Johannes Lüftenegger, BSc

Development and Evaluation of a User Interface Concept for an Industrial Wind Turbine Diagnosis Application

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa
Dipl.-Ing. Dipl.-Ing. Roxane Koitz-Hristov, BSc

Institute for Softwaretechnology

Graz, November 2017

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

Since industrial systems are becoming more and more complex, diagnosis applications are required in order to identify the system's health status as well as identifying faulty components. Model-based and knowledge-based diagnosis are methods to identify parts that cause abnormal behavior of a system. In contrast to knowledge-based diagnosis systems, the advantage of the model-based method lies in the updatability of the knowledge base.
Decades ago, the theoretical background of model-based diagnosis systems was researched, and various prototypes have been implemented to show performance enhancements. Yet, industrial applications are sparse. Whereas knowledge-based systems are widely used in different domains, only few model-based applications have been published. The existing prototypes mainly demonstrate the implemented algorithms, leaving users outside the research field behind. Further, there is no research about design and layout requirements in the context of model-based diagnosis applications.
This work introduces best practices for the design of a model-based diagnosis application with focus on user acceptance factors, i.e. usability and usefulness. Furthermore, the processes for designing and evaluating such systems is described, taking the needs and requirements of users with different levels of experience into account. A general interface was developed to suit a large variety of diagnosis purposes and a usability test was conducted to identify design issues for which possible solutions were stated.
In order to close research gap regarding user interface design of industrial model-based diagnosis systems, the requirements for such a system in the context of wind turbine maintenance were described. Moreover, a clickable prototype was created to support the fault identification process of wind turbine maintenance personnel. The interface was developed in an iterative design process, where the needs of several stakeholders needed to be considered. In order to evaluate the usability of the interface, the service technicians responsible for the wind turbine maintenance participated in a usability test.

The identified issues were analyzed and sparked ideas of alternative design solutions to further improve the usability of the interfaces. The proposed interface designs can be used to bring model-based diagnosis systems into industrial applications. It has been shown that users are able to interact with such systems using this interface, even when they have no technical background.

Kurzfassung

Da industrielle Systeme in den letzten Jahren deutlich an Komplexität gewonnen haben, ist es notwendig, Diagnoseanwendungen zur Überwachung und Identifikation fehlerhafter Komponenten einzusetzen. Mithilfe von modell- oder wissensbasierten Diagnoseanwendungen können jene Komponenten identifiziert werden, die ein fehlerhaftes Verhalten eines Systems verursachen. Dass die Wissensbasis bei modellbasierten Diagnosesystemen gewartet und wiederverwendet werden kann, stellt einen Vorteil gegenüber wissensbasierten Diagnosesystemen dar.
Obwohl der theoretische Hintergrund von modellbasierten Systemen in den letzten Jahrzehnten erforscht wurde, konnte sich die modellbasierte Diagnose in industriell verwendeten Anwendungen nicht durchsetzen. Im Gegensatz zu wissensbasierten Systemen, die bereits in vielen Bereichen Anwendung gefunden haben, wurden bisher nur wenige modellbasierte Anwendungen entwickelt. Es handelt sich hierbei hauptsächlich um Forschungsprototypen, um verbesserte Algorithmen zu demonstrieren. Außerhalb des wissenschaftlichen Umfelds finden diese Programme jedoch keine Anwendung. Die Anforderungen an das Design und Layout modellbasierter Diagnosesysteme sind bislang unerforscht.
Die vorliegende Arbeit zeigt die Anforderungen an eine modellbasierte Diagnoseanwendung im Bereich der Benutzerakzeptanz hinsichtlich der Faktoren "Benutzbarkeit" und "Brauchbarkeit". Um die Forschungslücke im Zusammenhang mit modellbasierten Diagnosesystemen im Bereich des Interface-Designs zu schließen, wird in der vorliegenden Arbeit zunächst ein generelles Design einer modellbasierte Diagnoseanwendung vorgestellt, das in vielen unterschiedlichen Anwendungsgebieten eingesetzt werden kann. Zur Identifikation allfälliger Designfehler wurde das Programm mittels eines Usability-Tests evaluiert. Für die dabei festgestellten Probleme wurden Lösungen erarbeitet. In der Folge wurden die konkreten Anforderungen einer Diagnoseanwendung für die Wartung von Windturbinen ermittelt. Es wurde ein Interface-Prototyp

mit dem Ziel entwickelt, Servicetechnikerinnen und Servicetechniker bei der Fehlerdiagnose zu unterstützen. Die Benutzeroberfläche wurde in einem iterativen Entwicklungsprozess stetig verfeinert, wobei die Anforderungen verschiedener Stakeholder berücksichtigt wurden. Um die Tauglichkeit des Prototyps zu ermitteln, wurde ein Usability-Test durchgeführt, an dem jene Techniker teilnahmen, die für die Wartung von Windturbinen zuständig sind. Für die dabei identifizierten Fehler des Systems wurden Lösungsansätze erarbeitet. Die entwickelte Benutzeroberfläche bildet die Grundlage für weitere industriell verwendbare modellbasierte Diagnoseanwendungen. Da die Benutzeroberfläche intuitiv bedienbar ist, kann sie auch von Anwenderinnen und Anwendern mit geringem oder durchschnittlichem Grad an Erfahrung bedient werden.

Contents

List of Figures

List of Abbreviations

AB	Abnormal Behavior
ATMS	Assumption-based Truth Maintenance System
ARR	Analytical Redundancy Relation
COMPS	Set of Components
CU	Casual User
CBD	Consistency-based Diagnosis
DP	Diagnostic Problem
DxPCs	Diagnosis with Possible Conflicts
EU	Expert User
FDI	Fault Detection and Isolation
GUI	Graphical User Interface
KB	Knowledge Base
MBD	Model-based Diagnosis
NAB	Not Abnormal Behavior
NU	Novice User
OBS	Set of Observations
PC	Possible Conflict
PHCAP	Propositional Horn Clause Abduction Problem
SUS	System Usability Scale
SM	System Model
SME	Subject Matter Expert
SD	System Description
TAM	Technology Acceptance Model
TTF	Task-technology Fit Model
UI	User Interface

1. Introduction

Since the early 1960s, artificial intelligence (AI) has been a popular topic in computer science. Expert systems are the most commercially successful approaches to AI [19], with the purpose of replacing human experts. Shortliffe et al. [59] develop the expert system MYCIN, which gives doctors recommendations for antimicrobial therapies. Although MYCIN was able to provide acceptable recommendations in 75% of the cases, it could not overcome the lack of trust in computer systems in 1975. Nevertheless, it was an important milestone in the research of knowledge-based systems.
The knowledge base of expert systems typically consists of *IF-THEN* clauses, where the *IF*-statement consists of symptoms or conditions and the *THEN*-part provides the consequence. In a large knowledge base, some rules may depend on other rules. Therefore, inserting new rules might cause unexpected effects [19] which causes high costs to maintain and extend such systems. Once the limitations of expert systems had been discovered, several other methods (e.g. model-based systems, neural networks) followed to assist humans in fault-identification processes and decision-making tasks. In various papers, prototypes including a graphical user interface (GUI) have been published to demonstrate the functionality or improvements of the underlying algorithms. Few publications showed the importance of usability and usability testing of diagnosis or fault identification systems for industrial applications. This is particularly troublesome since user acceptance is an important aspect in order to introduce a new technology. Davis [9] proves that user acceptance is strongly related to usefulness of a product. Freiberg, Striffler and Puppe [14] suggest to introduce an agile software development process to keep interface and interaction design in a central spot during the engineering process. Nurminen, Karonen and Hätönen [45] state the importance of usability and the involvement of experts during an interactive development process.

In order to bring MBD into industrial applications, this thesis has been part of a project in cooperation with Uptime Engineering GmbH. This company has many years of experience in wind turbine maintenance and is currently developing an application for fault identification of wind turbines. Since no research of usability and interface design in the field of model-based diagnosis systems exists, the aim was to identify the requirements for such systems. Additionally, the integration of the diagnosis system in their existing monitoring software was another prerequisite to be considered. Since the company runs strict style guidelines, the interface of the diagnosis system needs to follow their design conventions. The design of the interface should be extensible to be suitable for diagnosis scenarios outside the wind turbine environment.

This work gives an overview of existing diagnosis applications with focus on the usability of the user interface. Several research tools for model-based diagnosis are presented, analyzed and potential usability issues identified. The evaluation of the presented applications' user interfaces is based on operating system manufacturers' guidelines and recommendations of usability experts. Based on the findings, best practices were identified, which build the foundation for interface design to bring model-based diagnosis to industrial applications.

The main contributions of this master's thesis are:

- Identification of best practices based on existing diagnosis applications
- Development of the workflow and requirements for an general interface for model based diagnosis
- Design and evaluation of a general interface for model-based diagnosis applications
- Identification of the requirements of an interface for diagnosis of wind turbines
- Design and evaluation of a clickable prototype for wind turbine maintenance

1. Introduction

Two interface prototypes for abductive model-based diagnosis systems are presented. First, a general interface was developed which can be used for different applications and domains. Currently, smart home devices are very popular in consumer electronics. Therefore, the diagnosis interface was integrated into an application where users were able to interact which such devices and perform diagnosis activities for different fault scenarios. In order to evaluate the usability of the interface, several users with different levels of experience in computer systems participated in a usability test. The second interface was designed to support the maintenance personnel of an energy provider with their repair and replacement tasks of wind turbines. The needs of several stakeholders had to be taken into account to identify the requirements for such a system. After several design iterations, a clickable prototype was developed and evaluated.

The preliminaries of model-based diagnosis are summarized in Chapter 2. In this chapter, various approaches of model-based diagnosis are discussed. Since there is no literature available for usability in the area of model-based diagnosis, Chapter 3 presents usability studies of knowledge-based systems as well as research tools for model-based diagnosis. The fundamentals of usability testing and inspection methods are presented in Chapter 4. Chapter 5 introduces the requirements of a general interface for model-based diagnosis. The scenario used is presented as well as the screen design of the resulting application. Furthermore, the setup of the performed usability test and the results of this study are presented and discussed. Chapter 6 describes the iterative design process and the requirements of the wind turbine diagnosis system. Based on the identified workflow, the design of the clickable prototype is presented as well as the performed evaluation method and the observed usability issues. Chapter 7 concludes the thesis and points out some opportunities for future work.

2. Preliminaries

This chapter presents the logical principles required for model-based diagnosis and provides a comparison of the various approaches.
Two research communities exist in the field of model-based diagnosis: Control engineering and artificial intelligence, where in this work the former is defined as FDI (Fault Detection and Isolation) and the latter as MBD (Model-based Diagnosis) approach. The FDI method is based on analytical models and linear algebra where MBD deals with logical symbolic and qualitative models. Cordier et al.[6] provided a comparison between these two different approaches, which are presented in the following sections.

2.1. Fault Detection and Isolation Approach

The system model of the FDI method contains a set of constraints which describe the behavior of the components of the system and a set of sensors providing the observation. Definition 1 by Cordier et al. [6] formalizes the system model.

Definition 1. System model [6]: *The system model SM is defined as the behavioral model BM, i.e. the set of relations defining the system behavior, together with the observation model OM, i.e. the set of relations defining the observations that are performed on the system and the sensor models. The set V of variables can be decomposed into the set of unknown variables X and the set of observed variables O.*

In order to check the consistency of the observations against the system model, the analytical redundancy relation (ARR) is used. The ARR is a constraint which is satisfied if the observed behavior of a system satisfies the constraints of the system model, i.e. the observed sensor values are appropriate. Definition 2 illustrates the formalization of ARR [6].

Definition 2. Analytical redundancy relation [6]: *An analytical redundancy relation (ARR) is a constraint deduced from the system model which contains only observed variables, and which can therefore be evaluated from any OBS. It is noted r = 0, where r is called the residual of the ARR.*

If presence of a fault F_j does not affect ARR_i, then the fault signature s_{ij} is zero. If some s_{ij} is non-zero, the fault F_j is expected to influence ARR_i. The fault signature for a specific F and ARR is stored in a matrix, i.e. the signature matrix. Definitions 3 and 4 by Cordier et al.[6] formalize fault signature and signature matrix.

Definition 3. Fault signature [6]: *Given a set ARR of ARR_i: $r_i = 0$, with Card(ARR) = n, the (theoretical) signature of a fault F_j is given by the binary vector $FS_j = [s1_j, s2_j, ..., sn_j]^T$ in which s_{ij} is given by the following application:*

$$
\begin{array}{lcl}
s: & & \\
ARR \times F & \rightarrow & \{0,1\} \\
(ARR_i, F_j) & \rightarrow & s_{ij} = 1 \text{ if the component affected by } F_j \text{ is involved in } ARR_i \\
 & & s_{ij} = 0 \text{ otherwise}
\end{array}
$$

Definition 4. Signature matrix [6]: *Given a set ARR of n ARRs, the signatures of a set of faults $F = \{F_1, F_2, , F_m\}$ all put together constitute the so-called signature matrix FS of dimensions $n \times m$.*

Further, a diagnosis result can be obtained by comparing the theoretical fault signature with the observed fault signature. Noise and disturbance models are used in order to consider inaccuracies. The actual comparison is stated as a decision-making problem which is solved by defining a consistency criterion.

Definition 5. Consistency criterion [6]: *An observed signature $OS = [OS_1, ... ,OS_n]^T$ is consistent with a fault signature $FS_j = [s1_j, ... ,sn_j]^T$ if and only if $OS_i = s_{ij}$ for all i.*

Since this definition is not practicable in most situations, a weaker similarity-based consistency criterion is used to get feasible results.

2.2. Approaches to Model-Based Diagnosis

In contrast to the previously presented control engineering approach, this section summarizes the AI approach of MBD. While the FDI variant uses quantitative models, the MBD approach uses qualitative / symbolic descriptions of the system to diagnose. The following subsections present the consistency-based and the abductive approach of MBD.

2.2.1. Consistency-Based Diagnosis

The MBD approach was introduced by de Kleer & Williams [12] and Reiter [52]. In their definitions, a system (e.g. electrical circuit, car, power plant, ...) is represented by a system description (SD) which defines the normal behavior. The purpose of a diagnosis system is to determine a faulty component that causes the abnormal behavior of the system. Once the observed behavior of a system differs from the predicted behavior defined by the system model, a diagnostic problem can be formed, meaning the observed behavior is inconsistent under the assumption that all components of the system show normal behavior.

As mentioned before, the goal of MBD is to detect discrepancies between the observed behavior and the behavior of the logical model. A diagnostic problem is present once the observations are inconsistent with the assumption that the system's components are working properly. Definition 6 illustrates Picardi's [48] formalization of a diagnostic problem.

Definition 6. Diagnostic problem [48]: *A diagnostic problem is a triple* $\langle SD, COMPS, OBS \rangle$ *where*

- *SD is a system description*
- *COMPS is a set of component names mentioning the components that can be faulty*
- *OBS is a set of ground atomic formulas expressing the observations made on the system, such that the set of formulas* $SD \cup ok(c) \mid c \in COMPS \cup OBS$ *is inconsistent.*

In order to find a solution for a diagnostic problem *DP*, the assumption ok(c) from a component c is replaced by its negation ¬ok(c). If the assumption that the component c is faulty causes consistency, $\Delta = \{c\}$ is a diagnosis for the diagnostic problem. Definition 7 shows the formalization of the solution of a diagnostic problem, i.e. diagnosis [48].

Definition 7. Diagnosis [48]: *Let DP = $\langle$ SD, COMPS, OBS $\rangle$be a diagnostic problem. We say that a set $\Delta \subseteq$ COMPS is a consistency-based diagnosis for DP if it is a minimal set such that SD $\cup$ {ok(c) | c $\in$ COMPS Δ } $\cup$ {¬ok(c) | c $\in \Delta$ } $\cup$ OBS is consistent.*

A conflict set is a set of assumptions about the components' health status. At least one component inside the conflict set is not working as expected. The definition of a conflict set is stated below [48].

Definition 8. Conflict set [48]: *A set of components $\{c_1, \ldots, c_k\} \subseteq$ COMPS is a conflict set for a diagnostic problem DP if SD $\cup$ OBS $\cup$ ok(c_1), ... , ok(c_k) is inconsistent. A conflict set is minimal if there is no other conflict set properly contained in it.*

Each diagnosis result should have at least one component in common, therefore combining elements from different conflict sets form a hitting set. Definition 9 states Picardi's [48] formalization of a hitting set.

Definition 9. Hitting set[48]: *Given a collection C of sets, a hitting-set for C is a set $H \subseteq \bigcup_{s \in c} S$ such that $H \cap S \neq \emptyset$ for each $S \in C$. A hitting set is minimal if no proper subset of it is a hitting set for C.*

Example

Consider the simple boolean circuit [67] of two and-gates and two inverters depicted in Figure 2.1. The output of an and-gate (A1, A2) is only true, if both inputs are true. The output of an inverter (I1, I2) is the negated input value.

Providing the input values a = true, b = true, and c = true, the circuit computes f = false and g = true, which is obviously an incorrect behavior. In order to identify the faulty component using the consistency-based diagnosis approach,

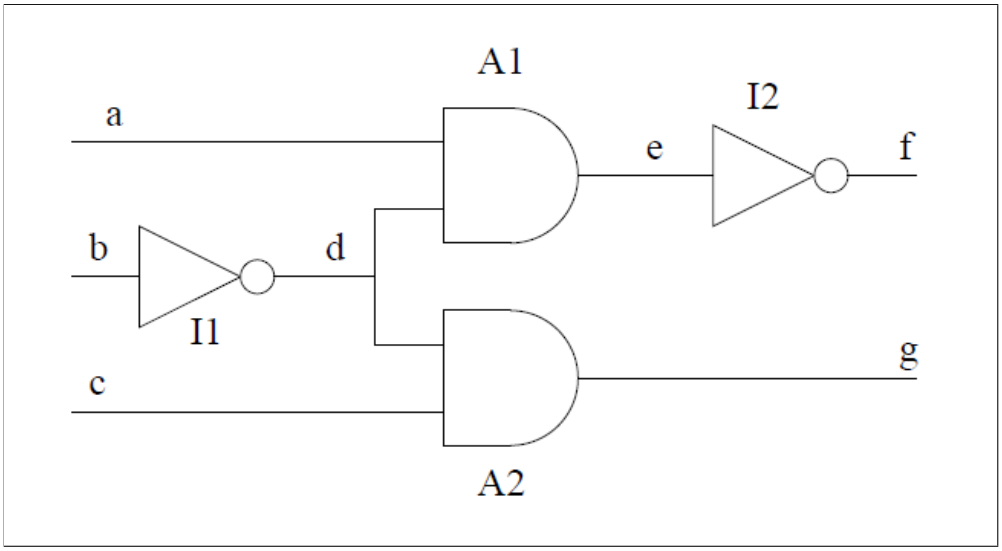


Figure 2.1.: Boolean circuit [67]

the normal behavior of the components is represented as propositional rules, where Nab(X) states the normal behavior of the component X [67]:

```
% Inverter I1
Nab(I1), val(b,false) -> val(d,true).
Nab(I1), val(b,true) -> val(d,false).
Nab(I1), val(d,true) -> val(b,false).
Nab(I1), val(d,false) -> val(b,true).
% And gate A1
Nab(A1), val(a,true), val(d,true) -> val(e,true).
Nab(A1), val(a,false) -> val(e,false).
Nab(A1), val(d,false) -> val(e,false).
Nab(A1), val(e,true) -> val(a,true).
Nab(A1), val(e,true) -> val(d,true).
Nab(A1), val(e,false), val(a,true) -> val(d,false).
Nab(A1), val(e,false), val(d,true) -> val(a,false).
% Inverter I2
Nab(I2), val(e,true) -> val(f,false).
Nab(I2), val(e,false) -> val(f,true).
```

```
Nab(I2), val(f,true) -> val(e,false).
Nab(I2), val(f,false) -> val(e,true).
% And gate A2
Nab(A2), val(c,true), val(d,true) -> val(g,true).
Nab(A2), val(c,false) -> val(g,false).
Nab(A2), val(d,false) -> val(g,false).
Nab(A2), val(g,true) -> val(c,true).
Nab(A2), val(g,true) -> val(d,true).
Nab(A2), val(g,false), val(d,true) -> val(c,false).
Nab(A2), val(g,false), val(c,true) -> val(d,false).
```

Using the observations *val(a, true)*, *val(a, true)*, *val(b, true)*, *val(c, true)*, *val(f, false)*, and *val(g, true)* leads to the following conflicts:

```
{Nab(I1), Nab(A2)}
{Nab(I1), Nab(A1), Nab(I2)}
```

By computing the minimal hitting sets $\{Nab(I1)\}, \{Nab(A2), Nab(A1)\}, and \{Nab(A2), Nab(I2)\}$ it can be concluded, that I1 alone, A1 and A2, or A2 and I2 are faulty.

2.2.2. Abductive Model-Based Diagnosis

The abductive approach defines a diagnosis result as a set of causes that imply the symptom itself [48]. The difference between consistency-based and abductive diagnosis is, that the consistency-based method requires only knowledge about the correct behavior of the system where abductive approach additionally requires knowledge about the faulty behavior of the components. Compared to the consistency-based approach, the abductive method requires a stronger relationship between diagnosis and observation. It requires knowledge of the faulty behavior in order to derive explanations for the observed symptoms by relying on the notion of logical entailment (see Definition 10).

Definition 10. Logical entailment [29]: *A set of premises ψ logically entails a conclusion ϕ if and only if for any interpretation in which ψ holds ϕ is also true. This relation is written as $\psi \models \phi$ and ϕ is called a logical consequence of ψ.*

Similar to the definitions of the propositional Horn clause abduction problem (PHCAP) proposed by Friedrich, Gottlob, and Nejdl [15], Wotawa, Rodriguez-Roda, and Comas [68] defined an abductive knowledge base as follows.

Definition 11. Knowledge base [68]: *A knowledge base (KB) is a tuple (A,Hyp,Th) where A denotes the set of propositional variables, Hyp* $\subseteq$ *A the set of hypotheses, and Th the set of Horn clause sentences over A.*

The set of hypotheses describe the possible causes in form of propositional variables, which can either be true or false. The theory *Th* represents the system description containing rules to describe the connections between hypotheses and their effects. A diagnosis problem is formed by a KB and a set of observations.

Definition 12. Propositional Horn Clause Abduction Problem (PHCAP) [68]: *Given a knowledge base (A,Hyp,Th) and a set of observations Obs* $\subseteq$ *A then the tuple (A,Hyp,Th,Obs) forms a propositional Horn clause abduction problem (PHCAP).*

A solution Δ for a PHCAP is a set of hypotheses which logically entails the observations *Obs* together with the theory *Th*. Since anything can be deduced from inconsistencies, only consistent solutions are taken into account.

Definition 13. Diagnosis [68]: *Given a PHCAP (A,Hyp,Th,Obs). A set* $\Delta \subseteq Hyp$ *is a solution if and only if* $\Delta \cup Th \models Obs$ *and* $\Delta \cup Th \not\models \bot$. *A solution* Δ *is parsimonious or minimal if and only if no set* $\Delta' \subset \Delta$ *is a solution.*

As the results of a diagnosis may lead to an exponential number of explanations, methods are required to lower the possible solutions and support efficient fault identification.
First, the space of possible solutions is decreased by inserting additional observations to the diagnosis engine. To do so, probing points with the highest entropy (i.e. the highest information gain) are computed in order to select observations with high discrimination capabilities [29].

Definition 14. Observation discrimination [29]: *Given a PHCAP (A,Hyp,Th,Obs) and two diagnoses* Δ_1 *and* Δ_2. *A new observation* $o \in A \setminus Obs$ *discriminates two diagnoses if and only if* Δ *is a diagnosis for (A,Hyp,Th,Obs* $\cup \{o\}$*) but* Δ_2 *is not.*

The calculation of the entropy for an observation is shown in Equation 2.1. The probability of an observation p(o) is defined as the ratio between the set of possible explanations and the total number of explanations.

$$H(o) = -p(o) \times \log_2 p(o) - (1 - p(o)) \times \log_2(1 - p(o)) \tag{2.1}$$

$$p(o) = \frac{|\{\Delta \mid \Delta \in \Delta\text{-Set}, \Delta \cup \text{Th} \models \{o\}\}|}{|\Delta\text{-Set}|} \tag{2.2}$$

It can be assumed, that in most practical scenarios multiple diagnosis results will be computed. In order to ensure efficient maintenance activities, a prioritization of the diagnosis results is performed. Koitz et al. [29] show the ranking of diagnosis results using Bayes' rule for conditional probability. Assuming independence amongst faults and uncertainty in the measurement, the probability of each result is calculated using the a priori probabilities p(h) of the hypotheses, shown in Equation 2.3.

$$p(\Delta) = \prod_{h \in \Delta} p(h) \prod_{h \notin \Delta} (1 - p(h)) \tag{2.3}$$

The probabilities of all results are computed and ranked accordingly. If information such as failure likelihood, costs for repair / replacement, and the like is available, these data can be used to influence the ranking of the diagnosis solutions as well [29].

Example

Koitz et al. [29] give an example diagnosing lubrication issues of a wind turbine gearbox. Lubrication is important as it protects the gears and bearings of the gearbox from mechanical damage and it is also responsible for the cooling of internal components. Consider the following fault scenarios:

1. A damaged oil pump leads to a loss of oil pressure and causes an oil flow reduction through the system.

2. A blocked filter inside the oil cooling system causes overheating of the oil which reduces the oil film thickness on the components.

The description of the scenarios leads to the root causes *blocked filter* and *damaged oil pump* which form the set of hypotheses, e.g. the faults that can be identified during the diagnosis process.

$Hyp = damaged\_pump, blocked\_filter$

Effects and hypotheses are part of the set of propositional variables A:

$$A = \left\{ \begin{array}{c} damaged\_pump, blocked\_filter, reduced\_pressure, overheating, \\ reduced\_film\_thickness\_bearing\_contacts, \\ reduced\_film\_thickness\_gear\_contacts, poor\_lubrication \end{array} \right\}$$

The Horn theory Th represents the conditions leading to insufficient lubrication:

$$Th = \left\{ \begin{array}{c} damaged\_pump \rightarrow reduced\_pressure, \\ reduced\_pressure \rightarrow poor\_lubrication, \\ blocked\_filter \rightarrow overheating, \\ overheating \rightarrow reduced\_film\_thickness\_bearing\_contacts; \\ overheating \rightarrow reduced\_film\_thickness\_gear\_contacts, \\ reduced\_film\_thickness\_bearing\_contacts \wedge \\ reduced\_film\_thickness\_gear\_contacts \rightarrow poor\_lubrication \end{array} \right\}$$

Given the observation $Obs = \{poor\_lubrication\}$, two minimal explanations, e.g. solutions to the PHCAP can be derived.

Δ-Set $= \{\{damaged\_pump\}, \{blocked\_filter\}\}$

The diagnosis results state that either a damaged oil pump causes a reduced oil flow or a blocked filter is responsible for poor oil cooling.

3. Related Work

Although MBD is an active research topic, the use of this technology in industrial applications is still insignificant. In recent years, there have been several attempts to raise the awareness of MBD systems. The INDIA (Intelligent Diagnosis in Industrial Applications) project attempted to bring MBD to industrial diagnosis applications. During the project, eight German academic and industrial partners collaborated on research and development of models and model-based reasoning techniques for real industrial applications [21] as well as integrating MBD into existing work processes [33]. The MONET project attempted to provide a technological strategy plan for MBD systems in industrial applications. In total, 50 member organizations from European universities, high technology enterprises and end users participated the project. Travé-Massuyès [61] states that the technical, human, and economical gap could not be closed during the project. Today, successful industrial model-based diagnosis applications are still barely developed.

