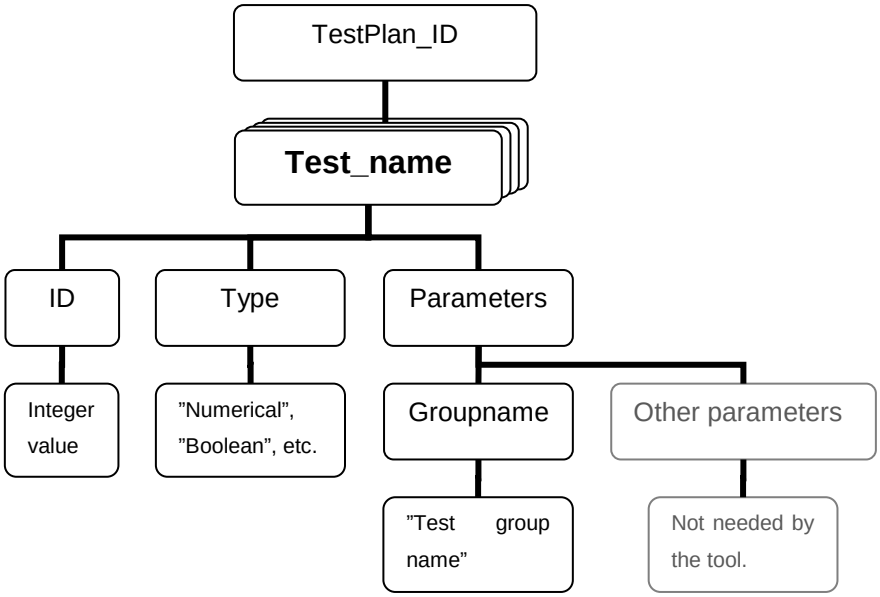

FIGURE 15. TestPlans element and children

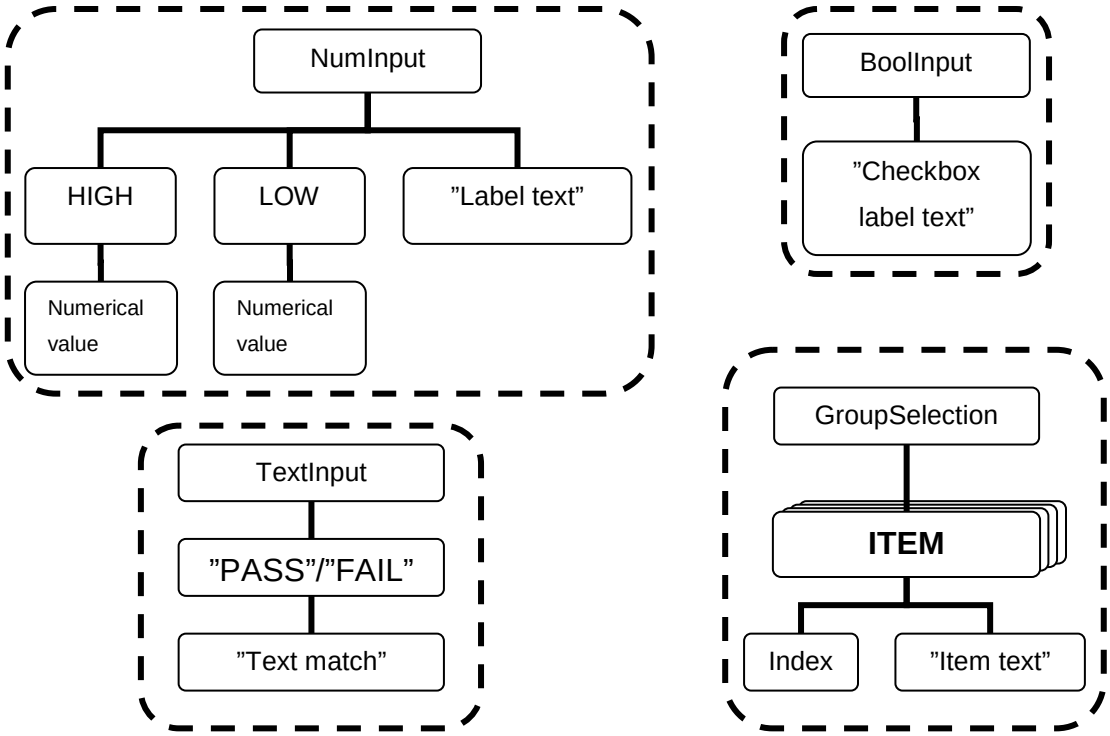

FIGURE 16. Input elements

4.3 Piloting

Two pilot programs were scheduled once the tool development reached a state where; no major bugs were interfering with normal operation, most of the messages and error handling was implemented and some diagnostic information was added to the database. First pilot was launched immediately in the Oulu factory repair area. The second pilot was launched after two weeks in the repair area of a high volume factory.

The pilot in Oulu started after the tool was installed on one repair area tester. The tool was advised to be used whenever failures within a predetermined scope were found. Log files would be gathered regularly to monitor the effectiveness of the tool. The main aims for this pilot were to find bugs and give the repair operators in Oulu a chance to try out the program and give improvement suggestions. However, because of the small amount of boards matching the scope and the nature of the faults on those boards, the usage of the tool was minimal and no bug reports or improvement were received.

The second pilot was launched in the high volume -factory after approximately two weeks of piloting in Oulu. Few minor improvements and bug fixes were done to the tool and the diagnostic database was updated. In the high volume factory, the program was installed on a production tester, where repair operators perform diagnostic measurements. Similarly, the log files were gathered weekly by local tester engineers and sent back to Oulu for analysis. See chapter 5 for detailed results.

5 CONCLUSIONS AND RESULTS

An automated diagnostic tool was developed in order to improve diagnostic accuracy in production rework process. The tool consists of a GUI and diagnostic databases for different products. In order to support further development of the tool, a software requirements specification document was drafted. The diagnostic databases contain system level fault models and lower level rule based diagnostic information. The tool also gathers logs of diagnosed products, which are used to help evaluate the performance of the tool.