This chapter presents interfaces of knowledge-based diagnosis systems as well as tools for model-based diagnosis. Since research on usability within MBD systems is unavailable, usability studies evaluating knowledge-based systems have been consulted. Some of the interfaces and their studies are presented and analyzed in the following sections.
Although industrial applications for MBD are sparse, several research tools and prototypes exist. Considering the interface of these tools, requirements for a general MBD application can be derived. Combining the findings from the interfaces of the knowledge-based applications and the MBD tools leads to the definition of surrounding conditions for designing a general interface for MBD systems.

3.1. Knowledge-Based Systems

Knowledge-based systems (KBS) apply rules to given facts in a knowledge base trying to solve real world problems. Developed in the 1960s, the first KBS were called expert systems. Expert systems were the first form of artificial intelligence and specifically designed to replace human experts. A KBS consists of a knowledge base and an inference engine [19]. The knowledge base provides facts about a domain, while the inference engine supplies logical rules connecting observations to results. These rules are typically represented by “IF-then” rules [44]. A KBS is developed by a knowledge engineer and a domain expert. The domain expert provides deep knowledge of a particular domain while the knowledge engineer then translates the knowledge into a computer usable language. The inference engine applies the rules to the knowledge-base and provides explanations for the user query.

KBS are used in a broad range of different scenarios, like diagnosis, patient support, interpretation, or prediction tasks [30]. The following chapter presents applications for KBS that cover some of the previously mentioned areas. A comparison of these systems can be found in Chapter 3.3.

3.1.1. Expert System for Tuberculosis Diagnosis

Osamor, Azeta and Ajulo [47] developed a rule-based expert system to diagnose tuberculosis. The web based system was designed to assist people without access to medical experts checking their health status. The implementation was verified by a usability study with doctors and patients. The Tuberculosis-Diagnostic Expert System consists of a knowledge database, a rule database, and a database for patient data. The inference engine receives user data and displays the processed response of the system to the user interface. The basic concept of the rule-based model consist of “if-then" statements. The statements represent heuristics that hold under certain circumstances. Osamor, Azeta and Ajulo [47] showed an example whether tuberculosis is suspected or not. The following arithmetic rule describe the given symptoms indicating a Mycobacterium infection. Therefore tuberculosis is suspected.

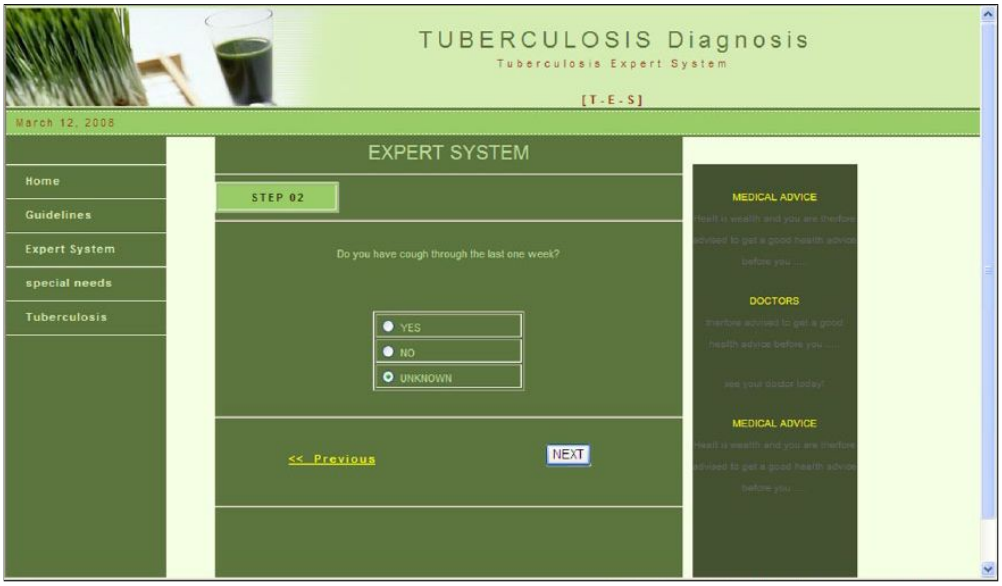


Figure 3.1.: GUI of the tuberculosis expert system [47]

If <symptom is cough AND
(NOT symptom is headache)
AND (symptom is bloody sputum)
AND symptom is swollen lymph/neck/joint>
then <Notify (Patient), "Tuberculosis is very likely, please go and see your doctor">.

Figure 3.1 shows the GUI of the tuberculosis expert system. The main area is placed in the middle column where the user is supposed to answer the displayed questions (e.g. "Do you have cough through the last one week?") by clicking on radio buttons. Possible answers are YES, NO, and UNKNOWN. Below the user can proceed or return to the previous question. The left column contains the navigation menu, where the user has access to additional information about the system. The right column contains medical advice and doctors' sections.

Evaluation

The usability test was performed via a user questionnaire in which four doctors and six patients participated. The questionnaire contained four sections (background information, effectiveness of the system, efficiency of the system and user satisfaction) with four questions each. The user was asked to answer the questions using a 5-point Likert scale, where 1 = strongly disagree, 2 = disagree, 3 = undecided, 4 = agree, and 5 = strongly agree. The survey questions and the evaluation results are shown in Table 3.1. The table contains the questions asked, the average rating (AVG), standard deviation (SD), and variance (VAR). The interpretation of the results is based on the work of Sauro and Kindlund [58]. The average result of the usability test was 4.08, therefore Osamor, Azeta and Ajulo [47] concluded the system has “Good Usability".

User Interface Analysis

The three-column layout of the tuberculosis expert system provides a good structure of the content. Users are able to navigate easily through the different screens using the navigation menu. The main content is located in the middle of the screen and additional information is displayed in the right most column. Overall, this layout is a good choice since it is standard for many web applications. However, there is plenty of unused screen space between the columns which creates an incomplete look of the system. The horizontal bar below the banner only contains the date, which is redundant information, as the date is displayed anyway in the taskbar of the operating system. The font size in the diagnosis section is too small and the chosen font color makes the questions hard to read, especially in combination with the dark background color. To respect a potential color blindness of users, green and red text colors should be avoided. Further, a font color with high contrast to the background color is needed to improve readability of the content. To remain consistent, the options for *Previous* and *Next* should be provided with the same style element, i.e. either both buttons or both text links.

Questions	AVG	SD	VAR
Background information			
Would you support the use of computer diagnosis for tuberculosis in your hospital? (yes/no)	6/4		
Do you need more computing skills/training/time to be able to use the system? (yes/no)	5/5		
Effectiveness			
I was able to complete my task successfully and correctly using the application.	4.10	0.57	0.32
The system did not show error message(s) while using it.	4.00	0.82	0.67
I was able to recover from my mistakes easily.	4.00	0.47	0.22
I feel comfortable using the application.	3.90	0.32	0.10
AVG	4.00	0.55	0.33
Efficiency			
Using the system saves me time.	4.10	0.57	0.32
I was able to complete my task on time.	4.00	0.47	0.22
I was well able to navigate the user interface on time when using the system.	4.20	0.42	0.18
I didn't have to carry out too many/difficult steps before completing my task.	4.10	0.57	0.32
AVG	4.10	0.51	0.26
User satisfaction			
The system was easy to learn.	4.00	0.67	0.44
The system was easy to use and user-friendly.	4.40	0.52	0.27
I am satisfied using the system.	3.90	0.57	0.32
I feel the system met my need.	4.30	0.48	0.23
AVG	4.15	0.56	0.32
Result AVG rating	4.08	0.54	0.30

Table 3.1.: Survey questions and results of the usability questionnaire of the Tuberculosis-Diagnostic system [47]

3.1.2. Expert System for Physical Examination of Skin Disease

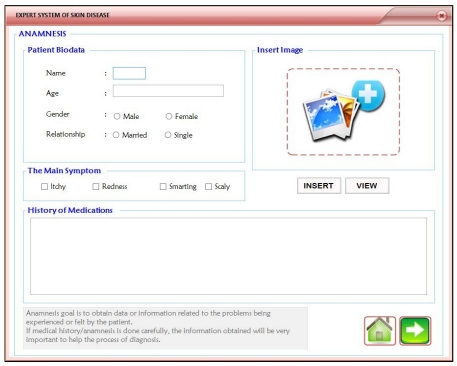
Suryani, Muhimmah and Kusumadewi [60] designed and evaluated two prototypes for physical examination of skin disease in a healthcare environment. The first prototype (prototype I) is a combination of several interaction style models such as a windowing system (WS), an icon-based (IB) system, a system menu (SM), a form-filling dialog (FFD), and natural language processing (NLP). A WS is an interaction style model for displaying information in one or more windows, SM is used to display a list of options, FFD is a screen like paper form for data entry and data retrieval, NLP enables interaction between user and computer, and the SBI model uses symbols to indicate the selection of specific activities. The second prototype (prototype II) is designed on a GUI-based dialog window. Prototype II reduces the amount of text input by replacing text input fields with an interactive graphical representation of the human body.

Workflow for Diagnosing a Skin Disease

Three steps are required to diagnose a skin disease with the support of the expert system: (1) acquire the patient's medical history, (2) perform the physical examination, and (3) diagnose the skin disease.

(1) Figure 3.2 shows the user interface of the first prototype for medical history review and physical examination. The location of the skin disease has to be typed in manually. Figure 3.3a depicts the second prototype where checkboxes can be ticked to identify the main symptoms of the patient. The location of the skin disease can be determined by a graphic representation of the human body, depicted in Figure 3.3b.
(2) Next, the physical examination needs to be performed. Prototype I (see Figure 3.4a) provides five drop-down menus for selecting type, shape, boundary, size, and color of the skin lesion. The middle section displays a scrollable list box in order to select other symptoms. Afterwards, the selection needs to be transferred to the right box. The group box *Palpation* on the bottom of the screen contains a text field and several radio buttons as well as the possibility to insert and view images.

Figure 3.2.: Layout for medical history review of prototype I [60]



(a) General information

(b) Visual representation

Figure 3.3.: Layout for medical history review of prototype II [60]

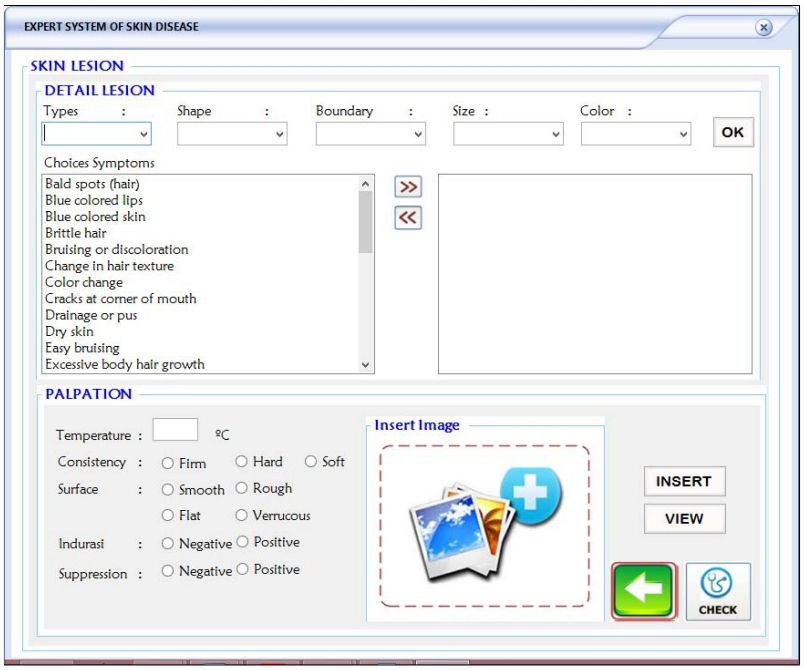
Prototype II simplifies the necessary user interactions (see Figure 3.5a). The list box for inserting lesions got replaced with a checkbox list. The items were structured in different categories and can be selected by ticking the check boxes. An info box on the bottom of the screen provides hints and useful information for the user. The three icon-based buttons on the bottom right can be used to navigate through the different screens. The group box *Other symptoms* and *Palpation* are located in a second screen. The buttons for moving the selected symptoms (see Figure 3.4a) are no longer icon-based to clarify their functionality.

(3) Last, the differential diagnosis is performed to identify the skin disease. The result screen is similar in both prototypes. The top section contains the summary of the inserted data and the differential diagnosis is displayed on the bottom of the screen (see Figure 3.4b and 3.5c).

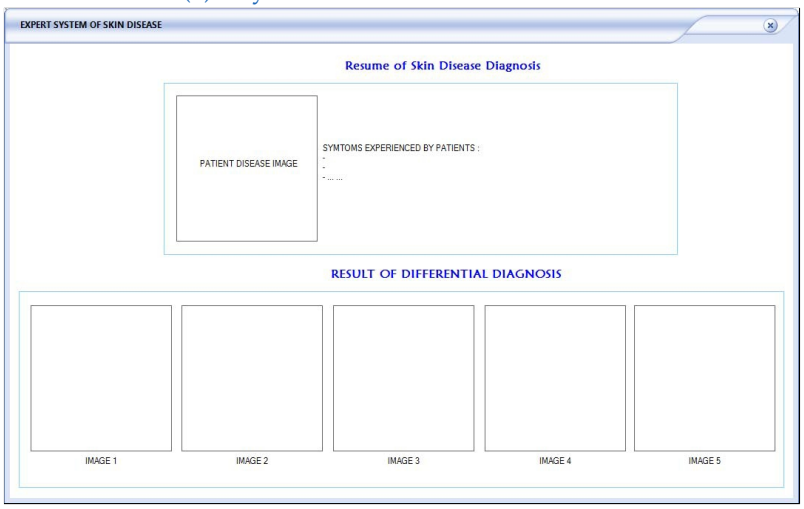
Evaluation

To evaluate the usability, a focus group discussion was performed for each prototype. The prototypes were presented to eight young doctors aged 23 to 25 years. Each of them had to assess each screen of both prototypes by filling out a technology acceptance model (TAM) [8] questionnaire. Each participant was also expected to provide their personal feedback and opinion on each prototype.

The results of the evaluation are depicted in Table 3.2. Prototype II was evaluated with an average of 81% while prototype I only scored 77% out of 100%. This shows that prototype II was preferred by the participants. It took less time to insert data and the GUI was more attractive compared to the prototype I. The GUI for physical examination shows higher user acceptance than the first prototype. It could be concluded, that this prototype supports the users' needs by providing the right information at the right time. It took less time to get used to the system and the users were interested in future use of the software.
During the focus group discussion the participating doctors requested an additional feature: the system should support a complete medical history review of the patient and his or her family members.

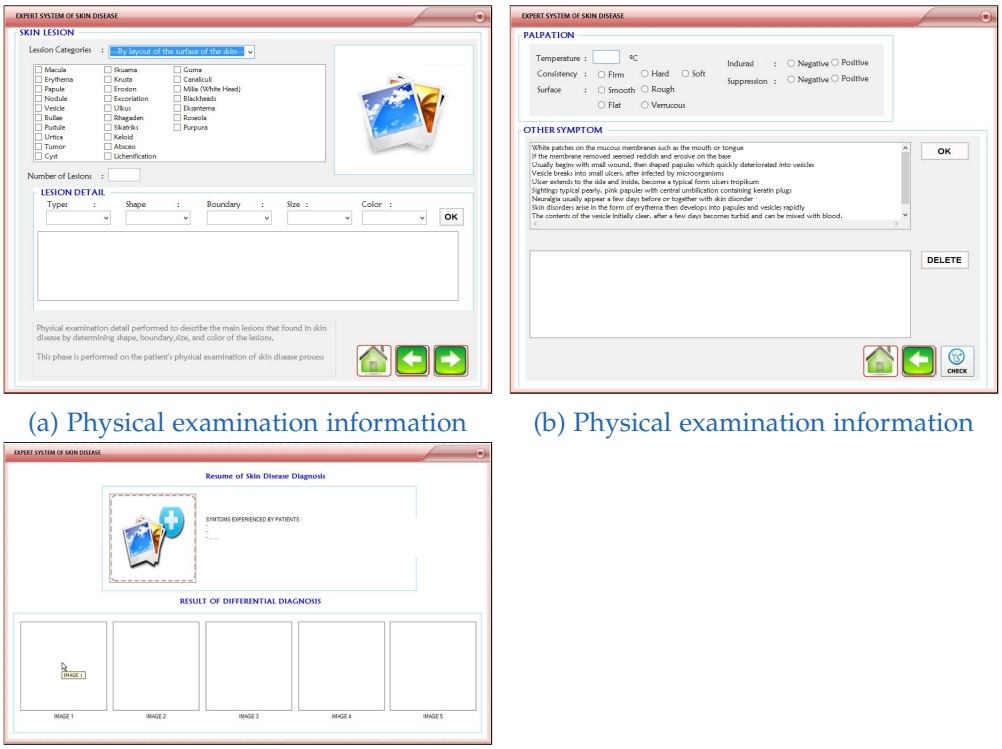


(a) Physical examination information



(b) Differential diagnosis

Figure 3.4.: Layout of prototype I [60]



(a) Physical examination information

(b) Physical examination information

(c) Differential diagnosis

Figure 3.5.: Layout of prototype II [60]

Prototype	Details of prototype	AVG [in %]
I	medical history review of physical examination	78
	physical examination	75
	for differential diagnosis	78
II	medical history review	79
	physical examination step 1	84
	physical examination step 2	81
	physical examination step 3	79
	differential diagnosis	82

Table 3.2.: Results of the usability questionnaire [60]

User Interface Analysis

The application provides a structured layout based on group boxes. Generally, prototype II was designed to offer a more natural way of inserting information compared with prototype I. The groupbox *Insert Image* (depiced in Figure 3.3a) only contains a placeholder image while the buttons below are placed outside the groupbox. A user might expect that clicking on the image will open the "insert-image" dialog, since this is the common behaviour of many applications. Both prototypes contain listboxes with several items. A search bar should be provided to minimize the users search time and avoid frustration. The Figure 3.5b shows two listboxes inside the groupbox *OTHER SYMPTOM* with the buttons *OK* and *DELETE*. The text inside the listbox is unnecessary long and could be replaced by keywords. Again, a search bar should be provided to support the user. The buttons on the right side should be renamed to *Insert* and *Remove* since *OK* and *DELETE* is semantically misleading.

3.1.3. Fault Diagnosis in Manufacturing Industry

Kluge and Termer [25] describe the design and the evaluation of a mobile application which supports maintenance workers in the fault finding process. A human-centered design approach was used to create the prototype. The evaluation of the mobile application with 42 maintenance workers showed that the problem solving task could be done twice as fast.

The design process included four major activities. First, an analysis was performed to identify the needs of the users, under which conditions the software will be used, and which features the users expect from it. After that, the prototype was designed and the last step was design evaluation with real users. In order to create software that is accepted, the system must be able to enhance the cognitive and associative skills of the users. Two theoretical constructs have been used to assess technology acceptance. First, the technology acceptance model (TAM) by Davis [9] describes how to lead users to accept and voluntarily use new information systems. Second, the

task-technology fit model (TTF) by Goodhue and Thompson [18] identifies the characteristics of technology in order to have a positive impact on the task solving performance. In particular, new technology will only be accepted by the users if the individual performance is enhanced or workload is relieved. Kluge and Termer assumed, that their information system will improve the fault finding process of the employees. The application is expected to support users to solve fault scenarios faster and make fewer errors than users which do not have access to the system. The hypotheses are stated as follows:

1. The group using the mobile application will solve the fault scenarios faster than the group without the application
2. The group using the mobile application will make fewer mistakes compared to the other group

Since the consideration and integration of target groups is an integral part of human-centered design, maintenance workers and subject matter experts (SMEs) are involved in the development stage of the fault diagnosis tool. Further, it is assumed that the input of domain experts will have a positive impact on the technology acceptance and task-technology fit. With the expertise of the SMEs, fault scenarios were developed. In order to test the performance of the software, two groups of maintenance workers had to solve these scenarios. The first group (experimental group) used the developed software and the second group (control group) used the current fault finding process. The time required to complete a scenario and the errors made during the fault diagnosis were recorded for both groups and compared against each other.

Implementation Process

During in-depth interviews with eight maintenance workers, the workflow of the diagnosis process was charted. The goal was to gain information about the communication between workers, how potential causes for a fault can be identified, which information is accessible, and how a fault is fixed. It was discovered that the plant operator calls the maintenance personnel and provides initial information such as symptoms and what kind of investigation was already performed. Thus, the maintenance worker is reliant on the

qualitative and quantitative information provided by the plant operator. Experience is stated to be the most important factor to reduce diagnosis time. An experienced maintenance worker is able to predict the right solution. The maintenance workers suggested to place manuals and error code lists closer to the manufacturing station, since they currently use a bicycle to reach them within the manufacturing hall. Some robotic systems provide information which is accessible via touch screen interfaces, which are reported to be not very user-friendly. Other systems provide error codes, but do not provide information about causes or repair instructions. Some maintenance workers carry handwritten notebooks where they note frequently occurring error codes and instructions for repair tasks. Although it is not allowed to use private smartphones within the manufacturing hall, some workers use apps from suppliers to have quicker access to instruction manuals. Maintenance workers reported, that the use of a smartphone would reduce the fault-finding process. Thus, Kluge and Termer decided to create a mobile application to support fault finding process.

Test Scenarios

In order to test the user interface of the diagnosis system, three fault scenarios were created. The fault scenarios represent the majority of possible faults. Three SMEs with several years of work experience were interviewed to gain knowledge about the problem solving process in respect to the fault diagnosis task. The interviews followed the Critical Decision Method [7], a structured interview method to focus on non-routine, challenging events. The interview is structured into four stages:

(1) Incident identification: Select non-routine events

(2) Timeline: Decision point verification

(3) Deepening: Get deeper understanding of events and decisions

(4) "What if" queries: Hypothetical questions to illuminate expert differences

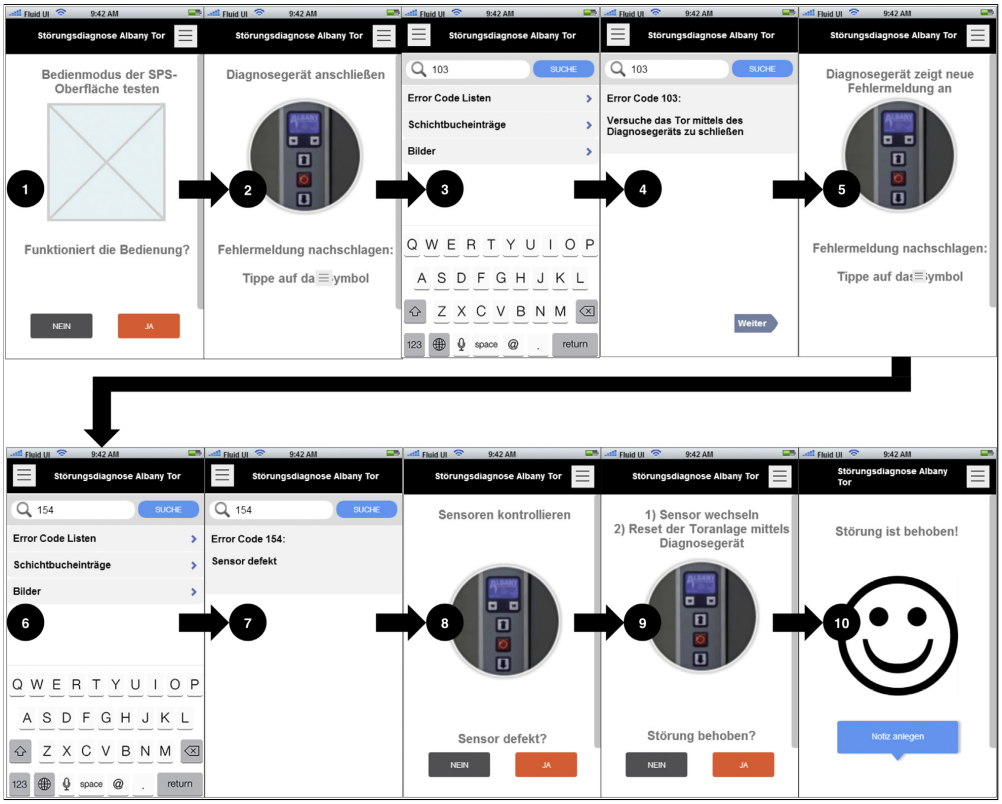


Figure 3.6.: User interface and fault-finding process of the diagnosis tool [25]

The SME provided information about the strategy used once a fault occurs, the available and relevant information, the symptoms, the probability for a certain cause of a fault, and the estimated time needed for the diagnosis task. The evaluation of the interviews provided information about the general requirements of the system and usability goals. The system should include a navigation, search function and visualization of faulty components. The workflow of the diagnosis process was modeled into a logical structure. A web-based prototyping framework was used to develop the layout and design of the mobile application. The rules and concepts of TAM and TTF were followed during the development stage. Hence, mockups were presented to the SME and the prototype was changed accordingly to their feedback.

Evaluation

To evaluate the usability of the prototype, a field study with 42 participants from the maintenance department of the manufacturer was conducted. The participants were divided into two separate groups, where the experimental group used the prototype to complete the tasks and the control group worked without the application. From each group, seven users were assigned to work on one of the three fault scenarios. The study was performed in a manufacturers' training center, where the three fault scenarios were implemented to ensure a consistent environment for all participants. The following three scenarios where simulated during the test:

- Scenario 1: Communication problem of several components caused by defective optical fiber.
- Scenario 2: Fault at the gate system caused by a defective cable. The system generated an error code and the gate could neither be opened nor closed.
- Scenario 3: Misalignment of robot axis caused by a defective connector in the communication module.

Scenario 1 is depicted in Figure 3.6. Each participant was individually asked to perform the tasks of one of the three fault scenarios, either with or without the use of the mobile application. The maintenance workers had a maximum time frame of twenty minutes to complete the task and were instructed not to talk to any peer workers about the contents of the study. The diagnosis performance was measured by timestamps in the application as well as manual time measurement. Further, errors were noted by an observer. Lastly, the time savings were computed for each scenario as well as the overall time saving. After completing the task, the participants were asked to fill out a questionnaire where they could rate the application according to their subjective impressions. The questionnaire was based on the recommendations of Davis [9] and was used to evaluate the prototype according to TAM and TTF. The TAM part contained two distinct scales. The first scale consisted of six items about perceived usefulness, measuring how much users believe

that an information system can enhance their work performance. Second, the ease of use scale consists of four items, measuring how easy it is for users to interact with a technology. To evaluate the TTF model, the users were asked to rate whether or not the right information was displayed at the right time, whether the level of detail met their requirements, and how difficult it was for them to interact with the software. Last, additional open questions were added to get feedback about the general impressions, particularly good and bad aspects or if there were any missing features.

Results

After the data analysis, the two hypothesis were proven for correctness. The first hypothesis claimed that users will complete the diagnosis task faster using the mobile application. The evaluation of the data showed, that users performed 24% better in Scenario 1, 47% better in Scenario 2 and 46% better in Scenario 3, compared with the users from the control group, which had to solve the task without using the mobile application. The second hypothesis stated that users supported by the diagnosis application will make fewer errors during the diagnosis task. This could also be confirmed, since the experimental group made 15 errors, whereas users from the control group made 36 errors in total. The analysis of the questionnaire (Likert scale from 1 to 5 where 5 denotes the ideal rating) showed a technology acceptance score of 4.1, a task-technology fit score of 3.1 and an ease of use of the software score of 4.57. The users rated the intuitive interaction with the application with a score of 4.34 and the visual attraction with 4.46. The evaluation of the open questions encouraged the data from above. The participants described a positive overall impression of the software, in particular that it enables faster diagnosis and is useful for inexperienced workers. The graphical representation of the faults and the clarity of the content was also positively remarked. Users criticized missing details for faults as well as the interaction with the mobile device. For additional features, a technical documentation, a tree structure of the diagnostic steps, and an extended level of detail to enhance learnability from the software was requested.

User Interface Analysis

This system was designed with flat navigation, which can be seen in many native applications for Apple's mobile operating system *iOS*. At the top of the screen, the navigation bar shows the description of the current diagnosis task and a menu button. The functionality of the button is to open the search bar where the user inserts error codes from the diagnosis device. In contrast to Google's mobile operating system Android, Apple's *iOS* styleguides[1] do not contain this type of button. The functionality of the iOS navigation bar is to navigate through the different screens of the application, but may contain controls like *search*, *edit*, or *done*. If no navigation is used (like in this prototype), a toolbar can be used to perform content relevant actions. According to the iOS guidelines, this toolbar has to be displayed at the bottom of the screen. In order to be consistent with the styleguide, either the navigation bar should be reworked or replaced by a toolbar. Both options require to change the menu icon to a magnifier icon, which is more strongly related to the underlying function. The interface is minimalistic and clean, i.e. only relevant information is displayed. The layout consists of a top section which displays the next action the users is supposed to perform, an image illustrating this action, and a bottom section where a question is asked to validate the performed action. The buttons to answer to questions are located at the bottom of the screen. The "Yes" button is colored orange and the "No" button is colored gray (see Figure 3.6). Usually orange or red colors are used to indicate an alert or danger. Thus, a neutral color for both buttons could help avoid irritation since questions are formulated positively and negatively. In some cases, a user might be unsure whether or not the error is resolved. An additional option *Skip* or *Unknown* should be added in order to prevent random user inputs.

3.2. Tools for Model-Based Diagnosis

The following section describes several tools that apply MBD. A comparison of the tools can be found in section 3.3.

[1] https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/

3.2.1. RODON

RODON is a model-based simulation, monitoring, and diagnosis tool developed by Uptime Solutions AB. As stated by Lunde [31], the major features of RODON are:

- Requirements analysis (design layout)
- Design verification
- Risk analysis
- Process monitoring
- Model-supported diagnosis
- Model-based diagnosis (on or off board)

RODON provides an equation-based object oriented language named Rodelica, which is related to Modelica [13]. Modelica is a language to describe physical systems in a component-oriented way. In contrast to Modelica, Rodelica is using constraints rather than differential equations to satisfy the requirements of MBD since the behavior of the model is often underdetermined. RODON's reasoning is based on the consistency-based MBD approach by determining if a consistent instantiation of the mathematical model is derivable using the data provided by system sensors. The system works properly, if the sensor values lead to a consistent solution. A fault is present in the system if no consistent solution can be found. RODON can perform a diagnosis by removing equations describing each component and solve the model with the provided sensor data. In order to determine the faulty component, the provided sensor data needs to lead to a consistent solution for a removed component. Additionally, fault modes can be included into the model. During a diagnosis, these fault modes are substituted for the nominal behavior of the component. The fault mode is presented as diagnosis result, if a consistent solution exists [50]. Rodelica's basic data type is defined as interval and not as real number. This value representation enables requirement analysis or design

verification since product specifications may contain tolerances. The model of the system is created in RODON's composer environment. The properties of components are described by model classes. These classes represent the components of the system and describe their behavior in form of constraints. The declarative modeling concept leads to simpler and more flexible models since components are modeled independently of their context and can be reused [31]. The development of classes can be done by writing the respective code in Rodelica or using the GUI to "drag and drop" classes from libraries to the model. Further, model topology can be imported automatically from CAD data. The component models are then chosen from existing libraries [32].

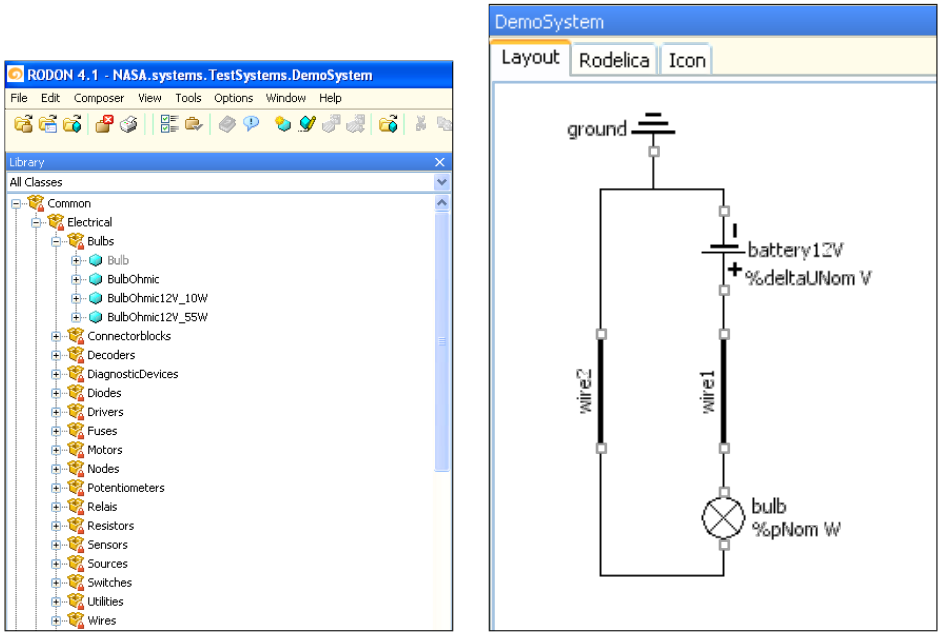
RODON Composer Environment

The model of a system can be created using the composer environment. RODON includes several standard libraries for electrical, hydraulical, and mechanical applications which contain a graphical representation as well as the behavior of the components. Figure 3.7a depicts the electrical library with the expanded bulb package. The components inside the package can be modified with additional attributes or different behavior constraints. Further, attributes or the behavior of classes (base classes) can be inherited.

The "drag and drop" feature of the RODON composer allows fast and simple development of a model. Figure 3.7b depicts a model of a simple electrical circuit. The circuit consists of a 12 Volt battery and a 10 Watt bulb connected by two wires, which may be disconnected. The bulb has three states:

- bright - if it consumes enough power
- off - if it consumes no power
- dimmed - otherwise

Each component behavior can be modified by manipulating the underlying Rodelica code. Listing 3.1 shows the Rodelica code of how the wire connecting the battery with the bulb has been modeled. It consists of two *WirePin* variables *p1* and *p2* which are the component's interface to the outside. The *FailureMode*



(a) Electrical library (b) Top level model

Figure 3.7.: RODON composer environment [22]

variable *fm* has the states *ok* and *disconnected*. The section *behaviour* describes the behaviour of the component according to the failure mode. If the failure mode is zero (i.e. *ok*), the sum of the current through the wire is zero and there is no voltage drop between *p1* and *p2*. Once the system is modeled, RODON can perform MBD, automatic generation of decision trees, and automatic generation of diagnostic rules for on-board diagnosis.

```
model WireBasic

public
  /*Electrical pin 1 of component*/
  WirePin p1;
  /*Electrical pin 2 of component*/
  WirePin p2;
  /* Failure mode variable*/
  FailureMode fm(max=1, mapping="ok, disconnected");

protected
  /*General pin type of wire to be replaced by individual connector types, so
that wires transferring different quantities basically can be described by the
same model. */
  replaceable connector WirePin = Pin;

behaviour
/*Constraints for health state "ok":*/
  if(fm == 0){
    /* Current balance: */
    p1.i + p2.i = 0;
    /*No voltage drop between pins 1 and 2:*/
    p1.u - p2.u = 0;
  }

  /*Constraints for failure mode "disconnected:*/
  if(fm == 1){
  /* There is no current: */
    p1.i = 0;
    p2.i = 0;
  }
end WireBasic
```

Listing 3.1: Rodelica code of the wire model [22]

RODON Analyzer Environment

Consider the model from Figure 3.7b. The system shows nominal behavior, either when the wires are connected to the battery and the bulb shines brightly, or when one of the wires is disconnected, leading to the bulb being off. To prove the correctness, the model can be loaded into the RODON analyzer environment. The constraints in the behavior sections (see Listing 3.1 as example for the wire model) describe the physical behavior of a component. The nominal behavior of the wire follows Kirchhoff's law, i.e. the sum of currents flowing into a node is equal to the sum of currents flowing out of that node. A different behavior indicates a faulty wire or, if the current is zero, a disconnected wire.

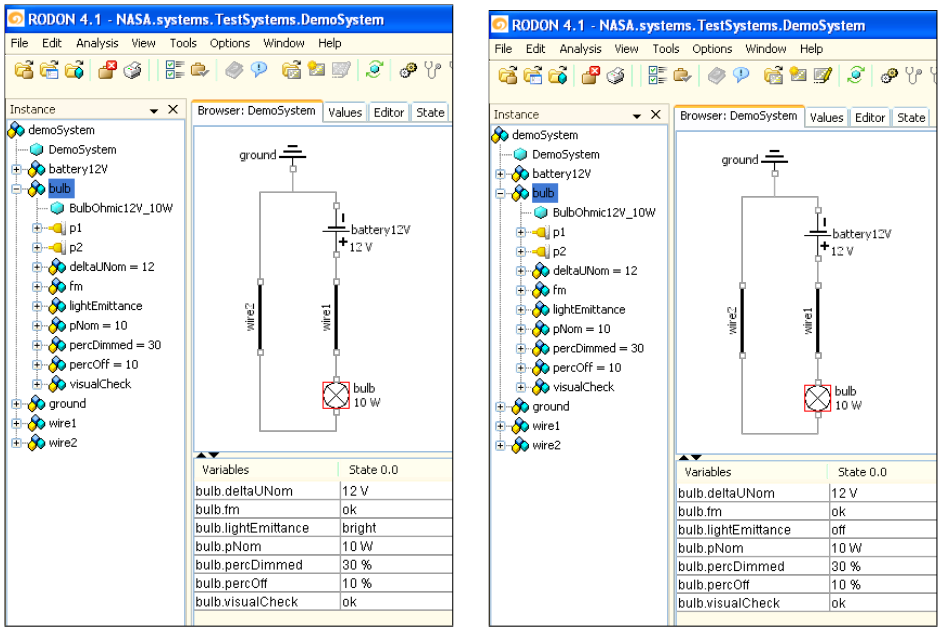
The behavior sections of all components form the equation system. Once the simulation is started, RODON tries to solve the equations. If the result does not lead to any inconsistencies, the nominal behavior of the system can be concluded. Figure 3.8 depicts the result of the simulation for nominal behavior. The simulation result for "both wires connected" leads to the state *bright* of the variable *bulb.lightEmittance* (see Figure 3.8a).

Disconnecting a wire from the bulb should prevent the bulb from shining. Figure 3.8b shows the result of the simulation with wire2 disconnected from the bulb. The status of *bulb.lightEmittance* changed to *off* as expected.

Consider the situation that all wires are connected properly but the bulb is still not shining. This behavior can not be described with the nominal behavior modes of the components. Thus, a conflict is created. RODON searches for combinations of failure modes, that are an explanation of the given observation. Figure 3.9 depicts the result of the diagnosis. Possible explanations for the observation *bulb off* are *bulb disconnected*, *wire1 disconnected*, and *wire2 disconnected*.

RODON User Interface Analysis

The layout of RODON consists of a menu bar, icon bar, tree structure, and a tab strip for the main content. This layout is used in many Windows applications. Several icons inside the icon bar are not easy to recognize for inexperienced



(a) Wire 2 connected (b) Wire 2 disconnected

Figure 3.8.: Simulations with RODON analyser [22]

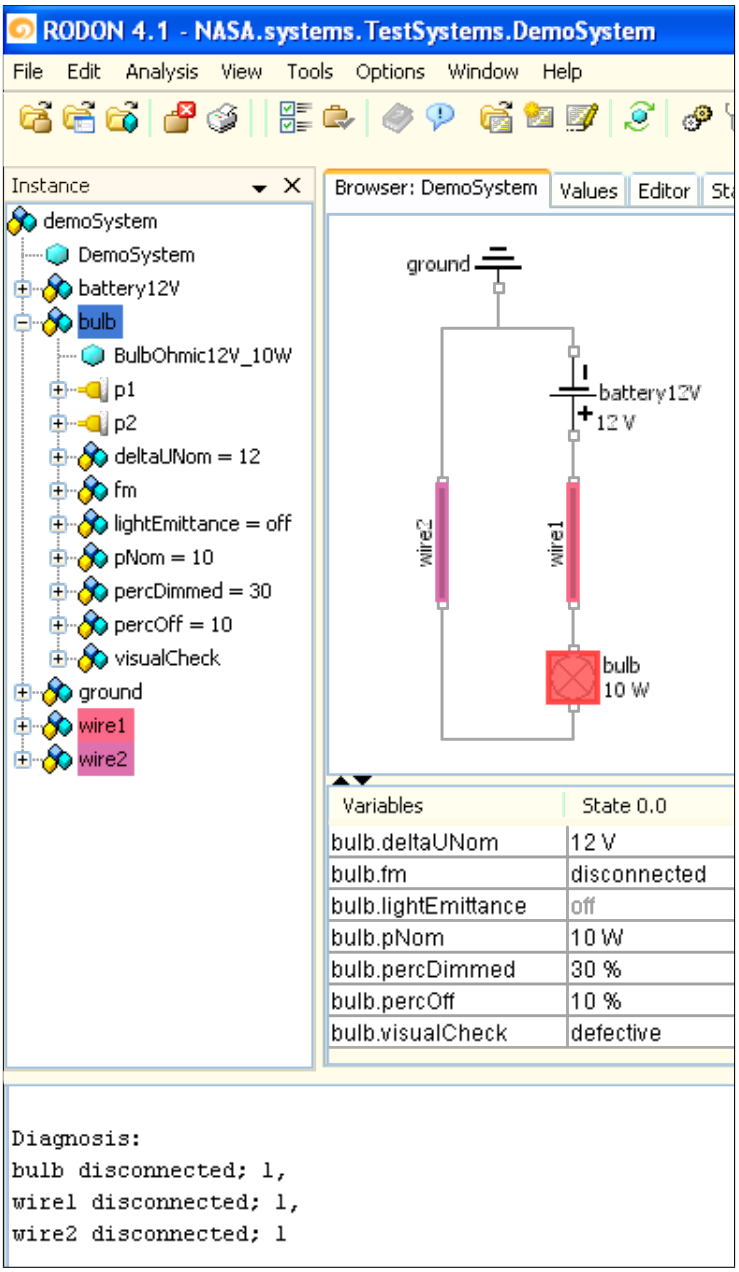


Figure 3.9.: Diagnosis with RODON analyser [22]

users. Microsoft faced the same problem in their office products and therefore changed the icon bar design in *Office 2007*, where a label was added to unrecognizable icons. Additionally, the tooltip was extended to provide a short description of the functionality of the item. A similar design could improve the users' understanding for the functionality behind the icons.

The tree structure on the left contains all used components and their parameters. The project label, components, and parameters share the same icon which could cause confusion to the users. The diagnosis result screen depicted in Figure 3.9 provides a good highlighting of the faulty components, unlike the confusing number ("1") the end of the diagnosis results.

3.2.2. Assumption-Based Truth Maintenance System

The Assumption-based Truth Maintenance System (ATMS) is a concept to deal with inconsistencies in logical theories. The facts describing a system are stored as nodes and the rules describe the connections between the nodes. A node represents a datum, i.e. the smallest unit of information of interest. Each node has a set of consistent environments where each environment contains a set of assumptions from which the node can be derived. The basic task of an ATMS is to compute minimal, consistent, sound, and complete labels for every node. The definitions below formalize the properties of the ATMS labels [67].

Definition 15. ***Consistent*** *A label for node is consistent if all of its environments are consistent.*

Definition 16. ***Sound*** *A label L for node n is sound iff n is derivable from every environment E L.*

$E \cup Th \models n$

Definition 17. ***Complete*** *A label L for a node n is complete iff every consistent environment $E \notin L$ for which $E \cup Th \models n$ is a superset of some E' in L, i.e., $E' \subset E$.*

Definition 18. ***Minimal*** *A label L for node n is minimal iff for every element E of L there exists no subset $E' \subset E$ from which n can be derived $E' \cup Th \models n$.*

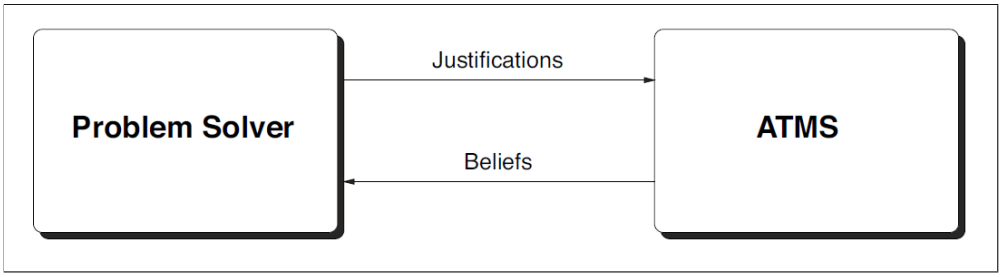


Figure 3.10.: Basic architecture of an ATMS system [66]

The labels are updated whenever a new rule is applied. Since two propositions cannot be true and false at the same time, contradictions might occur. These contradictions are represented within the NOGOOD node and can also be used for computing diagnoses [52]. The ATMS uses rules and facts represented as propositional Horn clauses. It identifies the beliefs and disbeliefs of propositions depending on the given assumptions. Since the ATMS performs deductions based on a node's validity, a problem solver is used to perform deduction grounded on the datum's semantics. The ATMS takes justifications (propositional rules and facts) and maintains consistency by adapting the truth state of the assumptions. The communication between the ATMS and the problem solver is depicted in Figure 3.10.

Logic Reasoning System (LRS)

The tool Logic Reasoning System (LRS)[2] developed by Wotawa [67] is written in Java and based on the ATMS by DeKleer [10] [11]. The GUI (depicted in Figure 3.11) contains a menu bar, a large text field for user input, and a smaller text field for status messages. The menu bar element *File* contains options to save and load models; the menu bar element *Proving* contains the option *Start*. Models can be represented using Prolog clauses or rules, where ',' is interpreted as the and-operator and '->' as an implication. Assumptions have to start with a capital letter. If propositions cannot be true at the same time, the predicate *false* can be used to specify the logical contradiction. Each line has to end with a '.'. Comments are indicated with a '%' sign.

[2] http://www.ist.tugraz.at/amor/software.html

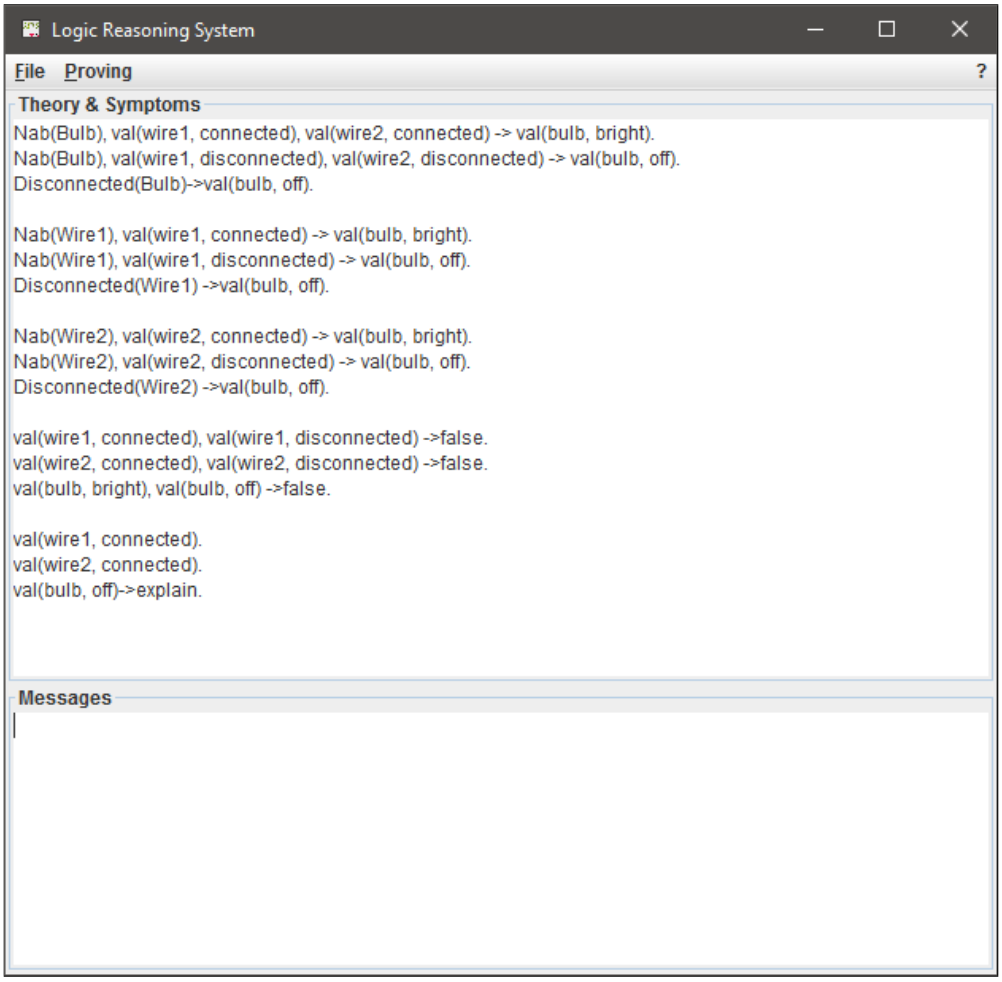


Figure 3.11.: User interface of LRS

Interface of the LRS

Consider the example from the previous section. The logical model is depicted in Figure 3.11. Again, the two wires are connected but the bulb is not shining. The statement *val(bulb, off)->explain.* is used to find explanations for this observation. Once the theorem proving has finished, a new window appears. It contains the nodes, which can be a proposition, an assumption or a NOGOOD. Since the model is designed for abductive diagnosis, the node *explain* is of interest. It shows explanations for the given observations, i.e. both wires are connected but the bulb is not shining. Figure 3.12 depicts the result window after proving has finished with the *explain* node expanded (per default, all nodes are collapsed). The result set contains the same explanations as already found in the previous section.

LRS User Interface Analysis

The user interface (Figure 3.11) of the LRS tool is very simple and clean. The two menu items *File* and *Proving* contain the necessary functionality. Since the *Proving* item only contains the option *Start*, this menu item is unnecessary. Instead, the option *Start* could be provided as button on the top level of the interface. This would also save one user interaction in order to run the diagnosis. The *'?'*-symbol on the top right could easily be overlooked by users but contains an option *Help*, which is currently not implemented. Debug and error messages are displayed in the *Messages* textbox at the bottom of the interface. Since this textbox is writeable as well, this could lead to confusion if users unintentionally type or copy content to it. Another issue is the height which scales with the height of the whole application frame. This can be uncomfortable when dealing with larger models and there is no option to change the height manually or hide the textbox. When a diagnosis is started, a second window appears on top of the LRS interface (see Figure 3.12). The result nodes (represented as folder-icons) contain the environments (represented as file-icons). Since for consistency-based diagnosis only the *NOGOOD*, and for abductive diagnosis only the *explain* node (see Figure 3.12) are of interest, the nodes represented as folder-icons should be highlighted to reduce users' search time. Most likely, a user will be interested in the results of only one of the folders. Therefore, an option could be implemented where

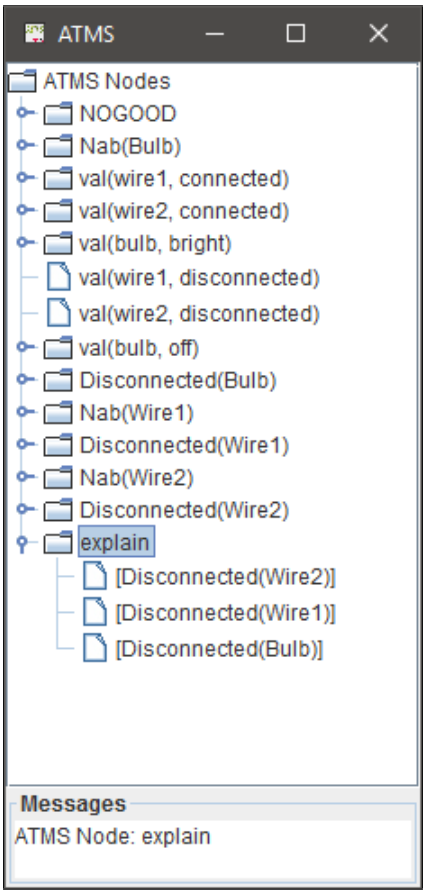


Figure 3.12.: Result window of LRS

the user can decide for which diagnosis type the model is designed. Based on this setting, folders can be rearranged and expanded to reduce periodic user interactions. Once the user starts a new diagnosis, LRS will open a new result window without closing the old one. This behavior can be used for comparison between different diagnosis results, but can easily lead to confusion, if various windows are open.

3.2.3. Diagnosis with Possible Conflicts (DxPCs)

DxPCs (Diagnosis with Possible Conflicts)[3] is a tool to perform consistency-based diagnosis of continuous dynamic systems, developed by Pulido, Alonso-González, and Bregon [51]. The tool itself is written in Java but relies on the Matlab environment to perform numerical simulations. It implements a comprehensive equation-based approach to MBD which incorporates fault detection and isolation capabilities for diagnosing continuous systems. The system behavior is modeled using Ordinary Differential Equations(ODEs). The residual signal computed from the model can be used for fault detection. A residual is the value between the real and estimated system output at a given time. Given a set of ODEs, DxPCs is able to generate an input/output model to create a set of Possible Conflicts (PC), which are computed offline and can become conflicts online. The system model M(Σ, U, Y, X, Z, Θ) requires the following parameters:

- a set of ODEs Σ
- a set of input measurements U
- a set of output measurements Y
- a set of state variables X
- a set of intermediate variables Z
- a set of fault parameters Θ

[3] https://www.infor.uva.es/~belar/SoftwareCPCs/PCs3.0_Setup.exe

In order to compute the PCs, the model M is abstracted to H_{SD} = {V, R}, where V is a set of variables and R is a set of relations among those variables. Two steps are required to compute the PC. First, a search for any minimally overdetermined subsets of equations in H_{SD} is performed. Those subsets are called Minimal Evaluable Chain (MEC). Second, for each MEC, every potential causal assignment for each relation is considered. If one globally valid causal assignment is found, it is called Minimal Evaluable Model (MEM). A PC is a set of constraints in a MEC, that gives rise to at least one MEM. If the tracking of one MEM detects a discrepancy, the corresponding PC is considered a real conflict. Through the set of PCs it is possible to compute the set of faulty candidates.