The tool was developed using Microsoft Visual Studio and C# programming language. The databases were implemented using XML language, and the tool utilizes XML built-in addressing language XPath to query the diagnostic database. The production tests and their limits are taken directly from a production test plan -file and the databases link to those tests.

Developing a diagnostic system and implementing an XML -based database deepened the author's knowledge about diagnostic systems and XML -related technologies. The initial experiences from the tool lead to a decision to take it into global use if the pilot programs yield positive results.

As of mid December 2014, the system has been piloted for a total of four weeks in two Nokia Networks -factories. The initial pilot in Oulu -factory did not yield any reasonable results because the number of faulty products matching the scope was minimal and hence the tool has not been used very much. The piloting in the high volume factory yielded encouraging results. During the piloting period the number of misdiagnoses for the entire product decreased by 88%, and no misdiagnoses were logged within the scope of the tool.

The results of the pilot programs are quite positive and repair operators have reported the tool to be easier to use compared to TSI-documents. The tool is planned to be taken into global use for Nokia Networks system module production rework. The roll-out will be in early 2015 and until then the tool will be further developed and diagnostic databases will be extended and refined to cover more faults with better accuracy.

SOURCES

1. Nokia Corporation. 2014. Financial report Q3 2014, Earnings release. Date of retrieval: 10 October 2014, http://company.nokia.com/en/investors/financial-reports/results-reports

2. IEEE 830-1998, IEEE Computer Society, Software Engineering Standards Committee & IEEE-SA Standards Board. 1998. IEEE Recommended Practice for Software Requirements Specifications. Institute of Electrical and Electronics Engineers.

3. Davis, R., & Hamscher, W. C., 1988. Model-based reasoning: Troubleshooting (No. AI-M-1059). MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB.

4. Fenton, W. G., McGinnity, T. M., & Maguire, L. P. 2001. Fault diagnosis of electronic systems using intelligent techniques: a review. Systems, Man, and Cybernetics, Part C: Applications and Reviews. IEEE Transactions on, 31(3). 269-281.