The features of DxPCs are:

- Create models for simulation and PC
- Simulate nominal and fault scenarios
- Visualize and store simulations
- Fault detection and localization using PC-based approach

DxPCs Interface

Pulido, Alonso-González, and Bregon [51] used the following example (depicted in Figure 3.13) to demonstrate the usage of DxPCs. Three tanks $T1$, $T2$, and $T3$ are connected via pipes q_1 and q_2. Water can run through the pipe to the different tanks if valves R_1 and R_2 are opened or closed accordingly. The input flow of T_1 is stated by q_i and the output flow is stated by q_3 of T_3. The fill levels hT_1*, hT_2*, and hT_3* of the tanks are measured by sensors. The behavior of the system is modeled using OEDs. Figure 3.14 depicts the interface of the DxPCs tool. The model of the system is shown on the left hand side inside the *System Model* box.

To model the transient behavior of the state-variables ($hT1$, $hT2$, and $hT3$), an explicit equation must be present to include the change over time. This is

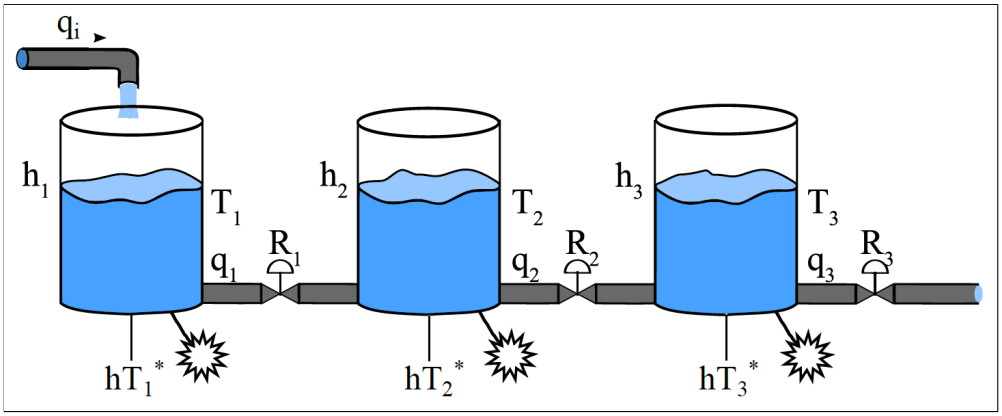


Figure 3.13.: Tank system example [51]

realized by the equations *d_hT1*, *d_hT2*, and *d_hT3*. There must be an explicit equation for the output variables (the sensor values hT_1*, hT_2*, and hT_3*) as well. To avoid confusion with the multiplication operator in Matlab, the notation *hT1mes* is used.

In order to perform a diagnosis, a diagnosis scenario must be defined. The setup of the scenario consists of an input file, a fault profile, and the initial conditions for the simulation. The input file contains different kinds of values for each input variable. The values can be fixed or periodically changing and may contain noise. The fault profile can provide single or multiple faults and contains the time instance, the ordinal number for faulty parameter, and the fault magnitude. The values of the state variables for the initial integration step and the nominal value for each fault parameter are provided by the initial conditions. To perform a diagnosis experiment, the system model, the set of PC models, and the diagnosis scenario are loaded into DxPCs. The PC simulation models are fed with the output values of the diagnosis scenario. For each simulation step, the residual is computed and analyzed. Once the residual activation is triggered, an incremental version of the minimal hitting set algorithm is used to compute the set of faulty components. The results of the simulation is depicted in Figure 3.15.

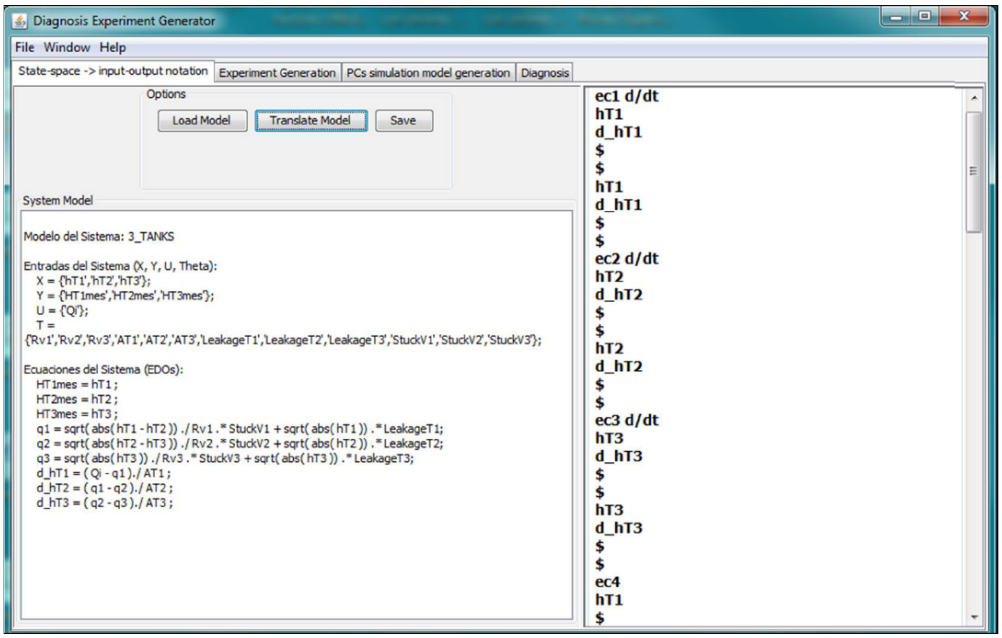


Figure 3.14.: Interface of DxPCs [51]

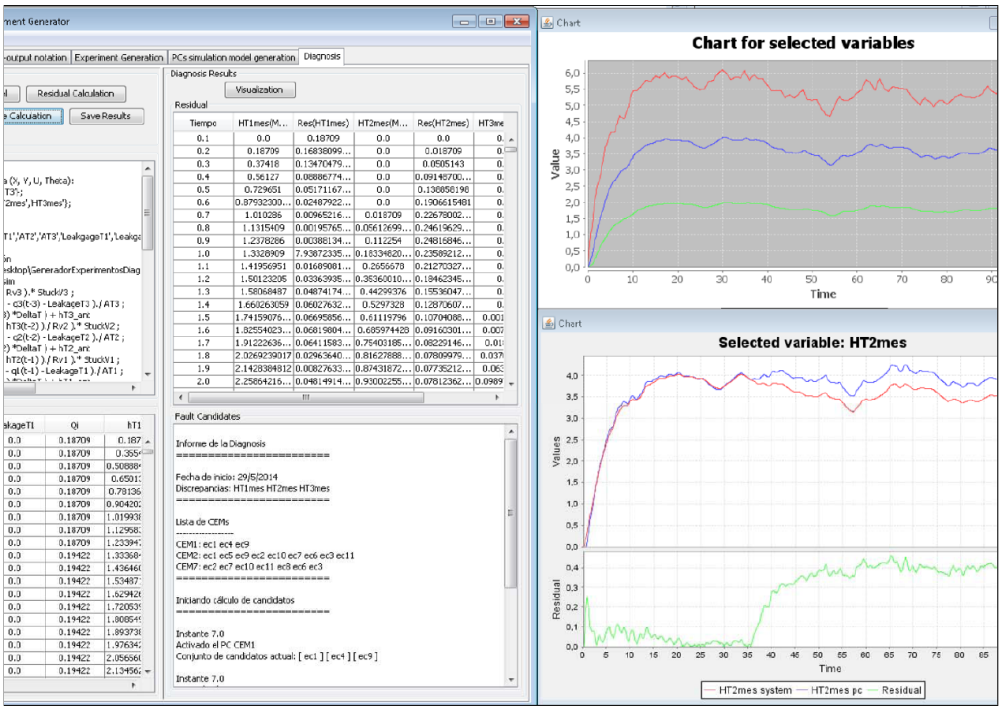


Figure 3.15.: Result the simulation [51]

DxPCs User Interface analysis

The layout of DxPCs is based on a tab strip (depicted in Figure 3.14), where each tab contains different elements and functionality. The groupbox *Options* inside the first tab contains buttons to load, save, and translate the model. A user would expect these options typically in an icon bar or inside the menu item *File*. In addition, there is unused screen space between the group boxes *Options* and *System Model* which could be used to display more content of the system model. The tables in Figure 3.15 contain the results of the simulation. In some cases, a user might want to investigate specific results; therefore a search option should be added. The Matlab generated charts displayed in external windows could be integrated into the layout. In order to make the charts more interactive, a option for range selection and zoom could be added.

3.2.4. JDiagEngine

The tool JDiagEngine[4] was developed by Weber and Wotawa [49]. It implements the consistency-based diagnosis approach of Reiter [52]. The diagnosis problem has to be prepared as a set of Horn clauses. Consider the example $a_1, ..., a_n \rightarrow b_1$. The implication denotes that if $a_1, ..., a_n$ is true, b_1 is true. While b_1 can represent only propositions, $a_1, ..., a_n$ may be propositions or assumptions. In order to negate a proposition, the prefix "$n\_$" can be added. The identifier for the negation can be changed in the interface. Assumptions define the operational mode of an item. NAB(X) represents the normal operation mode and AB(X) the abnormal mode. The interface provides a possibility to change this as well. JDiagEngine provides also a telnet-based interface which allows the program to run on an independent server.

JDiagEngine interface

Figure 3.16 depicts the interface of JDiagEngine. The interface consists of three textboxes to insert the propositions, system description, and observations. A

[4] http://www.ist.tugraz.at/modremas/downloads/jdiagengine.zip

settings box is located on the left hand side to define the maximum number and size of explanations. It also provides the option to customize prefixes. The buttons on the bottom left make it possible to check the consistency of the model and compute the minimal hitting sets. The two textboxes on the bottom of the screen contain the result set of the diagnosis. The menu bar item *File* contains open, reload, and save options for propositions, system description, and observations. It is not possible to save or load the complete scenario as in the previously described tools. Again, the example from the previous sections is modeled. Since JDiagEngine implements consistency-based diagnosis, the model requires only knowledge of the correct behavior of the system. Figure 3.15 shows the diagnosis results of the previously presented example. Since a consistency-based diagnosis was performed, the minimal hitting set denotes the components that cause inconsistencies with the given observations.

JDiagEngine User Interface Analysis

Similar to the tool LRS, the interface of JDiagEngine represents the model and the results in textboxes. In contrast to LRS, the user is able to change the width and height of the textboxes which is useful when dealing with larger models. The model is split into *Propositions*, *System Description*, and *Observations* where each of the parts are stored in different files. Unfortunately, a user has to save, load, or reload these files manually every time the file changes. The settings area on the left side of the interface is structured with group boxes *Settings* and *Edit Actions*. The latter is aligned to the bottom of the settings area which leads to plenty of unused screen space. Further, the group box *Edit Actions* contains a search box, which is semantically confusing. Generally, a user would expect a search box somewhere in the top-right corner or by pressing the keyboard combination *CTRL + F*. The textbox on the bottom left contains the result of the diagnosis, while the textbox on the bottom right displays no obvious data.

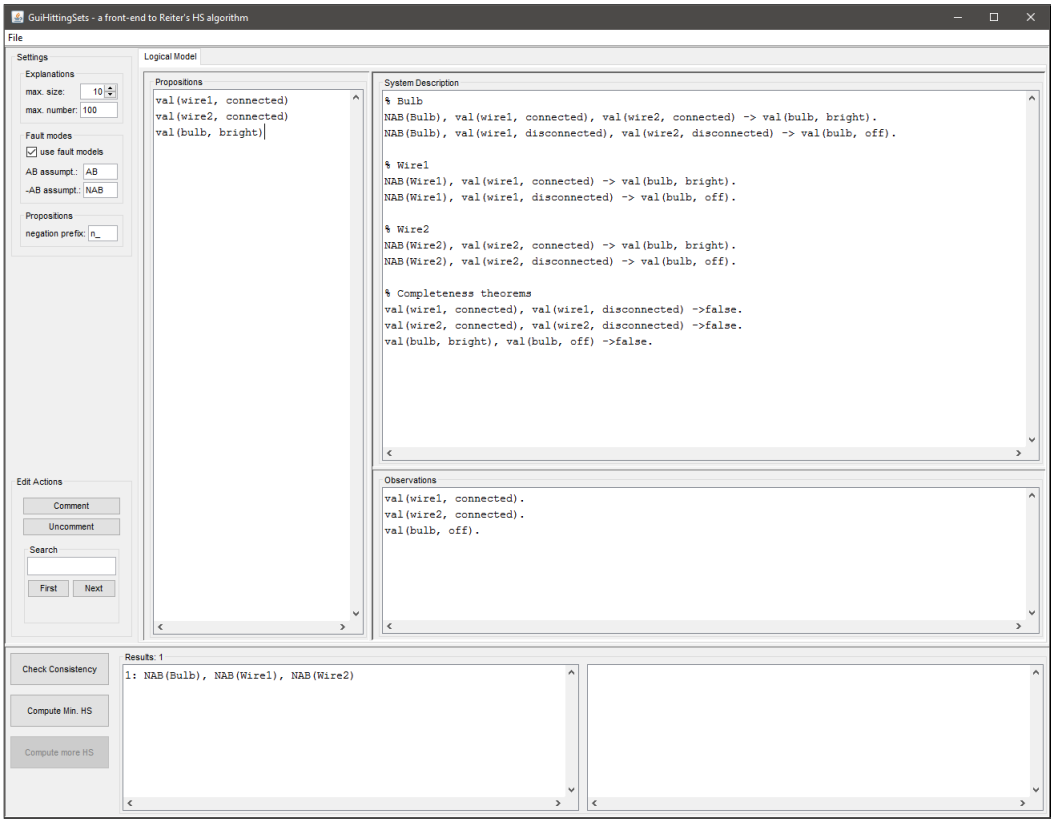


Figure 3.16.: JDiagEngine interface [49]

3.3. Discussion

The purpose of this work is to analyse existing diagnosis applications and identify best practices and mistakes to avoid. The gained impressions are important for the design process of a new user interface. The MBD tools introduced in Section 3.2 show how the behaviour of a system can be translated into a computer-usable language. The tool LRS provides a simple and clean user interface (see Figure 3.11). The icon bar from the RODON interface (see Figure 3.8) provides useful shortcuts, if icons are recognisable. Some icons are well known (like the printer icon), others are not. DxPCs does not use an icon bar but provides shortcuts using the tab strip. This ensures to provide only necessary options for the particular purpose of the tab. DxPCs provides a two-column layout, where the model is presented in the first column and the translated simulation model is displayed in the second column. This layout could be a useful feature for RODON, where the first column could contain the graphical representation and the second column could contain the textual representation of the model. LRS and JDiagEngine share almost the same interface, except JDiagEngine has separate textboxes for system description, propositions and observations while LRS combines them in one window.

The KBS introduced in Section 3.1 are diagnosis systems designed for a specific domain. The task of the tuberculosis expert system is to inform the user about a potential risk of getting tuberculosis. Apart from the obvious layout issues, the three-column layout is a good choice and standard for many web applications. Patients might have problems to read the tiny font, especially when green font color on green background is used (see Figure 3.1). The skin disease system is a good example for a window based application. The layout is based on group boxes which structures similar items. This supports the user's orientation when inserting data though multiple input methods. In order to avoid confusion, the groupboxes need to consistently contain items that belong together. The diagnosis system for manufacturing industry is designed to use on an Apple iPad. The application asks the user to enter one observation at a time, which ensures a straight-forward navigation. The three-section layout is consistent through out the whole application, containing explicit instructions and images to support the users' visual perception.

4. Usability

The term *Usability* denotes how well users can interact with a system to achieve certain goals. Nilsen [37] shows five attributes which are essential for a usable system, namely *Learnability*, *Efficiency*, *Memorability*, *Errors*, and *Satisfaction*. The ISO standard 9241-11 [23] defines usability as *"the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"*. Andrews [1] combined the attributes from Nilson and the ISO standard leading to the following six usability attributes:

1. **Learnability:**
   It should not take users long to learn the system. Providing an interface with a brief learning time shortens the period of time users need to start using the system efficiently.

2. **Efficiency:**
   Once the user has learned the application, a high level of productivity should be possible.

3. **Effectiveness:**
   Users should be able to achieve the goals they are aiming for.

4. **Memorability:**
   After a period of time of not using the system, users should be able to remember the application and not have to learn it again.

5. **Errors:**
   If users make mistakes, they should be able to easily recover from them.

6. **Satisfaction:**
   Users should feel comfortable using the system. In order to ensure that users like to use the application, subjective satisfaction is important especially in non-work systems.

In order to rate the usability of a system, these attributes need to be measured [38]. The attribute *Learnability* can be determined by recording the time a novice user needs to perform specific tasks. The results can be compared with the users' overall computer experience. To identify the *Efficiency* of a system, the time to complete tasks is measured and compared with the level of expertise of expert users. *Effectiveness* is rated by the number of tasks a user was able to perform successfully. The time casual users who have not used the system for a certain duration need to perform specific tasks determines the *Memorability*. Major and minor *Errors* users make while performing tasks can be counted and compared against the optimal path to complete the tasks. After using the system and performing some tasks, users can be asked about their opinions in order to gather information about the *Satisfaction*.
These attributes can be measured using usability inspection or test methods. While the former takes heuristics and guidelines into account, the latter requires real users to interact with the product to be evaluated. The different approaches and variants of these two methods are described in the following sections.

4.1. Usability Inspection Methods

In some cases it might be practicable to evaluate an interface without including real users. Therefore, usability inspection methods can be used where a group of usability experts analyses and judges an interface whether it follows specific usability principles [38]. These methods provide meaningful outcomes while being relatively easy and cost-effective to execute. Valid results can also be achieved by developers acting as evaluators since inspection methods are very easy to perform [41]. However, better results are usually obtained by usability experts.
The following sections provide an overview of the most common usability inspection methods.

4.1.1. Heuristic Evaluation

The most common usability inspection method is the *Heuristic Evaluation* developed by Nielson and Molich [43]. Usability experts evaluate whether each dialog of an application follows certain usability principles, i.e. heuristics. Nielson and Mack improved the heuristics resulting in the following ten usability principles [41]:

1. **Visibility of system status:**
   "The system should always keep users informed about what is going on, through appropriate feedback within reasonable time."

2. **Match between system and the real world:**
   "The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order."

3. **User control and freedom:**
   "Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo."

4. **Consistency and standards:**
   "Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions."

5. **Error prevention:**
   "Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action."

6. **Recognition rather than recall:**
   "Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system

should be visible or easily retrievable whenever appropriate."

7. **Flexibility and efficiency of use:**
   "Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions."

8. **Aesthetic and minimalist design:**
   "Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility."

9. **Help users recognize, diagnose, and recover from errors:**
   "Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution."

10. **Help and documentation:**
    "Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large."

Nielson showed, that the individual results of each evaluator reveal only up to 50% of the usability errors. In order to achieve sufficient results, the findings of each evaluator are consolidated in a large list of usability problems. The evaluators individually assign a rating based on the severity of the problem. Afterwards, the average rating for the each usability problem is calculated and the list is sorted accordingly. The resulting list of errors can be discussed to find possible solutions.

Performing a *Heuristic Evaluation* is a cost-effective and intuitive method to find usability problems. It can be performed in the early development stage, since a fully functional prototype is not necessarily required and also developers can act as evaluators although better results are usually gained when usability experts are evaluating the interface.

4.1.2. Cognitive / Pluralistic Walkthrough

In contrast to the previously described *Heuristic evaluation*, the *Cognitive Walkthrough* [63] is an inspection method used to determine how easy it is for new users to solve specific tasks when using the system for the first time. The participants of a *Cognitive Walkthrough* are a mixed group of designers, developers, and usability experts. In order to perform such a walkthrough, a task is chosen with defined start and end conditions. To complete the task, a series of steps is required by the user. After each step, the participants discuss the required interactions and answer the following questions:

1. Will the user be trying to achieve the right effect?
2. Will the user know that the correct action is available?
3. Will the user associate the correct action with the effect they are trying to achieve?
4. If the correct action is performed, will the user see that progress is being made towards to the solution of their task?

If one of the above questions leads to a negative answer, a usability error has been found. With the team of designers and developers, a redesign can be discussed at the end to resolve the issue.

The *Cognitive Walkthrough* is a valid method to find task-oriented usability problems. It can be performed in the early development stage since no working prototype is required. Analyzing an interface from a user's point of view requires some training and might be a challenge for the walkthrough participants.

The *Pluralistic Walkthrough* is an inspection method which involves a group of real users in the interface analysis process. This method was first presented by Bias [4] where Riihiaho [53] applied some important changes. Other participants are a group of developers, designers, and usability experts, where one person of this group acts as moderator of the inspection session. The basic setup of a *Pluralistic Walkthrough* is depicted in Figure 4.1. All evaluators need to be present in the same room and the moderator gives a brief overview

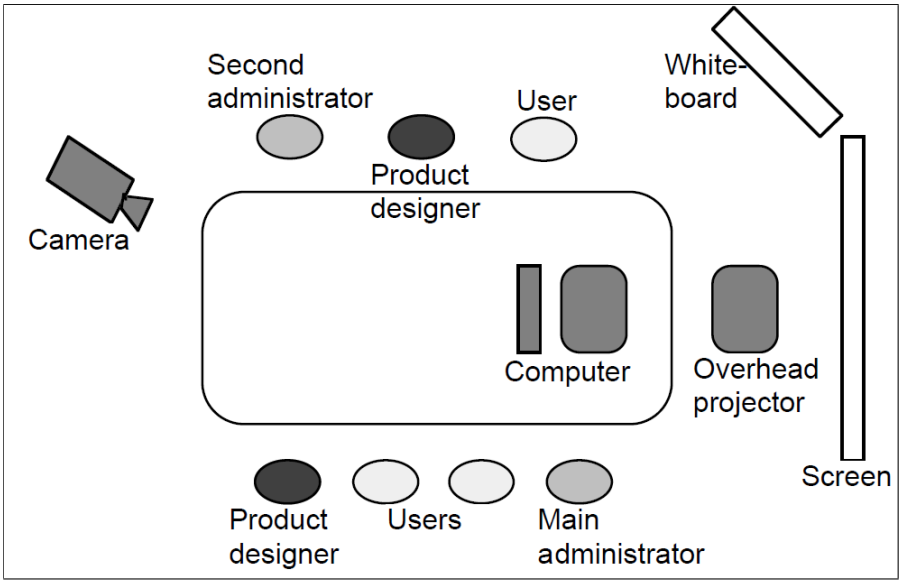


Figure 4.1.: Setup of a Pluralistic Walkthrough session [53]

of the general interface. The session starts by presenting hardcopy panels of the system. These are presented in the same order as they would appear in the real system. In order to simulate an interaction with the interface, each participant receives their own stack hardcopy panels, where they are asked to circle buttons or menu items they would click to proceed with the task. The task is presented and all participants write down all the required steps as detailed as possible to solve the task. The developers and usability experts try to solve the tasks from a user's point of view. The users start to present their solutions, followed by the findings of the other evaluators. Afterwards, the moderator reveals the optimal path and the divergences of the different solutions are discussed. During the discussion, potential usability errors can be identified as well as solutions for a redesign.

The *Pluralistic Walkthrough* is a suitable inspection method to gain insight into various human factors, since designers, usability experts, and users are participating in one session. Subsequently, design solutions can sometimes be developed on the fly. On the downside, the representatives of each group have to be present at the same time. Also, the group discussion can not start until

every participant has completed the task, which can slow down the progress of the session. Thus, the moderator needs to ensure that the group is working as a team and everybody stays in a good mood.

4.2. Usability Testing Methods

In contrast to the inspection methods presented in the previous sections, usability testing is an empirical technique which requires representative users. Designers think that a system is very easy to use based on their own experience. In reality, users might be less experienced with certain techniques or have different expectations of an application. Usability testing requires more effort than inspection methods, but provides an opportunity to analyze the behavior of users and understand why users interact differently than initially predicted. Usability tests can be performed in different stages of the development process. The following presents three types of usability tests [54]:

(1) **Formative Test:**
A formative usability test is performed in the early development stage with the purpose of analyzing the effectiveness of the design principles used. In order to perform such a test, a mockup of the interface or a clickable prototype is required, where users are asked to perform representative tasks. Since this test is performed so early in the development process, users can also "walk through" (i.e. review the prototype and answer questions from the moderator) the interface rather than try to solve tasks. The objective of a formative usability study is to identify confusing areas and get the user's opinion on how this issue can be solved. Further, it can be determined if the product supports the user's needs and if missing or redundant features are present.

(2) **Summative Test:**
In order to test how effectively a user is able to interact with an interface, summative usability tests are used. In contrast to the previously described formative test, this method always uses realistic tasks and less interaction between moderator and participant is preferred. Summative testing is also viable for collecting performance measurements.

(3) **Verification Test:**
Verification (or validation) testing is performed at the end of the development cycle. The main objective is to verify if the issues identified in the previous tests have been resolved. Further, performance data is compared against company standards, or used to set performance goals for further products. Another objective is to identify how well all components and features of a product work together, since they are often developed and tested independently from each other. Also, this test method is used to benchmark the risk of "hotfixes" or patches after the release.

To perform a usability test, it is essential to have a quiet room outside the usual working environment. All disturbance factors should be eliminated (e.g. disable phones, doorbells, ...) to keep the test users' attention focused on the system. Several test setups are suitable to perform a usability test, depending on the environmental circumstances. Figure 4.2 depicts a single room, single camera setup. The user sits in front of the computer monitor and the facilitator next to him. The test session is recorded via a screen capturing software, a microphone, and an external camera. Another option is to use an observer who follows a live stream of the monitor and takes notes of important events. Figure 4.3 depicts the setup within a usability lab, which consists of to room with separate doors. The usability experts enter the observer room from the left and the participants enter the test room from the right. The observers can follow the test session via live stream and through a one-way mirror. This setup is suitable for environments where usability tests are performed frequently.

Test users are typically divided into user groups based on their experience and characteristics. In order to get valid results, at least five users from each group should be tested [42]. Each user has to perform a set of realistic tasks within the system. Before the test starts, the facilitator gives an introduction about the method used and collects some data about the user with a background questionnaire. The first task is usually an introduction to make the user feel comfortable. Depending on the complexity of the system, a training task can be considered. Each task is presented to the user on a separate sheet in typical order. During the test, the facilitator and/or observer document the time required to complete a task, any mistakes that were made, important user comments, and other unexpected interactions. Andrews [1] presents a simple

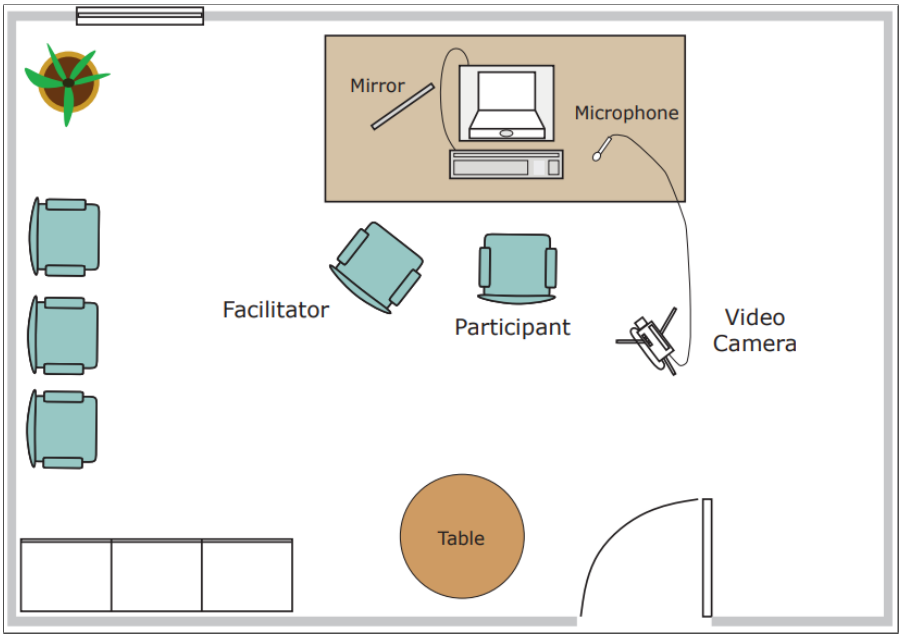


Figure 4.2.: Setup of a single room, single camera usability test [1]

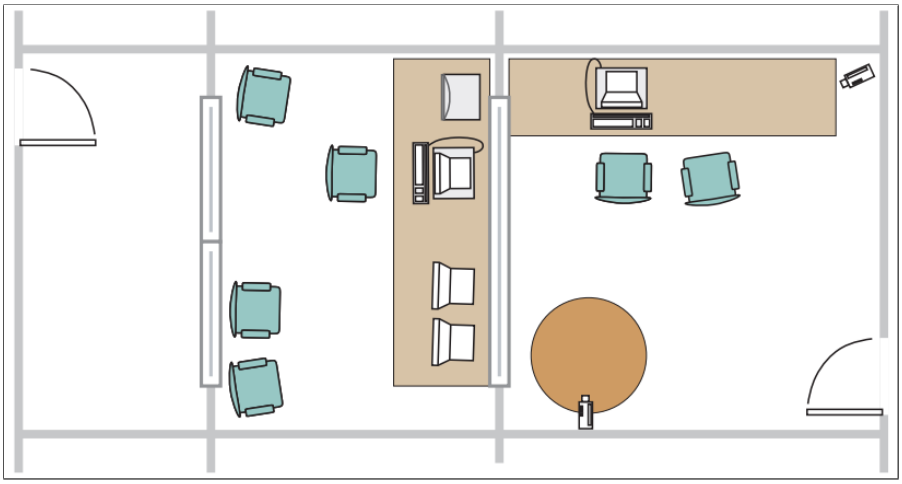


Figure 4.3.: Usability lab test setup [1]

Code	Event
S	Start of task.
E	End of task.
O	Observation (problem).
G	Positive impression.
Q	Verbatim user comment.
X	Error or unexpected action.
H	User given help by facilitator.
P	Prompted by test facilitator.
T	Timeout, exceeded maximum time.
?	Probe this activity during debriefing.
C	Comment by facilitator.
*	Very important action

Table 4.1.: Scheme for classifying events during a usability test

Test:		
Date:	Time:	
Task	Time	Observations

Table 4.2.: Sample form for collection data during a usability test [1]

scheme and form (see Table 4.1 and 4.2) for logging and classifying events. After the test the user is asked about his or her personal opinion on the system. Also unexpected behavior and other important events that occurred during the test can be discussed.

4.2.1. Thinking Aloud Test

Using one of the setups described in the previous section, users are asked to "think out loud" while trying to solve typical tasks. This means, that they should say what they are trying to do, ask questions that arise during a task, and mention what they find confusing. In contrast to *Formal Experiments* where users perform tasks without explaining their actions in order to be able

to evaluate the performance with statistical analysis, this method provides qualitative data about the users' expectations about the product and beliefs about every feature they use. The *Thinking aloud* method reveals a lot of usability errors and also gives information why these problems occur. It makes it possible to analyze the intention of users and understand their decision making process. On the downside, this method slows the users down since they are asked to talk all the time. Therefore, no performance data can be collected [38]. Some users may find it unnatural and confusing to work and speak simultaneously which can influence the concentration level of the participants [54]. Also, this method is more expensive and requires more effort compared to inspection methods.

4.2.2. A/B Testing

A/B testing is typically performed to evaluate changes in websites. Visitors are randomly assigned either to variant A (the standard interface) or variant B (the modified interface). Data (e.g. click rate, sales rate, ...) is automatically collected and compared over a longer period of time [26]. *A/B testing* is a very cheap method which allows to measure performance differences and the actual behavior of users under real conditions. On the downside, this method requires a fully implemented system and only automatically collected data can be used to confirm or negate the goals of a system.

4.2.3. Query Techniques

Query techniques are a cheap and useful extension to *Thinking aloud* tests. After the test, the user is asked about his personal opinion on the system either in form of an interview or a questionnaire. The *Post-Test Interview* is a flexible method which allows the facilitator to investigate interesting issues that came up during the test but is time-consuming and hard to compare.
In contrast, the *Post-Test Questionnaire* filled out by the user is less time-consuming, easy to analyze, compare, and repeat [1].

A popular post-test questionnaire is the *System Usability Scale (SUS)* [5]. This quantitative method consists of ten questions which have to be answered

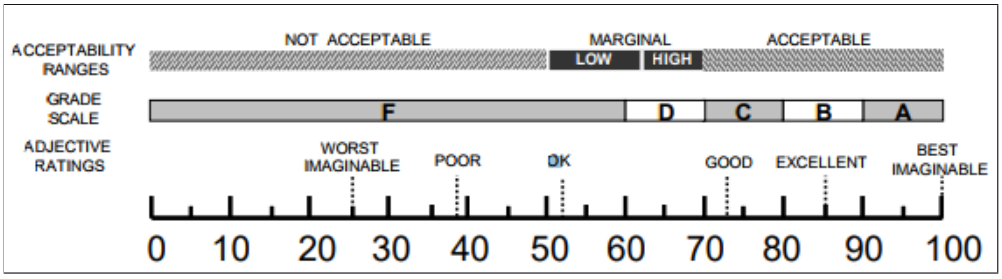


Figure 4.4.: Interpretation of the SUS score [2]

using a Likert scale from 1 to 5, where 1 is *totally disagree* and 5 is *totally agree*. The SUS questionnaire can be found in Appendix C. To evaluate the SUS, all odd numbered questions are assigned with values from 0 to 4 and all even numbered questions are assigned with 4 to 0. All scores are summed up and multiplied by the factor 2.5, which leads to a number between 0 and 100 as final result. It is worth mentioning that the SUS score cannot be interpreted as percentage value of how usable a system is. Research has shown that a score above 68 is considered to be above average [57]. Figure 4.4 shows the interpretation of the SUS score by applying different scales [2].

4.3. Mobile Usability Guidelines

In contrast to desktop programs, mobile applications have to be suitable for a small screen and are used in a dynamic environment. In order to ensure the same software to be usable for both desktop and mobile devices, Jendryschik [24] suggests the following six general mobile usability principles:

1. **Provide a mobile optimized interface:** In order to provide the best experience, a mobile optimized version of an interface should be available.

2. **Use same target location:** When dealing with web applications, the system should automatically detect the size of the screen and display the optimized page accordingly.

3. **Reduce text to minimum:** Users do not read long and complicated texts on small screens. The content should be reduced to a minimum in order to keep the users attention to the important information.

4. **Focus on primary content:** Since desktop interfaces can provide a large amount of functionality, mobile interfaces need to focus on the primary content. Also long loading times of content generates a negative user experience.

5. **Ensure recognition of interactable UI elements:** Most users know how to interact with standard UI elements (e.g. swipe gesture to navigate through a carousel gallery). The interface needs to indicate the elements the user can interact with and ensure that the UI element shows the behavior the user expects.

6. **Ensure that UI elements can be used with the thumb:** Most users interact with a mobile interface using only their thumbs. Therefore, the UI elements need to be large enough to prevent users from accidental interactions.

5. General Design of a Model-Based Diagnosis Application

The applications from Section 3.2 show the necessary features required in order to perform a model-based diagnosis. In order to integrate the MBD approach into industrial applications, Koitz and Wotawa [28] defined a process for real-world scenarios (depicted in Figure 5.1). The offline section states the generation of the knowledge base, which can be created automatically using Failure Mode Effect Analysis (FMEA). A FMEA is a table where each row consists of a component, a fault mode, and a set of effects the component is causing for the given fault mode [65]. The online part consists of the detection of an anomaly in the system, which can either be accomplished automatically (using a condition monitoring system) or manually. The observed symptoms together with the created model are then used to compute abductive diagnoses. Ranked diagnosis results can be obtained using a priori probabilities (see Section 2.2.2). In order to refine the diagnosis results, observations on probing points can be made and send to the diagnosis engine, which influence the probability, and therefore the ranking of the diagnoses.

Two main design goals were identified to apply abductive model-based diagnosis on a faulty system:

(1) The representation of faults: Most importantly, users need to get a clear idea of what is wrong with the system. Since the result might be a large number of faults, a vertical scrollable list is a suitable choice. The fault list may consume most of the vertical available screen space to indicate the importance of this information. The faults need to be specified in a simple and clear human-readable language based on the domain and the skill level of the target users. Error codes and similar terms need to be avoided, since most users cannot achieve any knowledge from such phrases. Additionally, users might want to know why the faults occur.

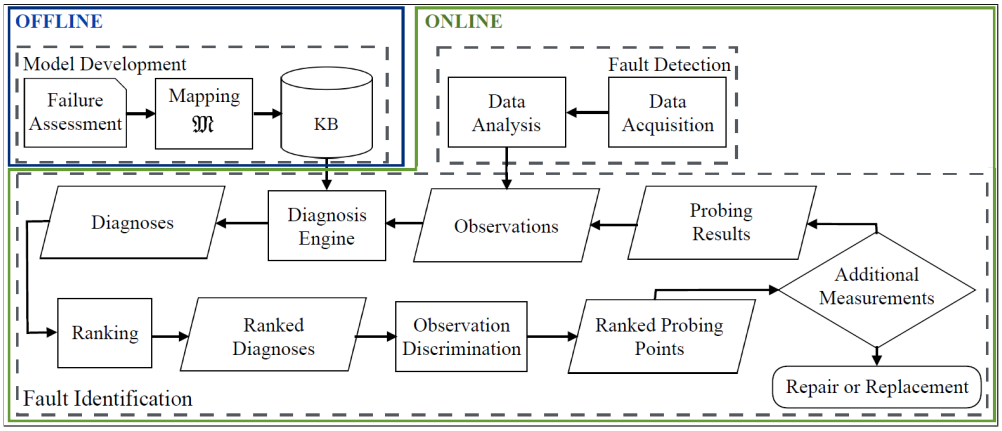


Figure 5.1.: Abductive approach for a MBD process [27]

This information is less important than the actual fault and might not be relevant for all users, therefore it can be displayed in a popup box or similar UI element.

(2) A possibility to discriminate faults: In order to discriminate faults, the user needs to perform observations on the real system and insert them into the diagnosis system. This is accomplished by asking the user if certain observations can be confirmed or negated. The application presented in Section 3.1.2 showed that users prefer to interact with a visual representation of the human body. Subsequently, visualizing components should assist users to find the right location for the observation to be made.

The following sections describe the requirements, design, and evaluation of a model-based application.

5.1. Identified Best Practices

In order to build the foundation for model-based diagnosis interfaces, the GUIs of the applications presented in Section 3 have been analyzed. The interfaces and the usability studies of the knowledge-based systems show

successful and problematic design approaches. Although the existing model-based tools were developed for users in the area of research and development, the user interfaces were investigated and compared in terms of functionality and usability. The gained information is used to identify best practices for the design of a new user interface for an MBD system. The requirements heavily depends on the environment where the diagnosis application will be used. For office use a desktop software is preferred due to stationary computer systems. If the environment changes, laptops as well as smartphones or tablet PCs can be used to perform diagnostic tasks. In Section 3.1.3, the evaluation has shown that mobile devices are suitable for diagnosis applications. Since the development of native applications for different operating systems is not cost-efficient and users might want to use the software on their own devices, an alternative solution needs to be found. For example, providing the diagnosis system as a web application allows users to work with their preferred devices. In order to support the different screen sizes of the users' devices, the web application needs to be fully responsive to take full advantage of the available screen space. Frameworks like Bootstrap[1], Foundation[2], or Skeleton[3] provide predefined classes which resize the content according to the device screen dimensions. Whenever the user is forced to select items from a list, the system should provide a search bar to support the users' search process. Input fields and options that belong together should be visually grouped together as described in Section 3.2.4 and 3.1.2. All GUI components should be displayed on the screen where a user most likely will expect it. If options are represented by icons, these need to be recognizable or replaced by buttons containing a meaningful label. Even though the evaluation in Section 3.1.1 showed, that the usability of an application does not necessarily depend on the design, the newly designed interface should have a professional look and feel. The frameworks mentioned before include style classes for common components which guarantee a consistent style throughout the whole system. The default behavior of GUI elements should not be changed to avoid confusion and ensure an intuitive user interaction. The application should be usable in a way that no additional help or tutorial is needed. In order to improve readability, the font size should not be smaller than 12 pixels and high contrast font colors

[1] http://getbootstrap.com/
[2] http://foundation.zurb.com/
[3] http://getskeleton.com/

are mandatory. To respect the potential color blindness of users, the colors red and green should not be used. As shown in Section 3.1.3, images can be useful to support users' visual perception. Thus, it is important to display images in a way, that they do not distract the user from the primary task or action on the screen. Given the analysis of the previously described tools, a summary of the best practices for an MBD interface can be found in Table 5.1.

5.2. Requirements

The identified requirements for a model-based diagnosis system are presented in Section 5.1. In order to develop an application for general domains, the following three areas have been identified:

(1) Control area: Users are able to interact with the system and perform defined operations.

(2) Diagnosis area: Once an error is detected, the user can start the diagnosis system and identify the fault.

(3) Report area: In this area, users can document identified faults and send their findings to a higher instance for report or repair purposes.

5.3. Scenario

The example from Section 3.2.1 was adapted to create an understandable and practical scenario for an abductive model-based diagnosis application. Currently, *Smart Home* is a popular topic in consumer electronics, therefore the example was adjusted to behave accordingly. The scenario consists of two bulbs which can be switched on or off by the control area of the smartphone application. Additionally, the light of the bulbs can be set either to red, green, or blue. The *Base* is responsible for the communication between the bulbs and the control-device. In order to work properly, the *Router* needs to provide a

Layout	Smartphones are suitable for diagnosis applications.
	Desktop or native mobile applications limit the group of users, therefore consider web-based applications.
	The layout should be flat, clear, and consistent.
	Use responsive design to take full advantage of available screen space.
	Provide a search bar if listboxes contain a high number of items.
	All functions need to be self explanatory in such a way, that no manual/help function is required.
	Use images to support the user's visual perception, but do not distract from the main action on the screen.
Design	Ensure a consistent and professional design.
	Use standard behavior of GUI elements.
	Use groupboxes or other elements to structure items that belong together.
	Use recognizable icons or provide additional text.
Typography	Provide content in a suitable font size.
	Use high contrast font colors.
	Respect potential color blindness of users.

Table 5.1.: Best practices for a diagnosis interface

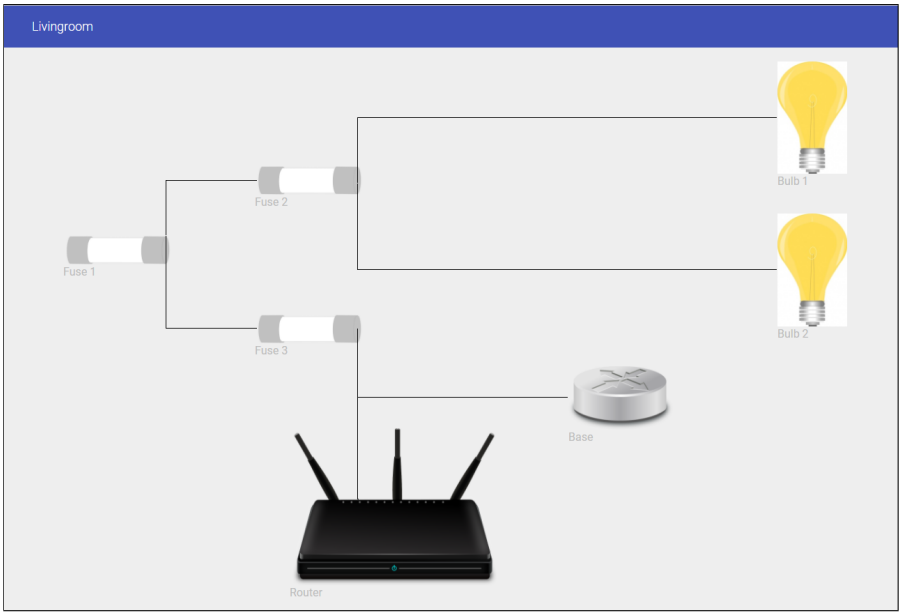
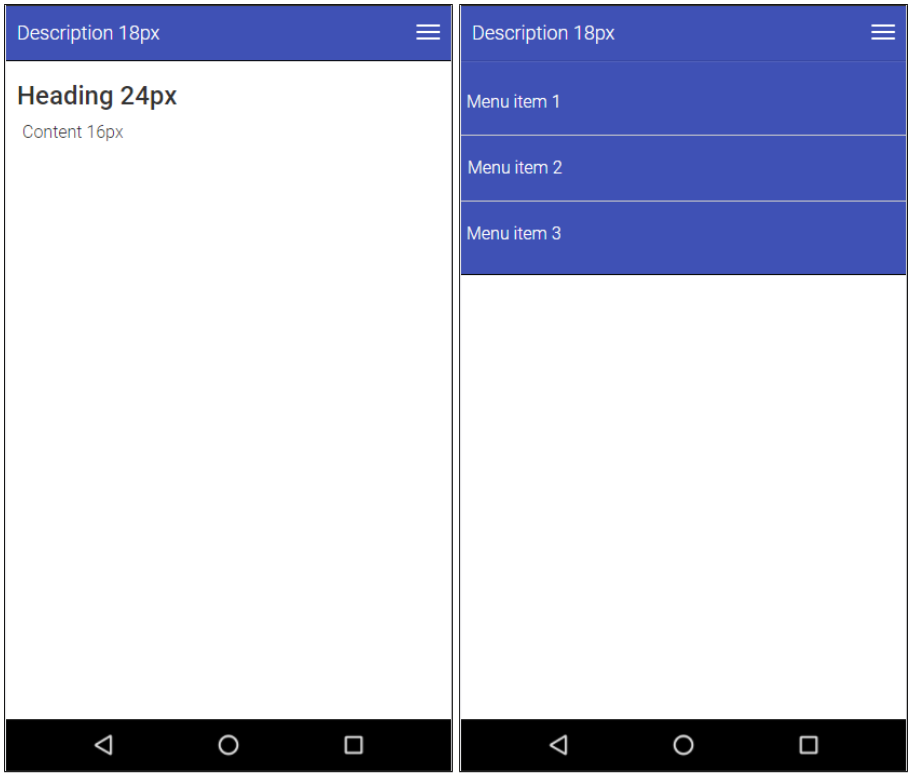


Figure 5.2.: Example for the system

connection to the internet. In case of a fuse breakdown, the bulbs might no longer be controllable. Figure 5.2 shows the modified bulb circuit. In order to test the diagnosis system, three faults are injected into the system which have to be identified by the user:

- Communication issue between base and user device
- Breakdown of a fuse
- Breakdown of internal components of the base

This scenario is used to develop a general application for model-based diagnosis, which should also be adaptable for other application areas. The advantage of this example is that users are aware of the *Smart Home* domain, but do not necessarily have used such devices.



(a) Layout of the content container (b) Layout of the menu

Figure 5.3.: Basic layout

5.4. Basic Layout

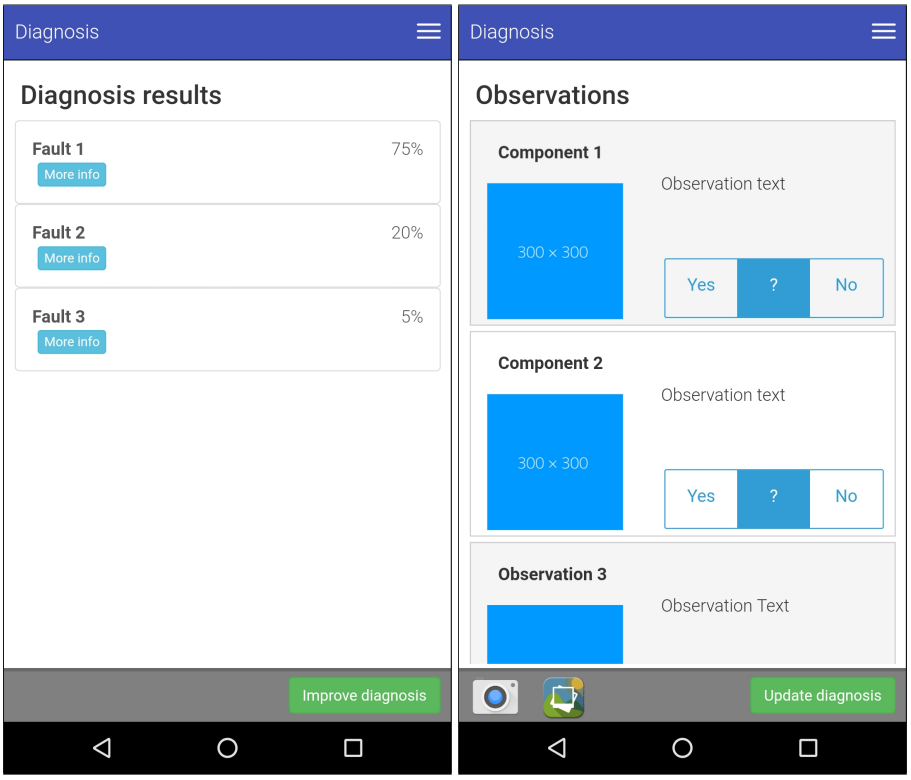
The general layout (depicted in Figure 5.3) consists of a colored header with a menu button on the right side (see Figure 5.3b). The font size of the active screen description is 18px. Each menu item has a total height of 60px and a font size of 16px to ensure accurate navigation even with broader fingers or gloves. Figure 5.3a shows the content container with a padding of 10px on each side and displays all other UI elements. It is worth noting, that the smallest font size throughout the application is 16px in order to ensure a good readability even on small devices.

5.5. Design

Regarding the requirements from Section 5.1 and 5.2, it was decided to design a web-application for mobile devices using the mobile-first design approach [16]. In order to follow this design principle, the layout for the smallest supported screen size is created first. Afterwards, the layout is scaled to larger dimensions. The advantage of this design method is that the application is usable on a variety of different devices. Compared to the *JAVA* applications presented in Chapter 3, web-application are not only platform-independent (since only a web-browser is required), using the mobile-first responsive design approach device-independency is accomplished. In order to ensure the responsiveness of the application, the framework *Bootstrap* was used. It provides a lot of predefined style classes, UI elements, and *Java Script* functions which ensures a consistent "look-and-feel" throughout all application screens. Figure 5.4a depicts the design of the diagnosis area. The menu-bar design at the top of the screen consists of a dark purple background with white color and menu icon. The dark gray colored control-bar at the bottom contains a button to operate with the diagnosis engine. Once the *Improve diagnosis* button is pressed, the application toggles to the observation area, where the icons for camera and gallery are added to the control-bar (see Figure 5.4b). Depending on the domain, additional features can be added. In order be consistent with common applications, the content area is kept white with back font color.

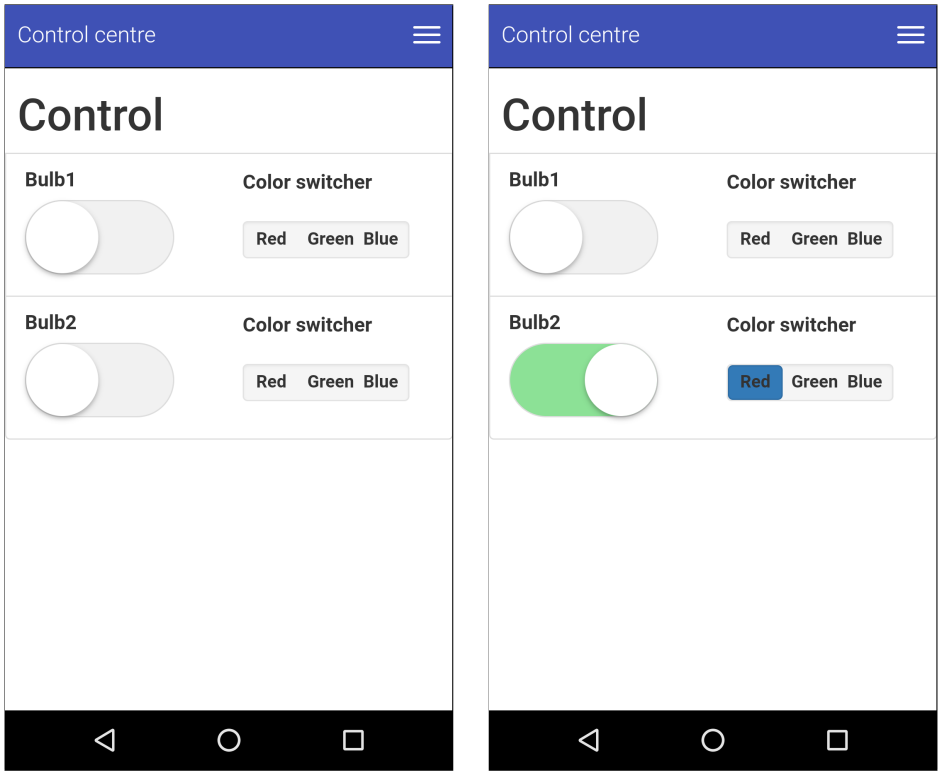
5.5.1. Control Area

Figure 5.5 depicts the screen for the control area which contains all options to interact with the system described in Section 5.3. Each bulb can be switched on or off by pressing the toggle-button on the left side. Once a bulb is activated, the color option is enabled and the user can choose between the colors red, green, and blue.
In certain domains, a control area might be not needed. Instead, performance parameters, system conditions, or other values can be displayed.