5. W3C. 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation 26 November 2008. Date of retrieval: 1 December 2014. http://www.w3.org/TR/xml/

6. W3Schools. 2014. XML DOM Tutorial. Date of retrieval: 1 December 2014. http://www.w3schools.com/dom/

7. W3Schools. 2014. XML Tutorial. Date of retrieval: 1 December 2014. http://www.w3schools.com/xml/

8. W3C. 1999. XML Path Language (XPath), W3C recommendation 16 November 1999. Date of retrieval: 1.12.2014. http://www.w3.org/TR/xpath

APPENDICES

Appendix 1 Memo of initial data (in Finnish)

Appendix 2 Thesis example diagnostic database

OAMK LÄHTÖTIETOMUISTIO
Tekniikan yksikkö
Tietotekniikan osasto
Langattomien laitteiden suuntautumisvaihtoehto

Tekijä Arto Drees

Tilaaja Nokia

Tilaajan yhteyshenkilö(t) ja yhteystiedot

Nokia Oyj, Kaapelitie 4 90620 Oulu, Ville Nurkkala, tel.+358504869550

Projektin nimi High pin count component diagnostic improvement

Projektin tavoitteet

Projektin tavoitteena on tukiasematuotannossa tapahtuvan korjaus- ja diagnosointikyvykkyyden parantaminen. Parannetaan systeemimodulin kahdelle piirikortille tuotannon korjausohjetta suurten komponenttien osalta. Korjausohjetta käytetään vaatimusmäärittelynä automatisoidulle korjausohjelmistolle, jolla pyritään ohjaamaan ja helpottamaan korjaustoimintaa. Tavoitteena vianhakuohjeella ja ohjelmistolla on etsiä ja esittää selkeästi tarvittavat vianhakutoimet sekä varmistaa niiden oikeaoppinen suoritus, jotta juurisyy löytyy nopeasti ja tuotteet saadaan korjattua luotettavasti.

Projektissa käytettävä prosessimalli (mahdollinen vaihejakoon perustuva aikataulutus)

A3 project process: Background, Current condition, Goal statement, Root-Cause analysis, Countermeasures, Actual Results, Follow-up