(a) Design of the diagnosis area (b) Design of the observations area

Figure 5.4.: Design of the application interface



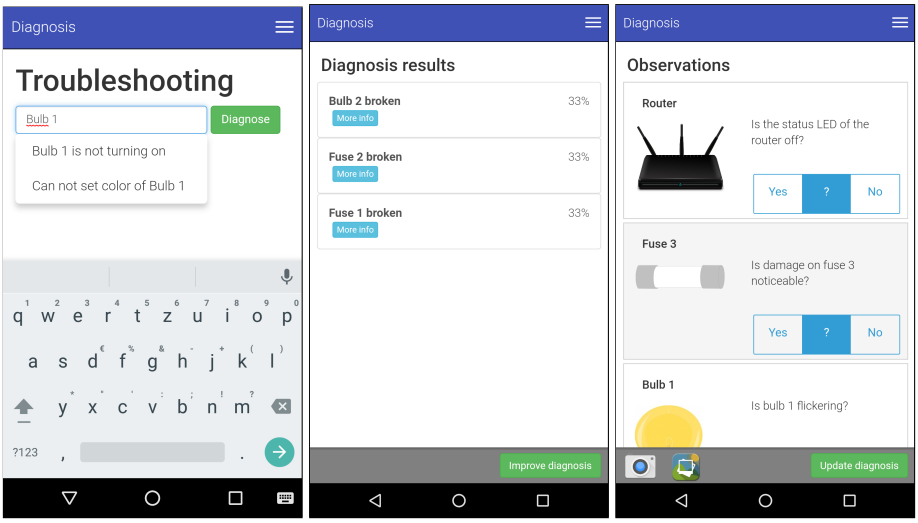
(a) Interface of control centre (b) Bulb control demonstration

Figure 5.5.: Screen design of the control area

5.5.2. Diagnosis Area

This area allows the user to identify faults in the system. The user input is sent to an ATMS (presented in Section 3.2.2) which returns a set of sets explaining the inserted observations. The data sent and received from the diagnosis system is mapped against database values in order to convert human-readable text to machine code and vice versa. The diagnosis system has to be triggered manually by an initial observation. Once the user selects the diagnosis entry from the menu, he is asked to enter the faulty behavior in the input field (depicted in Figure 5.6a). In some systems, a large number of symptoms can be possible. Therefore, the overlay under the input field shows the filtered results based on the user input. Currently, only one symptom is supported to calculate the initial diagnosis. Once the user clicks on the *Diagnose* button, the diagnosis engine is triggered computing the initial results. Figure 5.6b shows the initial diagnosis results, where each list item represents one possible fault. The *More info* button opens a collapsible panel which contains information about the cause of the fault and the repair/replacement task. The likelihood for each fault is based on an initial value and increases or decreases whenever new observations are added to the system. In order to discriminate the possible faults, the *Update diagnosis* button is used to navigate to the *Observations* screen depicted in Figure 5.6c. The *Observations* screen contains a scrollable list where each list-item represents an observation not yet considered in this diagnosis problem. The left part of the list-item informs the user about the component, where the right part contains the observation to be made. The questions can either be approved (*Yes*), denied (*No*), or not answered (*?*). Once the *Update diagnosis* button is pressed, the system updates the likelihood of the diagnosis results with respect to the inserted observations. Further, the list of faults is rearranged and arrows indicate if the fault converges or not. The diagnosis can be improved several times until an acceptable certainty for a fault has been accomplished.

In some cases, users might prefer larger devices for the diagnosis process. The mobile-first design principle allows the interface to scale up to any display size. Since more screen space is available, additional content can be displayed. Figure 5.7 and 5.8 show screen mockups for a desktop diagnosis with a three-column layout where the menu is present on the left, the diagnosis results are displayed in the middle, and additional domain specific information is located



(a) Start diagnosis (b) Diagnosis results (c) Diagnosis refinement

Figure 5.6.: Screen design of the diagnosis area

in the right column. On desktop system, the mouse is used as general input device. Therefore, UI elements optimized for touch input can appear smaller to save screen space and display more important information. The items in the fault list are displayed inline and a default height of 35px is used. Further, the scalable interface ensures an easy integration in existing applications. Since the interface only contains standard UI elements, the visual appearance can be easily modified to conform with company guidelines.

5.5.3. Report Area

After completing the diagnosis process, users should be able to document their findings. Depending on the domain, users might want to document the performed work process to their company or schedule a repair/replacement request in order to solve the identified problem. The general design of the report area is shown in Figure 5.9a. Clicking the *Add diagnosed fault* button, users are asked to insert the diagnosed fault by selecting the list item accordingly. In some cases, the fault might not be diagnosed within the system,

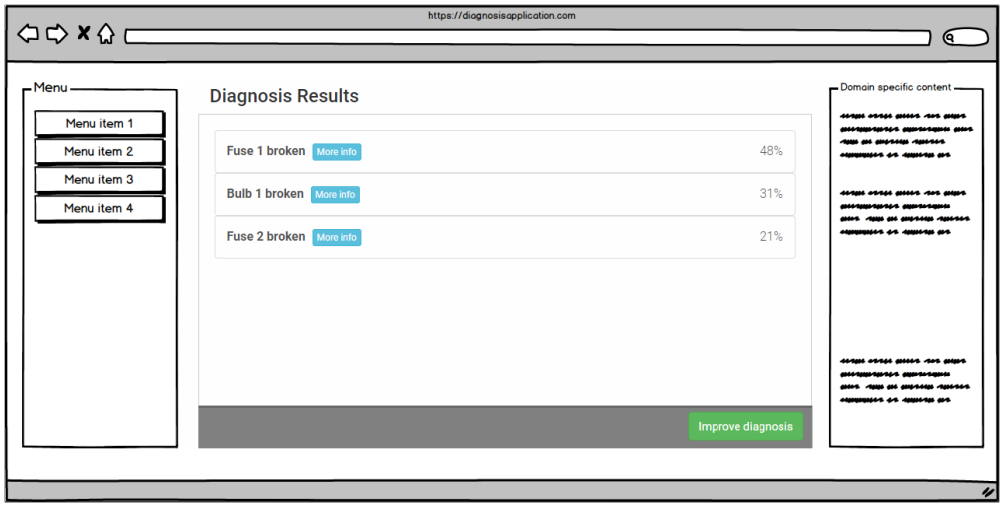


Figure 5.7.: Mockup of a desktop diagnosis application; Results screen

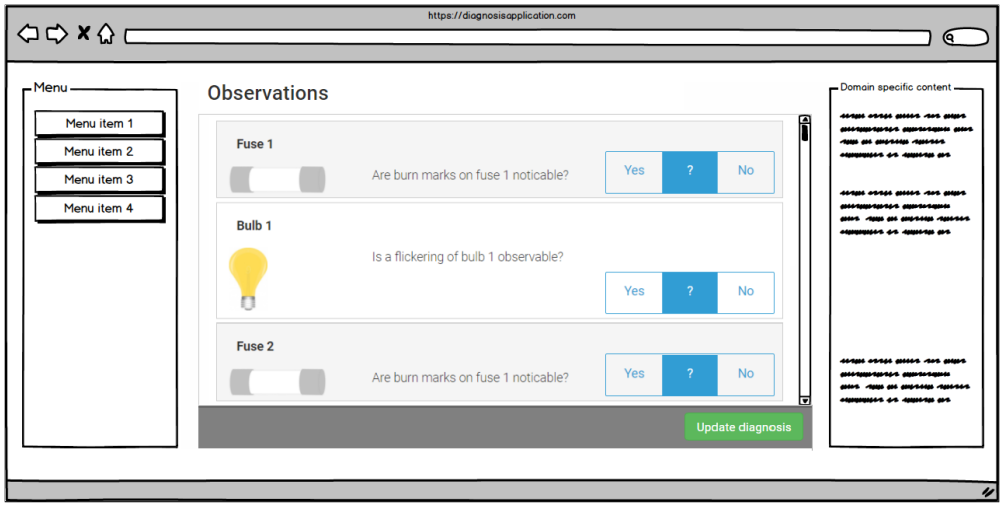
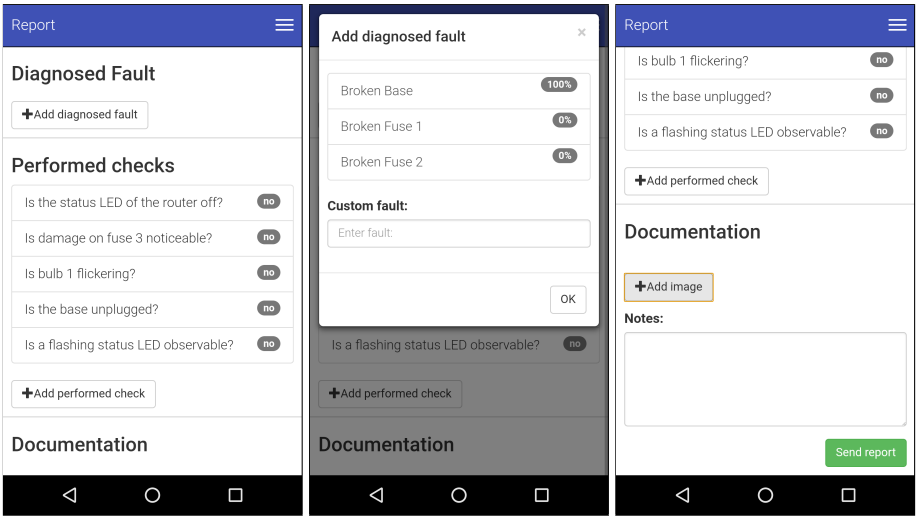


Figure 5.8.: Mockup of a desktop diagnosis application; Diagnosis refinement screen



(a) Report page (b) Fault input (c) Documentation

Figure 5.9.: Screen design of the report area

therefore users are able to report a custom fault using the input field at the bottom of the dialog depicted in Figure 5.9b. The system automatically logs the performed observations and displays them in the *Performed checks* section. If other observations are required to identify the fault, users have the possibility to insert them using the *Add performed check* button. This information can be used to update the underlying logical model to achieve more accurate results. In order to solve the identified problem, serial numbers of the faulty components or modules may have to be reported. Since smartphones provide high resolutions cameras, users can simply take pictures by clicking on the *Add image* button (see Figure 5.9c) which are then automatically inserted into the report. Additionally, the report area provides a text field, where users can leave notes or comments about their findings. The *Send report* button stores the entered information in a database for further processing.

5.6. Evaluation

To evaluate the usability of the system described in the previous sections, a *Thinking aloud* test was performed. The purpose of the test was to identify usability issues in the layout and design. One important criterion when conducting a thinking-aloud test is how the moderator interacts with the participants during the test session. Olmsted-Hawala et al.[46] state three types of thinking-aloud protocols:

- **Traditional protocol**: Remind the participant to keep talking after 15 seconds of silence.
- **Speech-communication protocol**: Verbal feedback in form of "um-hmm" and "un-hmm" is allowed as well as formulating a question with the last word the participant said (e.g. Participant: "This was weird..." Moderator after a pause: "Weird?").
- **Coaching protocol**: Active intervention during the test session is possible, e.g. ask direct questions or assist when the participant struggles.

The conducted thinking-aloud test followed the *Coaching* protocol. This protocol was chosen in order to get the most information why participants are struggling in certain areas of the application. Barnum [3] lists the requirements for an unbiased and effective moderator. The following questions have been used to assist users to keep talking when they are stuck during the task (Questions adopted from [3]):

1. "What are you trying to do?"
2. "What would you expect here?"
3. "What do you think should have happened?"
4. "Can you tell me what you are thinking right now?"
5. "What are you looking for?"

6. "Can you read the task again?"

5.6.1. Participants

Since the general interface needs to be usable for the vast majority of users, three user groups were defined [37]:

1. **Novice users**: Users that did not grow up with digital technology and use smartphones and computers only occasionally.

2. **Casual users**: Users that grew up with digital technology and use smartphones and computers on a regular basis.

3. **Expert users**: Users that grew up with digital technology and have a technical education/background.

In order to get valid results, three to five users per user group need to be tested [42]. Therefore, five novice users, eight casual users, and seven expert users participated the usability test study.

The users were selected by convenience sampling, where seven females and 13 males participated in the usability study. The gender distribution is shown in Figure 5.10. Unfortunately, no female expert user was available during the period of the usability evaluation.

The distribution of the participants' age is depicted in Figure 5.11. The age of the casual and expert users lies between 20 and 39. The participants from the group of novice users were between 53 and 65 years old. Figure 5.12 illustrates the educational background of the users. Most novice users stated that they completed an apprenticeship while the group of casual users stated a Master's degree and A-levels as their highest completed education. The expert users quoted to apply to a Bachelor's / Master's degree program in computer science or software engineering.

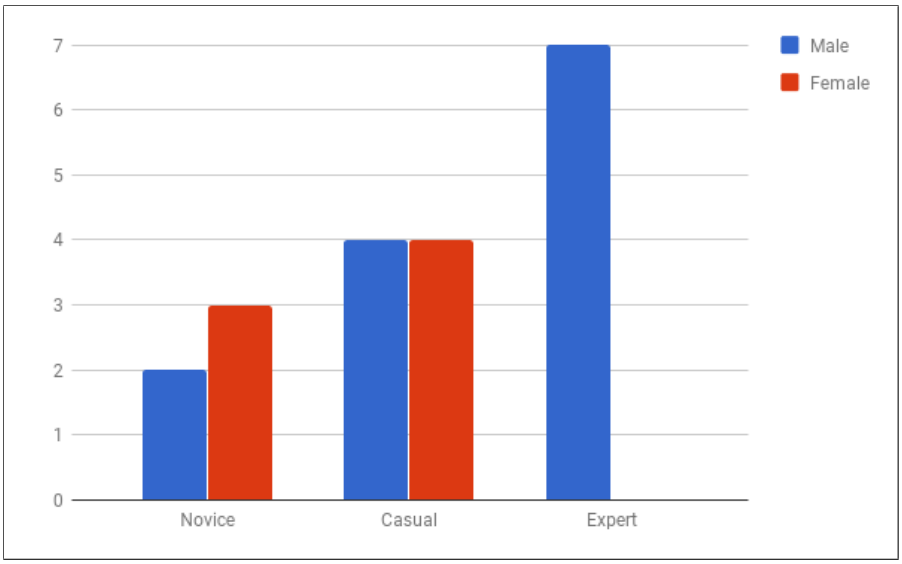


Figure 5.10.: Gender distribution of the participants

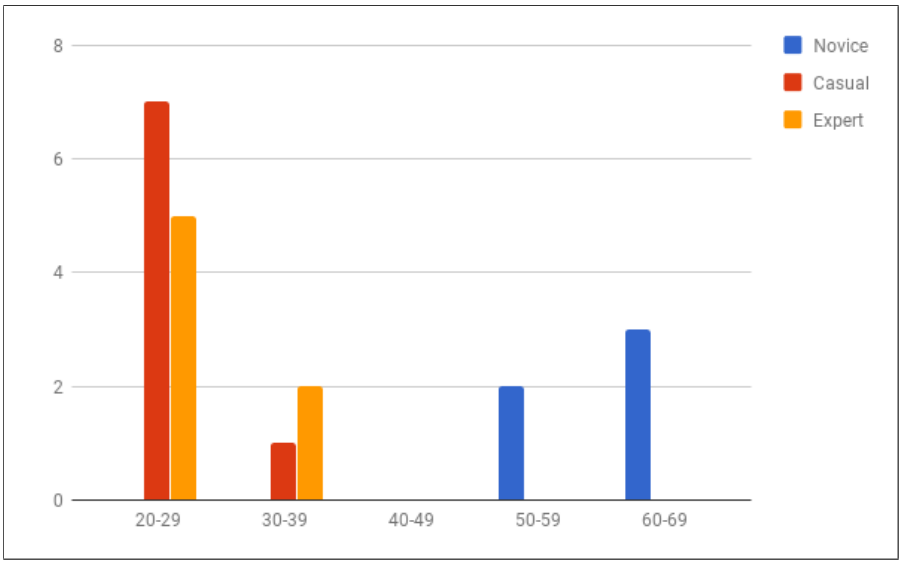


Figure 5.11.: Distribution of the participants' age

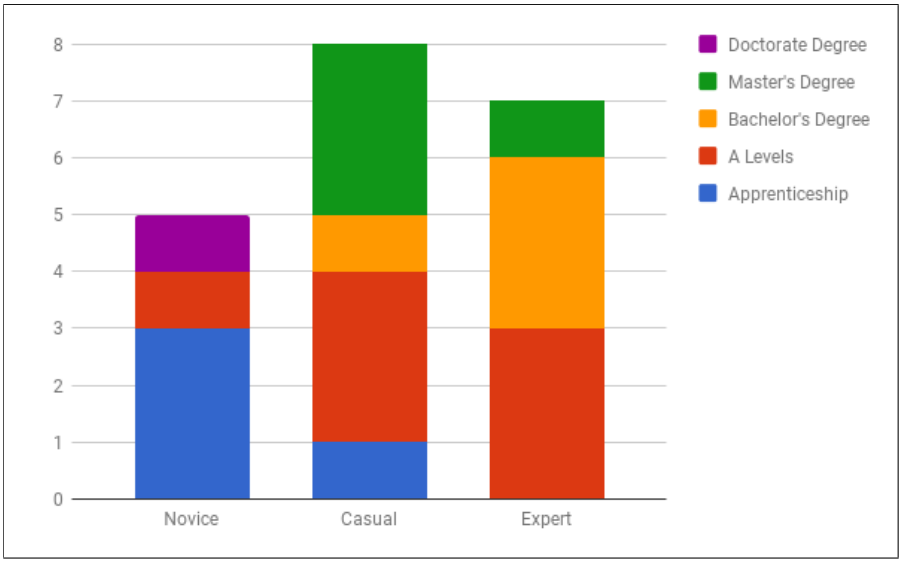


Figure 5.12.: Educational background of the participants

5.6.2. Usability Test Setup

The usability test was performed in a separate quiet room following the *Single room single camera* setup described in Section 4.2. Figure 5.13 illustrates the setup used for the usability study. The application was installed on an Apache webserver and accessed via a 5.5 inch smartphone. The scenario was presented on a 15 inch laptop. Users were able to interact with the smartphone and see the consequences of their actions on the laptop screen. To record all user interactions, a screen recording software was used on the laptop and the smartphone. The entire test session was recorded from a distance with a camera attached to a tripod. Additionally, the users' facial expressions were recored using the laptop's front facing camera.

The test session was structured into five parts:

1. **Background questionnaire:** In order to assign the test user to an user group, each user was asked about education, computer-, and smartphone experience. The background questionnaire (adapted from [1]) as

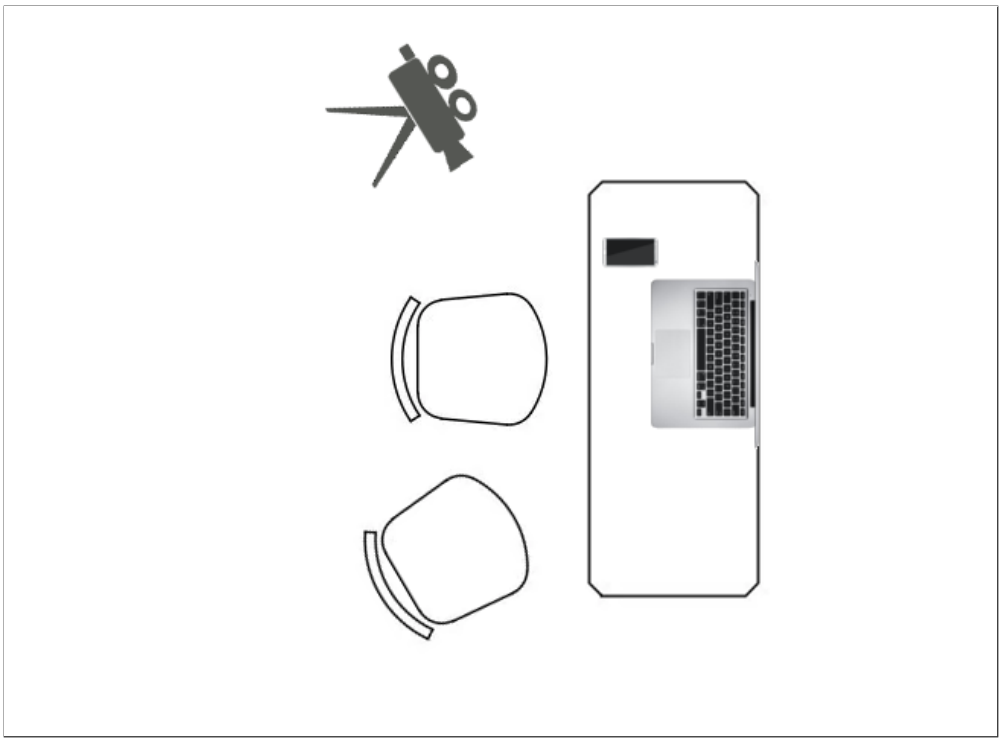


Figure 5.13.: Illustration of the used single room, single camera setup

well as the results can be found in Appendix A.1.

2. **Explanation of the scenario and test process:** Each user got a brief introduction to the *Smart home* scenario, where the two bulbs can be controlled by the smartphone application. It was mentioned that the application includes a system for fault identification, in case an unexpected behavior can be noticed. Finally, the test methodology was explained and the users were asked to verbalize their thoughts while performing the tasks. Afterwards, questions regarding the scenario or test method were answered.

3. **Test session:** The *Thinking aloud* test consisted of five challenges with increasing complexity. Since challenge 1, 2, and 3 consisted of two tasks, each user had to perform eight tasks in total. A list of the tasks can be found in Appendix A.2.

4. **Interview I:** After the usability test, the users were asked about their opinion on the application, especially what they most liked and disliked. If there were interesting interactions noticeable during the test, the users were confronted with their actions and the required steps as well as possible redesigns were discussed.

5. **Interview II:** Afterwards, the users were asked their opinions of two different interfaces presented on a paper panel. The purpose of this comparison was to find out which interface the users would prefer and why.
6. **Quantitative evaluation method:** Finally, the users were asked to fill out an SUS questionnaire. This quantitative evaluation method gives comparable results and provides a score identifying the average usability of a system.

The recorded video material from each test session was synchronized and prepared for the analysis using the scheme depicted in Figure 5.14. The screen capture of the smartphone is located in full height on the left side of the analysis schema with the video of the laptop's front facing camera right next to it. Thus, an efficient analysis of the user's emotions after an activity can be performed. The recording of the laptop screen is located at the top

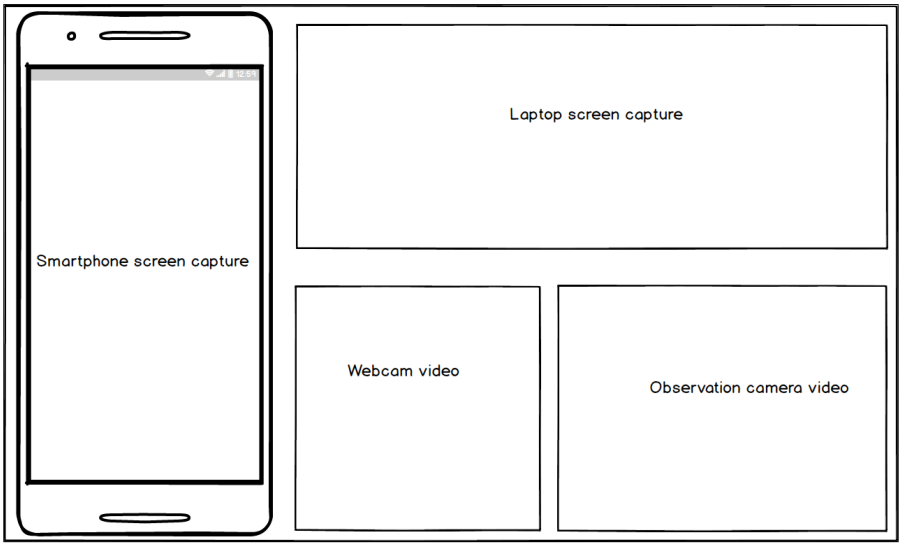


Figure 5.14.: Video scheme for the usability test analysis

while the video of the observation camera is placed in the bottom right. This arrangement of the different video sources helps to analyze the performed actions while taking the user's emotions and gestures into account.

5.6.3. Tasks

The following tasks were used for the usability study. In order to avoid language barrier, the tasks were translated to German (see Appendix A.2). Each task was presented to the participants on a separate sheet of paper.

Task 0: Introduction task

Challenge:
Run the application and get an overview of the available functions.
Goal:
The users get to know the application, exploring the different menu items, screens, and should feel comfortable using the software.

Task 1a: Automatically triggered diagnosis

Challenge:
Turn on Bulb 1. If you recognize any problems, please run the diagnosis system and diagnose the observed error.
Goal:
After pressing the button to activate the bulb, the user should recognize the error message at the bottom of the screen and tap on the button *Start diagnosis*.

Task 1b

Challenge:
You want to fix the error. Find informations about the cause of the problem and repair instructions.
Goal:
Users should identify the list item as diagnosis result and tap the *More info* button to receive repair / replacement instructions.

Task 2a

Challenge:
Turn on Bulb 2. If you recognize any problems, please run the diagnosis system and diagnose the observed error.
Goal:
Users should recognize that the bulb is not turning on. In order to start the diagnosis engine, the user needs to tap on the menu icon and press the menu-item *Diagnosis*. The appearing UI contains an input field, where the user is asked to enter the observed symptom, i.e. "Cannot turn on Bulb 2". The task is completed once the user confirms the input by pressing the *Diagnose* button next to the input field.

Task 2b

Challenge:
Refine the diagnosis results until you receive an acceptable result.
Goal:
The user should recognize the *Improve Diagnosis* button at the bottom of the screen. Once the button is tapped, the observation area appears where the user is asked to inspect three components and confirm or deny the corresponding questions. The task is completed once the user answered the questions and pressed the *Update Diagnosis* button.

Task 3a

Challenge:
Turn on Bulb 1 and change the color. If you recognize any problems, please run the diagnosis system and diagnose the observed error.
Goal:
Users should recognize that the color of the bulb does not change. In order to start the diagnosis engine, the user needs to tap on the menu icon and press on the menu-item *Diagnosis*. The appearing UI contains an input field, where the user is asked to enter the observed symptom, i.e. "Can not change color of Bulb 1". The task is completed once the user confirms the input by pressing the *Diagnose* button next to the input field.

Task 3a

Challenge:
Refine the diagnosis results until you receive an acceptable result.
Goal:
Once the users enters the first three observations, three possible results remain. The user should recognize that the diagnosis results can be further refined by pressing the *Improve Diagnosis* button. The task is completed when the user enters three more observations and a single fault remains.

Task 4

Challenge:
In order to get a replacement for the faulty device, send a report to the manufacturer of the bulb system. Add the diagnosed fault into the report. Insert an picture of the base to help the service center identifying the model of the device.
Goal:
The users need to tap on the menu-icon and select the menu-item *Report*, which opens the report area. There, users need to press the *Add diagnosed fault* and select the identified fault. In order to insert an image of the defect component, users need to scroll down to the bottom of the screen and press the *Insert image* button. Once a picture has been taken, it is automatically inserted to the report. The task is competed, if the users confirm their inputs by tapping on the *Send report* button.

5.7. Results

The analysis of the recorded test sessions revealed several usability issues. The issues identified can be classified as layout, design, and workflow issues as well as missing features. In order to determine the severity of usability issues, Nielsen [35] recommends the following scale:

> **5 - Critical usability problem**: An issues classified with this rating needs to be resolved before the product can be released.
>
> **4 - Major usability problem**: Finding and implementing a solution for this issue has high priority.
>
> **3 - Minor usability problem**: Fixing this type of issue has low priority.
>
> **2 - Cosmetic problem**: Issues of this type do not need to be fixed unless extra time for this project is available.
>
> **1 - No problem**: This issue can be classified as a non-usability problem.

A complete list of the identified issues is depicted in Table 5.2. The table shows a description, the classification, the priority, and the number of users affected by this issue, where novice users (NU), casual users (CU), and expert users (EU) as well as the total number of affected users are listed.

5.7.1. Critical Issues

Three of the issues identified are caused by missing or inefficiently implemented features.

Missing input validation in troubleshooting field: Figure 5.15a shows the input field where the user input validation is missing. Users clicked the *Diagnosis* button without entering any information. Therefore, a blank diagnosis result was presented which caused confusion and users were not able to recover from that mistake. Users claimed during the post test interview, that they thought they only needed this field to enter additional information. This issues can be fixed by implementing a proper input validation, informing the user to click on the field and enter the observed behavior.

Back button functionality not implemented properly: During the test, users tried navigate backwards or correct their mistakes by using the *Back* button on the device. When users tried to navigate from the observations to the diagnosis results screen, the system failed to display the expected content. Since this is a standard functionality of all Android and Windows phone devices, a proper implementation of the *Back* button functionality has high priority.

Swipe gesture not implemented: Another missing feature occurred in the control area of the application. Users tried to switch on the bulb using a left to right swipe gesture on the according UI element shown in Figure 5.15b. Since only a click event was recognized by the system, any movement over the UI element would not trigger the event.

Performed inspections cause confusion: Users did not remember that the listed *Performed checks* (depicted in Figure 5.15c) were answered in the previous step. Since the automatically collected data is important information for documentation purposes but is not necessarily shown to the user at this location of the screen, the *Performed checks* can be hidden and

only displayed if the user requests it. This redesign will keep the user's attention on the available functions, since less information is displayed.

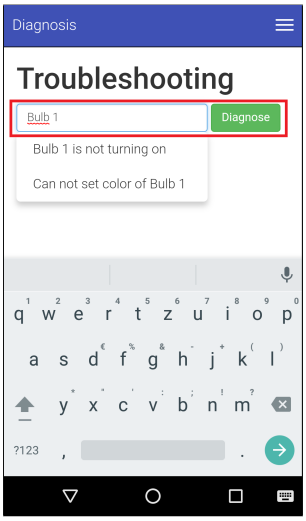
5.7.2. Major Issues

"Add diagnosed fault" button not recognizable: Although the *Add diagnosed fault* button depicted in Figure 5.16a is in a very prominent spot, users had problems recognizing it. The analysis of the video material revealed, that the list of performed checks distracted the users from finding this important function (see Figure 5.15c). Removing the list of *Performed Checks* from the *Report area* gives this important functionality more visibility.

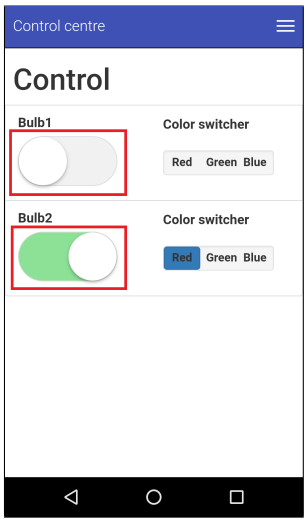
"Add diagnosed fault" procedure unclear: It could be observed, that the users had problems adding the diagnosed fault to the report. The evaluation of the recorded sessions showed that users thought the fault was already added to the report since it was present in the list (see Figure 5.16b). In order to fix this issue, a text label needs to be present where the user is asked to select the according fault from the list. Subsequently, the system needs to prevent the user from sending a report without including a fault.

"Improve diagnosis" button hard to find: Three casual as well as two expert users had problems recognizing the *Improve diagnosis* button in the control-menu at the bottom of the screen (depicted in Figure 5.16c) when they entered the *Diagnosis area* for the first time. This issue could be resolved by setting the background color of the control-bar to the same color as the menu-bar to visually indicate a similarity to these areas of the screen.

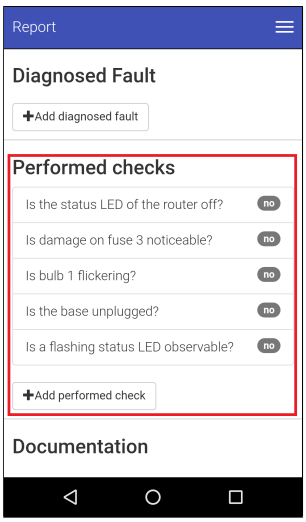
Wrong symptom for initial diagnosis chosen: Figure 5.16d depicts the drop-down menu, where four users selected the wrong symptom for the initial diagnosis without noticing. This issue can be solved by displaying a confirmation screen where users have the possibility to approve or correct their selection.



(a) Missing input validation

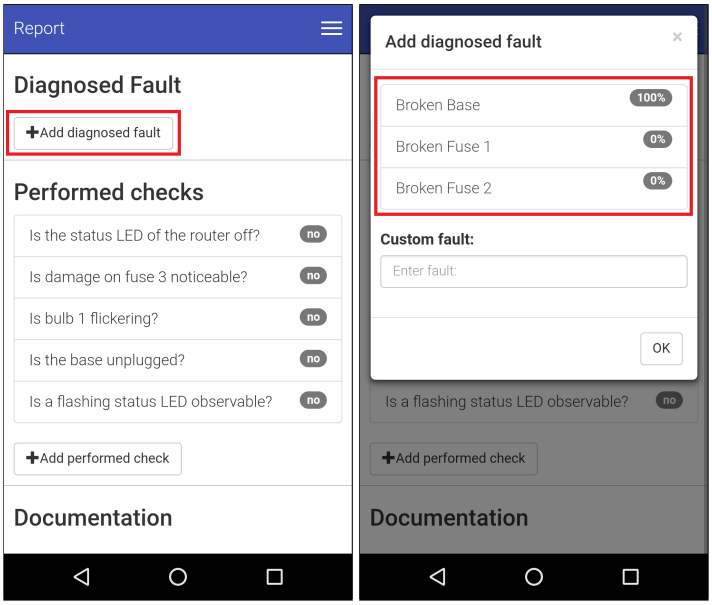


(b) Swipe gesture not implemented



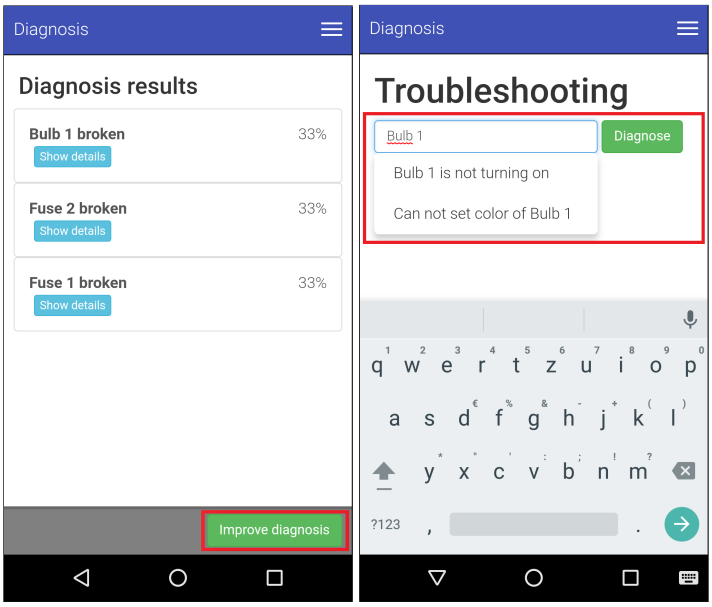
(c) Unnecessary information

Figure 5.15.: Identified critical usability issues



(a) Improve diagnosis button hard to find

(b) Add fault procedure unclear



(c) Improve diagnosis button hard to find

(d) Wrong symptom for diagnosis selected

Figure 5.16.: Identified usability issues with high priority

5.7.3. Minor Issues:

Three usability problems could be identified as minor issues. These problems did not affect many users and also require little effort to correct.

Negative question is confusing: The users were asked by the diagnosis system, if a certain status is not observable. Some users were confused whether to confirm or deny this question. Therefore, all questions should be formulated positively and clearly.

Scroll up issue after updating diagnosis: After updating the diagnosis, the system did not reset the scroll position of the list containing the diagnosis results, which caused confusion to two users. This can be fixed by automatically resetting the position of the list after the *Update diagnosis* button is pressed.

Trash can icon not recognized: One user misinterpreted the *Trash* icon. In order to fix this issue, icons need to be displayed in combination with a text label.

5.7.4. Issues Identified as Non-usability Problems:

"Add image" procedure unclear: Two novice users had problems inserting images into the report using the default android camera application. Since the users admitted not using their smartphone camera too often, the issues is caused by the lack of experience with such devices.

Wrong observation value inserted: Two users inserted the wrong observation to the system. This was caused by not reading the question carefully enough.

Menu icon not recognized: Although only standard icons were used, two novice users could not identify the *Menu* icon.

5.7.5. Interview I: Interface Comparison

After the usability test, users were asked on their opinion on two different prototype interfaces (see Figure 5.17 and 5.18) for a model-based diagnosis application. *Prototype I* used collapsible panels to represent the diagnosed faults (see Figure 5.17a). Clicking on a panel expands the observations area

Issue	Description	Type	Priority	NU	CU	EU	Total
1	Missing input validation in troubleshooting field	Feature	5	4/5	1/8	2/7	7/20
2	Back button functionality not implemented properly	Feature	5	3/5	2/8	2/7	7/20
3	Swipe gesture not implemented	Feature	5	1/5	3/8	3/7	7/20
4	Performed inspections cause confusion	Layout	5	3/5	2/8	2/7	7/20
5	"Add fault" button not recognizable	Design	4	1/5	3/8	2/7	6/20
6	"Add fault" procedure unclear	Workflow	4	2/5	2/8	2/7	6/20
7	"Improve diagnosis" button hard to find	Design	4	0/5	3/8	2/7	5/20
8	Wrong item for diagnosis chosen	Interaction	4	2/5	2/8	0/7	4/20
9	Negative question is confusing	Language	3	1/5	1/8	1/7	3/20
10	Scroll up issue after updating diagnosis	Workflow	3	2/5	0/8	0/7	2/20
12	Wrong observation value inserted	Workflow	3	1/5	1/8	0/7	2/20
14	Trash can icon not recognized	Design	3	1/5	0/8	0/7	1/20
13	Menu icon not recognized	Design	0	2/5	0/8	0/7	2/20
11	"Add image" procedure unclear	Design	0	2/5	0/8	0/7	2/20

Table 5.2.: Identified usability issues

Prototype	Novice user	Casual user	Expert user
I	3	4	3
II	2	4	4

Table 5.3.: Results of the comparison of the two prototype interfaces

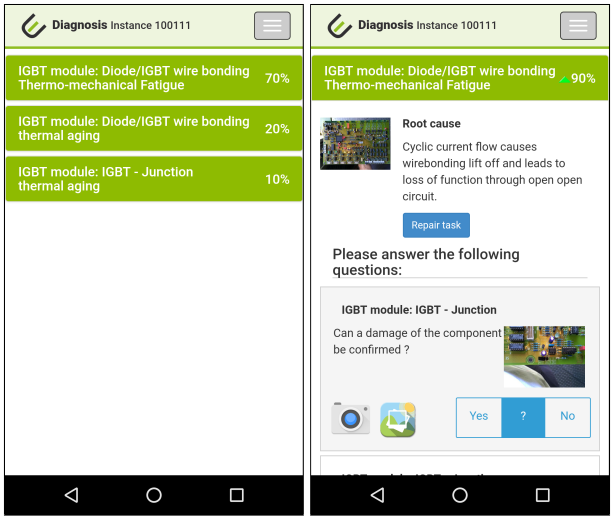
depicted in Figure 5.17b where also the root cause and the repair task are displayed. Figure 5.18a shows *Prototype II* which uses list items to represent faults and a separate area for observations (see Figure 5.18b). Table 5.3 shows the results of the comparison of the two interfaces. Since there is no difference between the users' preferred interface, it can be concluded, that both layouts are usable for a model-based diagnosis application.

5.7.6. Interview II: Post-Test Interview

Users were asked to state their opinions on the tested application. The users reported that they really enjoyed interacting with the software. The interface was said to be nice and clean with good looking colors and appropriate font size. Users pointed out that the overall speed of the application was great as well as the fluent interaction. The negative aspects of the application were the missing swipe gesture, the functionality of the *Back* button, and the procedure to add the identified fault to the report. Two users mentioned that the likelihood of the diagnosed faults were unclear and that they would prefer a simpler representation of possible errors.

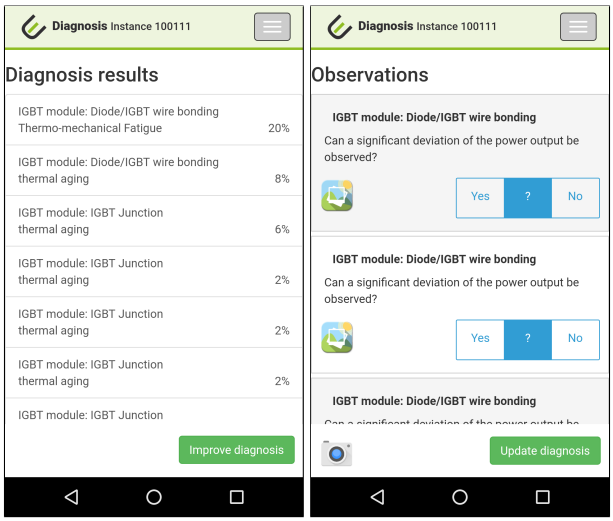
5.7.7. Evaluation of the System Usability Scale Questionnaire

Figure 5.19 depicts the average SUS rating of each user group visualized as semantic differential. Users with higher experience in digital technology tend to use such systems frequently, while users with less experience reported that they would need more practice and assistance in order to feel comfortable using such an application. Surprisingly, the novice users thought that the system will be easy to learn for other people, whereas experienced users gave a more conservative rating. Figure 5.20 visualizes the evaluation of the SUS ratings. Based on the average score of the novice users (75.5), the casual users



(a) Results area (b) Observations area

Figure 5.17.: Interface of Prototype I



(a) Results area (b) Observations area

Figure 5.18.: Interface of Prototype II

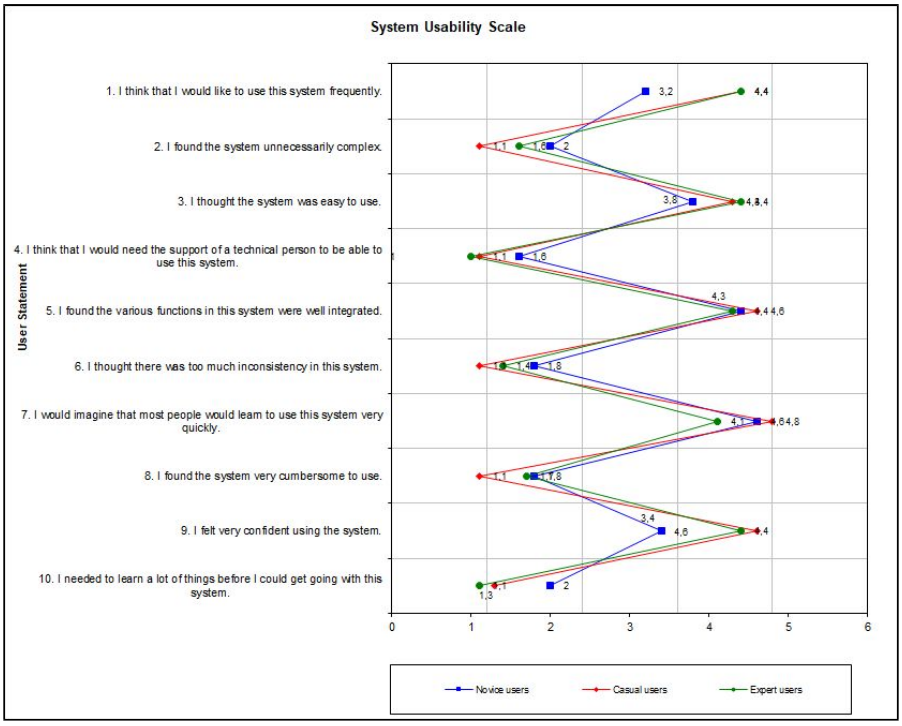


Figure 5.19.: Semantic differential of the SUS ratings of each user group

(93.4), and the expert users (86.1), the application achieved an overall mean SUS score of 86.4. This result confirms that the software is accepted and usable even for users with less experience in digital technologies.

5.7.8. Usability Metrics

In addition to the previously identified issues, quantitative data can be gathered from the video analysis. Nielsen [36] introduces the user success rate that provides a percentage value of how many tasks a user was able to complete correctly. Partially successful tasks were also taken into account. In order to compute a percentage value for the success rate, a completely successful task is rated with 1 point, a partially successful task with 0.5 points, and an unsuccessful task with 0 points. Figure 5.21 depicts the average user success

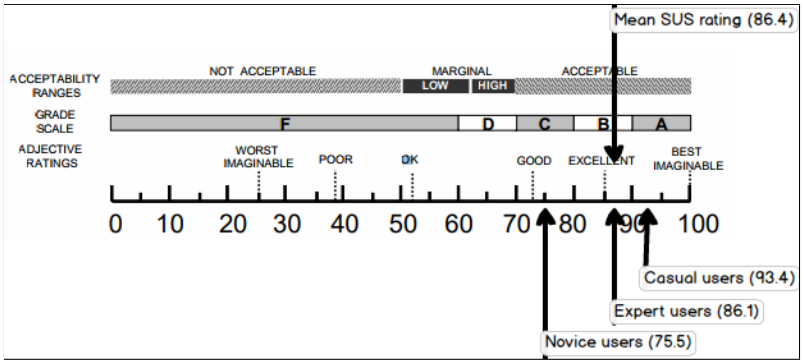


Figure 5.20.: SUS scores (adapted from [2])

rate and standard deviation (SD) of each user group. The novice users scored 53.7% (SD: 12.6), the casual users 94.6% (SD: 5.4), and the expert users 94.9 (SD: 5.8). The results show that users with more experience could also complete more tasks successfully. As Nielsen [36] states, these results can be used in a future usability test to determine if the suggested redesign proposals have had a positive impact on the overall usability.

The chart in Figure 5.22 shows the total number of mistakes the users made while performing each task. It can be concluded that users with less experience in digital technologies make far more mistakes compared to experienced users.

The average error rate (AER) for each group of users is depicted in Table 5.4. Sauro [56] shows that the average number of mistakes per task is 0.7. Calculating the average error rate for the tested interface leads to 1.4 for novice users, 0.4 for casual users, and 0.2 for expert users.

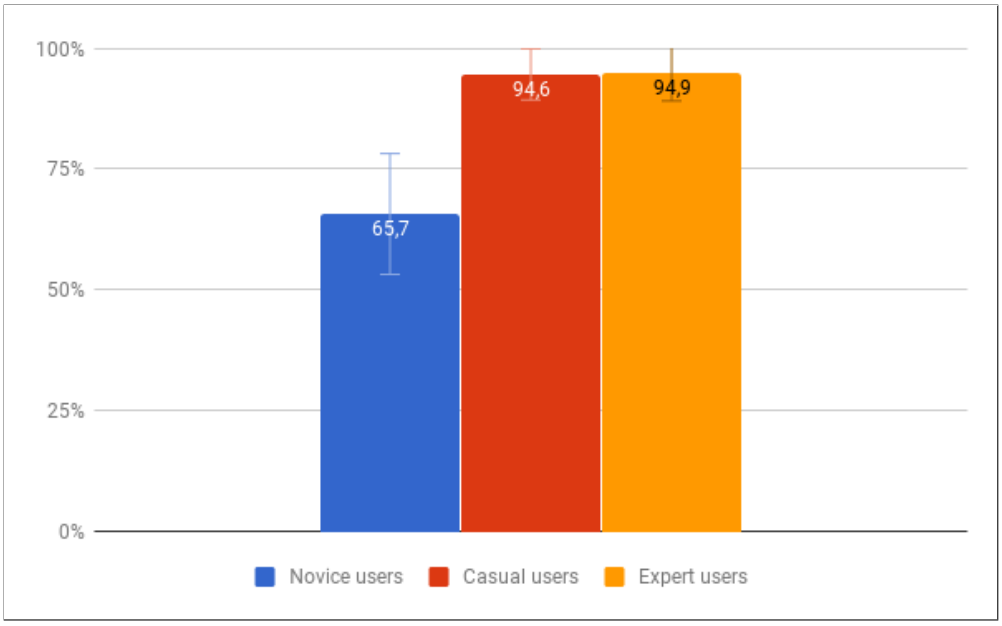


Figure 5.21.: User success rate

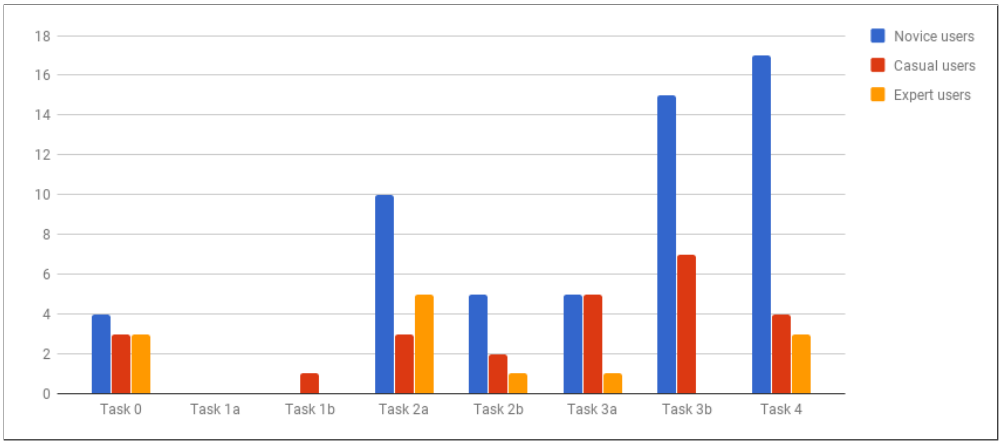


Figure 5.22.: Error rate

Task	AER novice users	AER casual users	AER expert users
Task 0	0.8	0.4	0.4
Task 1a	0.9	0.0	0.0
Task 1b	0.0	0.1	0.0
Task 2a	2.0	0.4	0.7
Task 2b	1.0	0.3	0.1
Task 3a	1.0	0.6	0.1
Task 4b	3.0	0.9	0.0
Task 4	3.4	0.5	0.4

Table 5.4.: Average error rate

6. Design of a Model-Based Application for Wind Turbine Fault Diagnosis

In order to bring model-based diagnosis into industrial applications, Graz University of Technology and Uptime Engineering GmbH founded the project EXPERT. Within this corporation, an application is developed to support the self maintenance activities of an local wind energy provider. Usually, maintenance and repair tasks are covered within a service contract and performed by an external company, i.e. the manufacturer of the wind turbines. The company performing the repair or maintenance activities schedule the appointments on their behalf without taking the current wind situation into account, which can have an negative impact on the efficiency of a wind turbine. Considering the highly competitive energy marked, reducing all type of operation costs is necessary to defend the current market position. Therefore, the energy provider decided to install their own service center in order to reduce the maintenance costs. The following optimization factors could be identified:

- To maximize the efficiency of a wind power plant, all maintenance activities are scheduled based on the wind forecast.
- Rather than replace expensive modules, faulty parts should be replaced and the reworked modules reused.
- Non-critical repairs should be performed at the next planned maintenance appointment.
- Service technicians should carry potentially required parts and tools with them to reduce travel time.

Currently, the troubleshooting of wind turbines is based on the alarm codes reported by the turbine. Since the reported alarm code does not provide information about the faulty component and does not necessarily have to be related to the cause of the problem, the troubleshooting is mainly based on the experience of the service technicians. Therefore, the EXPERT project focuses on the development of an intuitively usable MBD application which should reduce the service technicians' troubleshooting time and support their workflow.

This section describes the design and evaluation process for a model-based diagnosis application in the field of wind turbine maintenance. Based on the resulting prototype, an abductive model-based fault identification system will be implemented into an existing condition monitoring system for industrial wind turbines. The design of the interface considers the current workflow of the maintenance personnel and aims to enhance their performance. An iterative design process was applied and continuous feedback from several stakeholders was taken into account. After the final iteration, a clickable prototype was developed and evaluated using the *Thinking aloud* method in combination with a SUS questionnaire.

6.1. Design Process

The development of the interface followed an iterative design process [34]. In this process, several steps are repeated until an acceptable result has been achieved. The process is depicted in Figure 6.1. Based on an initial plan, the following steps are performed iteratively:

1. Definition of the requirements
2. Creating a design
3. Implementation of the functionality
4. Test of the implementation

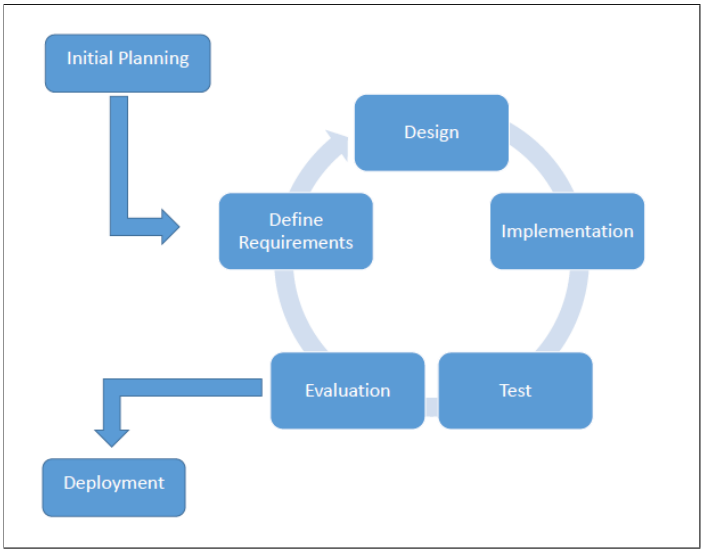


Figure 6.1.: Iterative design process (adapted from [17])

5. Evaluation of the resulting product

The process starts over, if the evaluation of the product shows that the goal has not been reached. As Nielsen [34] shows, developing a software product using an iterative design approach reduces the number of usability issues by 38% on average per iteration.

In order to derive the requirements for the interface, all stakeholders needed to be identified. During the design process, three stakeholders were involved:

1. The service technicians who will use the software for troubleshooting remotely at the service center and in the field.

2. The management department of the energy provider which plans to extend and optimize their maintenance activities.

3. The company Uptime Engineering GmbH who is developing condition monitoring software for wind turbines and wants to extend their portfo-

lio.