Projektissa käytettävät menetelmät ja teknologiat

Boundary Scan, BIST, POST, C, Labwindows, Labview, Visual Studio, C#

Projektin alustava aikataulu

Projektityö 5 viikkoa, opinnäytetyö 2014 loppuun

Päivväys ja allekirjoitukset

```xml
<?xml version="1.0"?>
<MEASUREMENTS>
  <ExampleMeasurement>
    <Description>This measurement is an example created for illustration purposes.

        Flowchart shows troubleshooting to fix a broken lamp.

        Flowchart -picture from Wikimedia Commons.
        http://commons.wikimedia.org/ </Description>
    <ObjectFilePath>\THESIS_EXAMPLE\fdsfdsafds\EXAMPLE_Flowchart.png</ObjectFilePath>
    <ObjectFileType>Image</ObjectFileType>
    <SIGNALS>
      <ExampleSignal>
        <Description>This signal is an example created for illustration purposes.

            Voltage should be 3,2 ... 3,4 @ both measurement points.

            Measurement points -picture from Wikimedia Commons.
            http://commons.wikimedia.org/ </Description>
        <ObjectFilePath>\THESIS_EXAMPLE\ExampleMeasurement\THESIS_SIGNAL_EXAMPLE.jpg</ObjectFilePath>
        <ObjectFileType>Image</ObjectFileType>
        <NumInput>SIGNAL Voltage @ IC<HIGH>3,4</HIGH><LOW>3,2</LOW></NumInput>
        <NumInput>SIGNAL Voltage @ FET<HIGH>3,4</HIGH><LOW>3,2</LOW></NumInput>
        <BoolInput>Voltages stable</BoolInput>
        <Group>0</Group>
      </ExampleSignal>
    </SIGNALS>
    <TESTPLANTESTS>
      <EXAMPLE_TESTPLAN>
        <PowerUpCurrent>
          <ID>10</ID>
          <TYPE>NumLimitTest</TYPE>
          <PARAMETERS>
            <GROUPNAME>"#,x.x.x, Powerline tests"</GROUPNAME>
          </PARAMETERS>
        </PowerUpCurrent>
      </EXAMPLE_TESTPLAN>
    </TESTPLANTESTS>
  </ExampleMeasurement>
  <ExampleMeasurement_2>
    <Description>This measuremet is an example created for illustration purposes.

        Flowchart shows troubleshooting to fix a broken lamp.

        Flowchart -picture from Wikimedia Commons.
        http://commons.wikimedia.org/ </Description>
    <ObjectFilePath>\THESIS_EXAMPLE\ExampleMeasurement_2\EXAMPLE_Flowchart.png</ObjectFilePath>
    <ObjectFileType>Image</ObjectFileType>
    <AttachmentFile>\THESIS_EXAMPLE\ExampleMeasurement_2\Attachment bitmap.bmp</AttachmentFile>
    <AttachmentFile>\THESIS_EXAMPLE\ExampleMeasurement_2\Attachment PDF.pdf</AttachmentFile>
    <AttachmentFile>\THESIS_EXAMPLE\ExampleMeasurement_2\Attachment powerpoint presentation.pptx</AttachmentFile>
    <AttachmentFile>\THESIS_EXAMPLE\ExampleMeasurement_2\Attachment word document.docx</AttachmentFile>
    <AttachmentFile>\THESIS_EXAMPLE\ExampleMeasurement_2\Attacment text file.txt</AttachmentFile>
```

```
    <SIGNALS>
      <ExampleSignal>
        <Description>This signal is an example created for illustration purposes.

                     Voltage should be 3,2 ... 3,4 @ both measurement points.

                     Measurement points -picture from Wikimedia Commons.
                     http://commons.wikimedia.org/ </Description>
        <ObjectFilePath>\THESIS_EXAMPLE\ExampleMeasurement_2\THESIS_SIGNAL_EXAMPLE.jpg</ObjectFilePath>
        <ObjectFileType>Image</ObjectFileType>
        <NumInput>SIGNAL Voltage @ IC<HIGH>3,4</HIGH><LOW>3,2</LOW></NumInput>
        <NumInput>SIGNAL Voltage @ FET<HIGH>3,4</HIGH><LOW>3,2</LOW></NumInput>
        <BoolInput>Voltages stable</BoolInput>
        <Group>0</Group>
      </ExampleSignal>
    </SIGNALS>
    <TESTPLANTESTS>
      <EXAMPLE_TESTPLAN>
        <PowerUpCurrent>
          <ID>10</ID>
          <TYPE>NumLimitTest</TYPE>
          <PARAMETERS>
            <GROUPNAME>"#,x.x.x, Powerline tests"</GROUPNAME>
          </PARAMETERS>
        </PowerUpCurrent>
      </EXAMPLE_TESTPLAN>
    </TESTPLANTESTS>
  </ExampleMeasurement_2>
  <New_Measurement_example>
    <Description>This is an example measurement</Description>
    <ObjectFilePath>\THESIS_EXAMPLE\New_Measurement_example\EXAMPLE_Flowchart.png</ObjectFilePath>
    <ObjectFileType>Image</ObjectFileType>
    <AttachmentFile>\THESIS_EXAMPLE\New_Measurement_example\Attachment word document.docx</AttachmentFile>
    <AttachmentFile>\THESIS_EXAMPLE\New_Measurement_example\Attachment powerpoint presentation.pptx</AttachmentFile>
    <SIGNALS/>
    <TESTPLANTESTS>
      <EXAMPLE_TESTPLAN>
        <CurrentTest_BOARD>
          <ID>350</ID>
          <TYPE>NumLimitTest</TYPE>
          <TOLERANCE_LIMITS>0</TOLERANCE_LIMITS>
          <LIMIT1>
            <LOW>0.3</LOW>
            <HIGH>3.0</HIGH>
            <UNITS>A</UNITS>
          </LIMIT1>
          <PARAMETERS>
            <GROUPNAME>"#,x.x.x, Powerline tests"</GROUPNAME>
          </PARAMETERS>
        </CurrentTest_BOARD>
      </EXAMPLE_TESTPLAN>
    </TESTPLANTESTS>
  </New_Measurement_example>
</MEASUREMENTS>
```