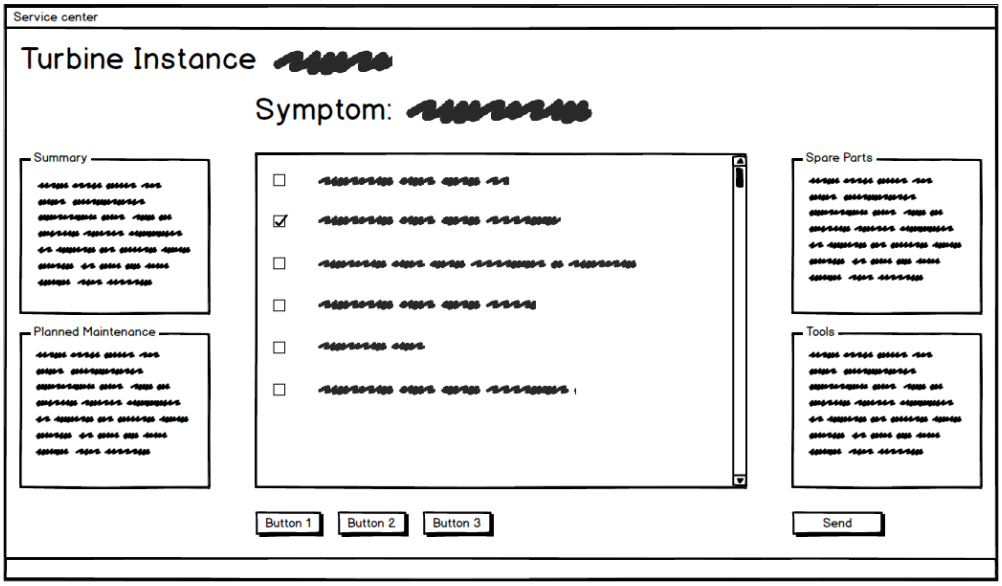
During the first meeting with Uptime Engineering, an initial set of requirements was developed. It could be concluded, that the main goal of the application is to not cause additional effort but support the service technicians' current workflow. This ensures usefulness, which is a key aspect in technology acceptance [9]. The general structure of the system was defined as well as the overall workflow and a small set of features. Based on the identified requirements, a low fidelity paper prototype was created (depicted in Figure 6.2). The layout was refined several times in order to prepare a paper mockup for the first meeting with the management department and the users. The feedback from the four participating service technicians revealed some usability and wording issues. The maintenance personnel provided detailed knowledge about the work process and the typical activities throughout the fault correction tasks. The information gained was used to adapt the mockup accordingly. Based on the paper mockup, the the focus on the next iteration was to develop a clickable prototype.

6.2. Requirements

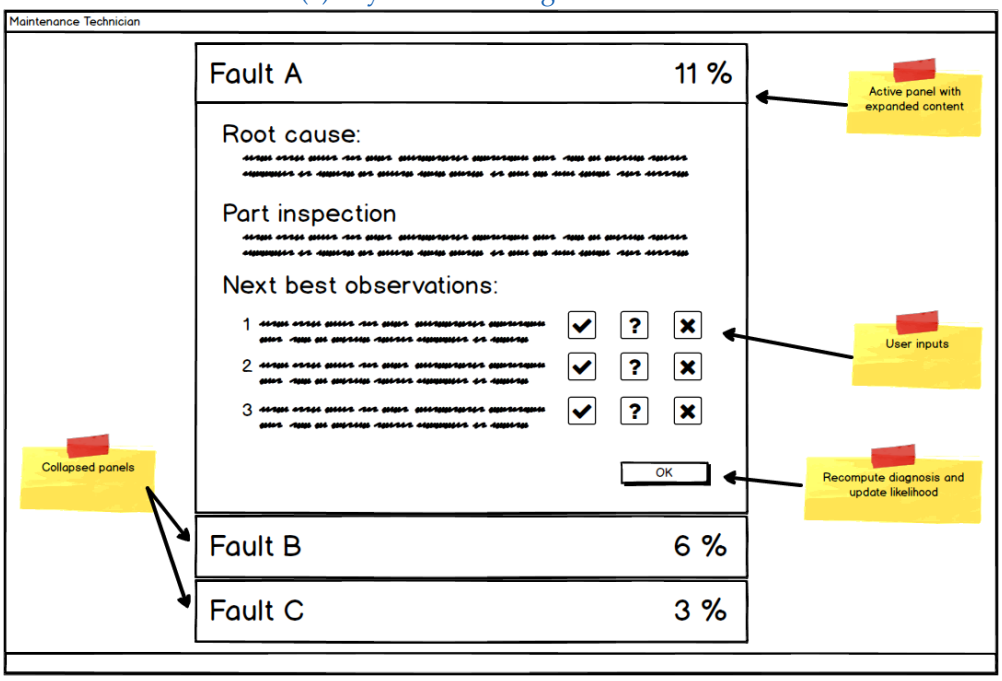
The requirements were identified during the various meetings with the three stakeholder groups. Each of them reported different requests which had to be analyzed and prioritized. Since the service technicians were the target user group, their suggestions were considered with special attention. The following requirements could be identified:

- The current fault detection of the maintenance technicians relies mainly on visual inspection, therefore images of components should be used to improve the detection speed.

- After the fault has been identified, the repair or replacement task is performed according to the instruction manual provided by the wind turbine manufacturer. This information needs to be included in the application in order to increase the usefulness of the software.



(a) Layout of the diagnosis results



(b) Layout of the diagnosis refinement

Figure 6.2.: Initial low fidelity mockups of the diagnosis application's interface

- Once the repair or replacement task is completed, the service technicians are required to create a report where performed activities are documented. Since documentation is an essential part of the current work process, the application need to provide an adequate solution.

- The working environment inside a wind turbine is uncomfortable, therefore a strictly defined path through the system with minimal user interaction is required.

- The management department of the energy provider wants to follow the trend of industrial digitalization. Further, safety and productivity are topics of the management's interest. In order to ensure minimal downtime, the application should support the technicians to prepare all required spare parts and tools before traveling to the turbine.

- Given the dangerous working environment inside the wind turbines, the system should also take the current safety processes into account.

- The requirements and specifications of the industrial partner Uptime Engineering GmbH needed to be satisfied. The interface of the application needs to be compatible with the company's design guidelines, which strictly define the representation of information and guarantee a consistent interface throughout the monitoring system.

- In order to extend and update the knowledge base of the diagnosis system, users should be able to report new faults which have not yet been considered.

- To support other domains for future projects, the interface should be extendable and adaptable.

6.3. Workflow and Interface Design

This section describes the workflow design of the diagnosis application. Figure 6.3 illustrates the identified work process. Wind turbines are equipped with

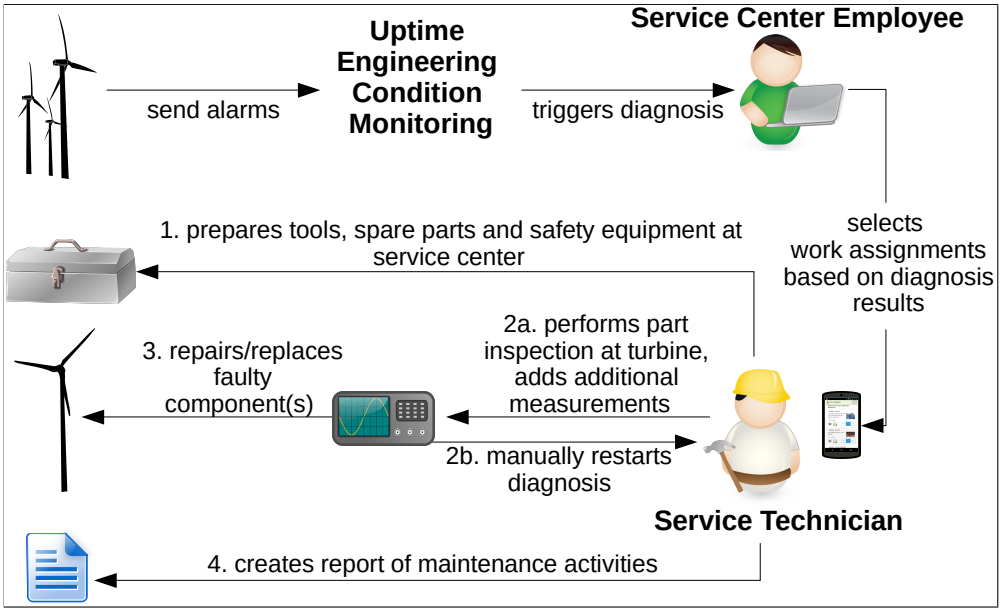


Figure 6.3.: Identified workflow of a wind turbine repair/replacement task [27]

sensors which measure critical values (e.g. rotation speed, produced power, ...). A basic monitoring system reports alarm codes if the sensor values differ from the expected range. Uptime Engineering's monitoring software processes the received data to allow more precise anomaly detection. The software identifies a symptom and triggers the diagnosis system by supplying the observations from the wind turbine and the system model to an MBD diagnosis engine. The initial diagnosis is displayed in the *Operations Center* depicted in Figure 6.4.

Each diagnosis result is represented as collapsible panel and contains the diagnosed faults. The main panel consists of the representation of the wind turbine as well as the detected symptom responsible for triggering the diagnosis. The diagnosed faults are represented as list items where the location of the fault, the faulty component, and the root cause are displayed. The right area of a list item is used to display the likelihood of the fault as well as a button for additional information. It is worth noting that this area is designed to contain additional UI elements for different purposes (e.g. links to

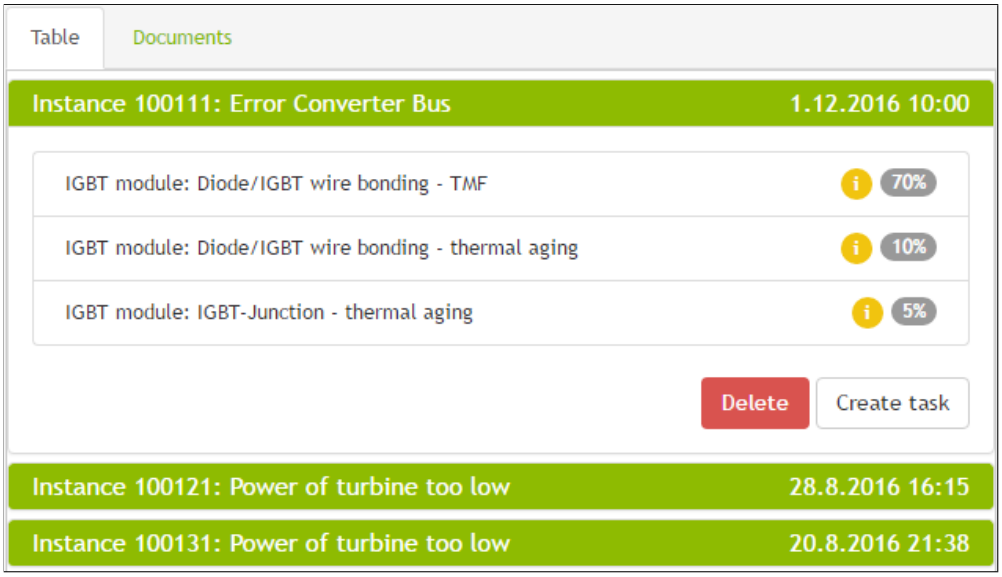
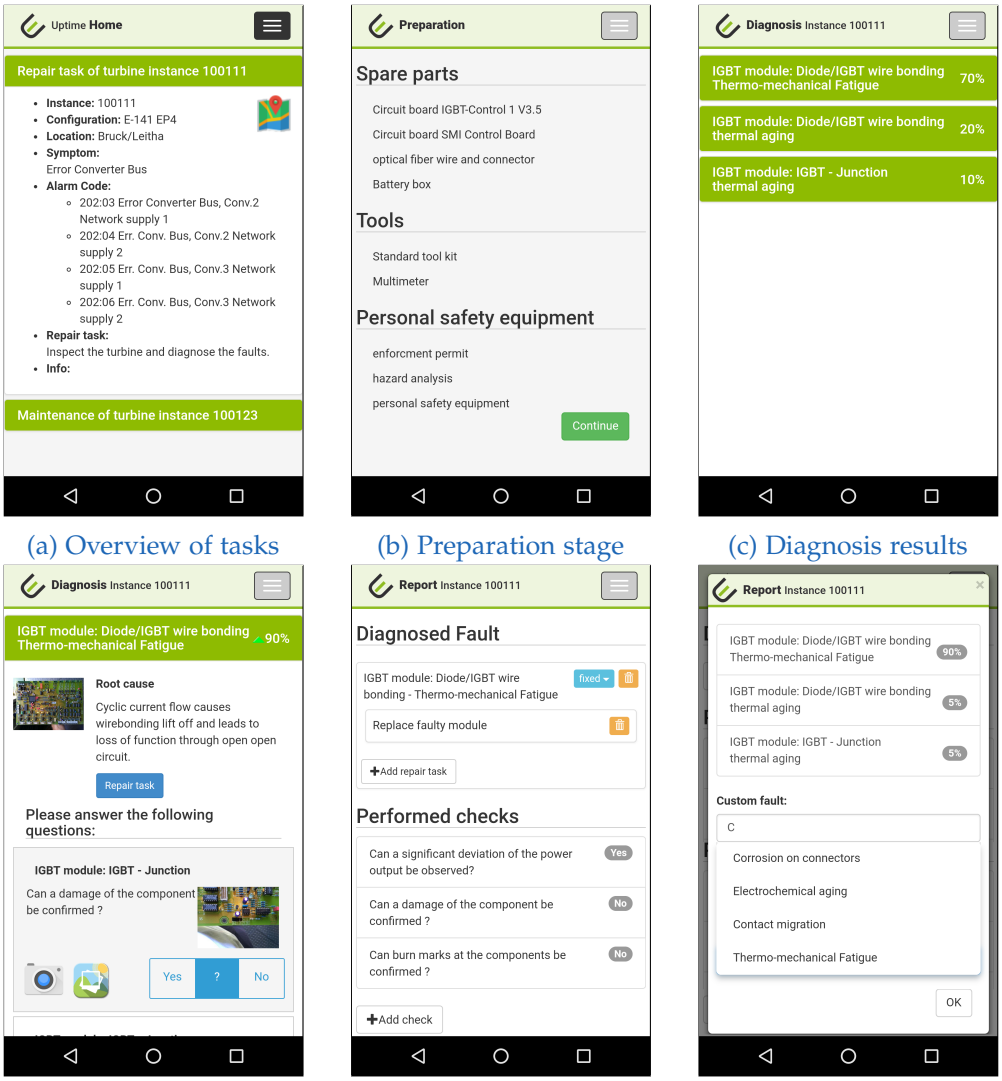


Figure 6.4.: Design of the Operations Center [27]

data-sheets, repair instructions, ...). Considering the current workflow of the maintenance personnel, a service technician in the *Operations Center* creates work assignments for one of his colleagues for one or more turbine instances. The work assignment contains possible faults for consideration, which could be identified in similar situations during the field work and information about previous issues with this specific turbine. The interface of the *Operations Center* is designed for desktop computers since the work assignments are distributed from the service technicians' headquarters.

The service technicians requested not to use laptops during the diagnosis process regarding the narrow environment inside a wind turbine. Therefore, it was concluded that the system in the field is most usable on a mobile device, since the technicians have to carry a smartphone for safety purposes anyway and they currently use it to take pictures of the identified faulty components for their work reports. In order to ensure a practicable interface, design guidelines [62, 20] and best practices for mobile applications [40] were taken into account. The prototype features a flat navigation since this is used in many other web applications for mobile devices. In contrast to a deep navigation



(a) Overview of tasks

(b) Preparation stage

(c) Diagnosis results

(d) Expanded panel for diagnosis refinement

(e) Report diagnosed fault / repair task

(f) Custom fault with autocomplete function

Figure 6.5.: Design of the mobile application interface [29]

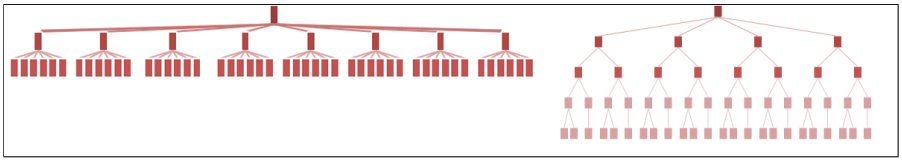


Figure 6.6.: Comparison between flat and deep website navigation hierarchy [64]

hierarchy, it is touch input friendly, ensures minimal user interactions, and gives every screen a defined role in the work process. Figure 6.6 depicts a comparison between flat and deep hierarchy for the same amount of information.

Once the user starts the application, the *Home* screen (depicted in Figure 6.5a) provides an overview of all service tasks assigned to the technician. The *Preparation* screen in Figure 6.5b provides a checklist to ensure that all spare parts and tools for the work assignment are taken. This is usually performed at the service center, where the stockroom is located. The screen also provides a verification of the personal safety equipment which has to be checked in order to continue the workflow. Afterwards, the service technician travels to a turbine for which a maintenance task has been assigned to him.
At the turbine, the *Diagnosis* screen (see Figure 6.5c) provides the possible faults which are ranked according to the calculated probabilities. The diagnosis results are represented as collapsible panels to provide an overview of all possible faults. Once the user clicks on the panel, the expanded area contains information about the root cause, the repair/replacement task, and a subset of the next best observations to discriminate the results (shown in Figure 6.5d). The application asks the user to answer questions which represent the ideal probing points. The displayed images next to the textual description supports the user in the identification of the correct component for the measurement. Each question can be confirmed (*Yes*), denied (*No*), or not answered (*?*). Each measurement can be documented separately using the camera icon at the bottom of the question area. The recorded pictures are included automatically in the maintenance report.
Given the new information from the observations, the diagnosis engine is restarted manually from the device in order to receive a newly computed diagnosis result. The probabilities and the arrangements of the faults are updated with arrows indicating whether a fault converges or not. This process

can be repeated several times until an acceptable assurance is reached or the fault could be detected during the measurements. Once the fault has been identified, the *Report* area can be used to inform the *Service center* about the confirmed diagnosis, performed repair/replacement task, consumed spare parts, and made observations.
Figure 6.5e depicts the *Report* area with the diagnosed fault and replacement task inserted. In certain scenarios, the fault might not be considered within the abductive model, therefore the user can insert a custom fault (see Figure 6.5f). Once the user clicks the *Send* button, the maintenance task report is automatically created out of the inserted data and sent to the *Operations Center* where it is stored and post-processed. The interface for the report management is not shown here, but it is already considered within Uptime Engineering's monitoring system.

6.4. Evaluation

The usability of the interface was evaluated using a *Thinking aloud* test and an SUS questionnaire. Again, the *Coaching* protocol (described in Section 5.6) was used. Before the usability test, a background questionnaire (adopted from [1]) was conducted to collect some data about the participants. The background questionnaire as well as the results can be found in Appendix B.1 This section describes the participants, setup of the usability test, and the presented tasks.

6.4.1. Participants

Five service technicians from the local energy provider participated in the usability test. The distribution of the participants' ages is depicted in Figure 6.7, where 26 was the youngest and 42 the oldest user.

The educational background of the service technicians is depicted in Figure 6.8. Three technicians completed electrician apprenticeships, one finished a secondary technical college, and one has a Master's degree in electrical engineering.

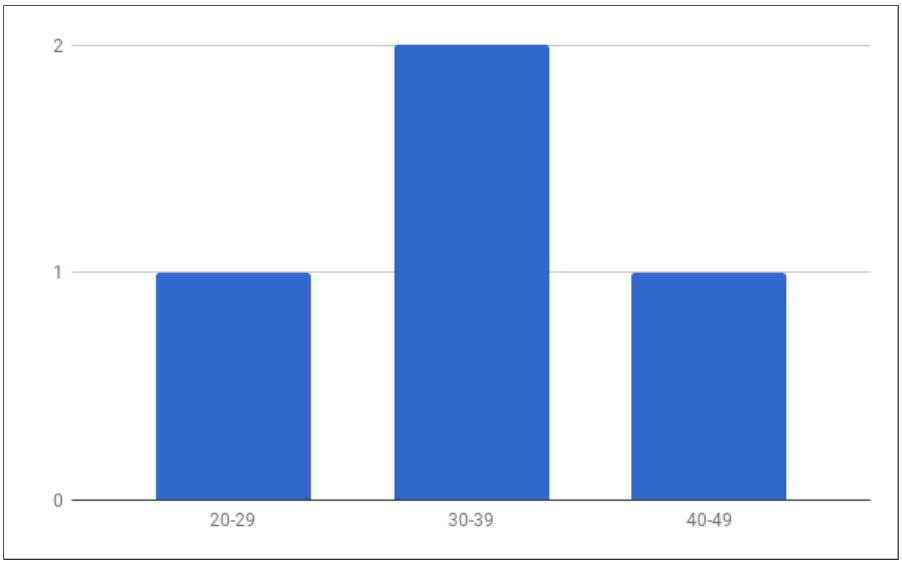


Figure 6.7.: Distribution of the participants' age

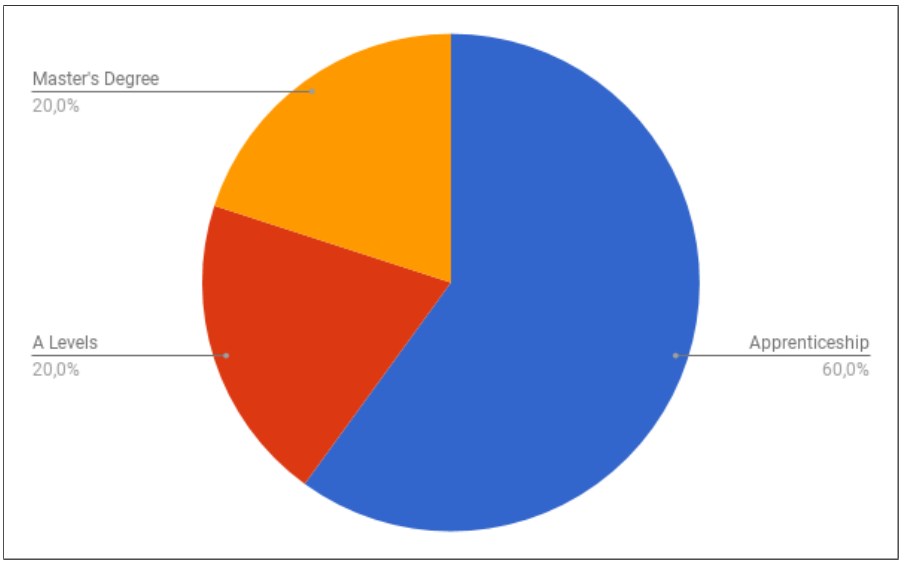


Figure 6.8.: Distribution of the participants' educational background

6.4.2. Usability Test Setup

The usability test was performed in a separate room at the energy providers' service center where five maintenance technicians participated the test. It is worth noting, that the test users were involved in the design process of the interface. Thus, some usability issues may not have been revealed during the *Thinking aloud* test session. Each user was tested individually and asked not to talk to any colleagues after finishing the test. The whole session was recorded using a screen capture software on the mobile phone, and a camera to record the users' faces. The session was started with an introduction of the test methodology and the basic functions of the interface. Afterwards, the users had to perform four tasks using the interface of the *Operations Center* and seven tasks using the mobile interface. The prepared tasks simulated a typical maintenance task, where an error is reported by the condition monitoring software, a service technician receives a maintenance task, diagnoses the fault at the turbine and creates a report with the performed repair/replacement activity. In order to avoid a potential language barrier, the tasks were translated to German. The originally presented tasks can be found in Appendix B.2. At the end of the usability test, users were asked about their opinions on the interface. Further, an SUS questionnaire was filled out by each of the five service technicians. In order to avoid the language barrier, a German translation [55] was used (see Appendix C).

6.4.3. Tasks

The following tasks were presented to the service technicians, where each task was presented on a separate sheet of paper.

Desktop Task 1: Investigate diagnosis results

Challenge:
Your supervisor informs you that a new diagnosis result from turbine instance 100111 was reported in Uptime Harvest.

- Go to the diagnosis area and investigate the results.

Goal:
Users should recognize the menu item *Diagnosis Results* and click on it to open the diagnosis area. The first collapsible panel inside the tab-bar is the error report from turbine instance 100111. The task is completed once the user clicks on the panel and notices the three IGBT fault candidates.

Desktop Task 2: Create a repair task

Challenge:
Create a repair task for instance 100111.

- The repair task should be performed by service technician A, B and C.
- Supervisor should be service technician B

Goal:
Users should click on the *Create task* button and enter the provided information. The task is completed once the user clicks on the *Send* button.

Mobile Task 1

Challenge:
You have received a new repair task. The "Home"-Screen gives you an overview of the available repair tasks.

- Open the menu and get an overview of the available menu options.

Goal:
To complete the task, the user needs to click on the menu icon.

Mobile Task 2

Challenge:
Navigate to the "Preparation"-screen and prepare spare parts and tools. Confirm when preparation is done.
Goal:

The users need to recognize the menu item *Preparation* and click on it. The task is completed once the user clicks on the *Continue* button in the *Preparation* area.

Mobile Task 3

Challenge:
You are at turbine 100111. Before continuing with the repair task, the enforcement permit needs to be requested.
Goal:
Users should navigate to the *Overview* area of the turbine instance. The task is completed once the user clicks on the *Acquire enforcement permit* button.

Mobile Task 4

Challenge:
The alarm code reported by the turbine is equal to the alarm code in the repair task assignment. Confirm the alarm code.
Goal:
Users should click on the menu item *Alarm code*. To complete the task, users need to click on the *Confirm* button.

Mobile Task 5

Challenge:
Navigate to the diagnosis screen and get an overview of the possible faults.
Goal:
Task is completed once the user clicks on the menu item *Diagnosis*.

Mobile Task 6

Challenge:
You make the following observations:

- Deviation of the power output
- No component damage observable
- No burn marks observable

Insert the observations and update the diagnosis results.

- Which fault has the highest probability after the update?
- How would you proceed to get a more precise result?

Goal:
The user need to enter the provided probing points and answer the questions. The task is completed once the user identifies the fault with 100% probability.

Mobile Task 7

Challenge:
You have identified a thermo-mechanical fatigue. The IGBT module with serial number 313125 has been replaced.

- Navigate to „Report“-screen
- Enter the diagnosed fault.
- Enter the performed repair task.
- Enter the serial number of the module.
- Enter the work time.

Goal:
The users need to click on the menu item *Report* and enter the provided information. The task is completed, once the user clicks on the *Send report* button.

6.5. Results

The analysis of the recorded video material revealed several usability issues. The collected issues were rated based on their severity using the following scale [35]:

5 - Critical usability problem: An issue classified with this rating needs to be resolved before the product can be released.

4 - Major usability problem: Finding and implementing a solution for this issue has high priority.

3 - Minor usability problem: Fixing this type of issue has low priority.

2 - Cosmetic problem: Issues of this type do not need to be fixed unless extra time for this project is available.

1 - No problem: This issue can be classified as a non-usability problem.

An overview of the identified usability errors is depicted in Table 6.2.

6.5.1. Critical Issues

Four usability issues with critical priority have been identified.

Users overlook location and component information: During the diagnosis refinement, all five service technicians had problems recognizing the location or the component where the observation should be made (see Figure 6.9a). In order to direct the users' attention to this important information, the font size can be increased. Further, the padding of the headline needs to be decreased to create a stronger relationship between the location information and the question.

***Add diagnosed fault* button hard to find:** Although the *Add diagnosed fault* button (depicted in Figure 6.9b) is in a very prominent spot, users did not recognize the button. A solution to this issue can be achieved by adding a background color to buttons. However, it could be observed that users got distracted by the list of performed checks. Since this is redundant information, it should only be displayed if requested by the user.

***Continue* button has wrong label:** To complete the *Preparation* stage, users had to click on the *Continue* button depicted in Figure 6.9c. This caused irritation, since this action was required to proceed to the next step but

the screen did not change. Therefore, the label of the button needs to be renamed to "Confirm", "Acknowledge", or similar.

Option for acquiring enforcement permit hard to find: Before entering a wind turbine, the service technicians have to acquire an enforcement permit. This option (depicted in Figure 6.9d) is located at the bottom of the *Preparation* screen of the maintenance task and not easy to find. The button should be placed below the *Overview* section in order to be visible without scrolling.

6.5.2. Major Issues

Two issues with high priority could be identified.

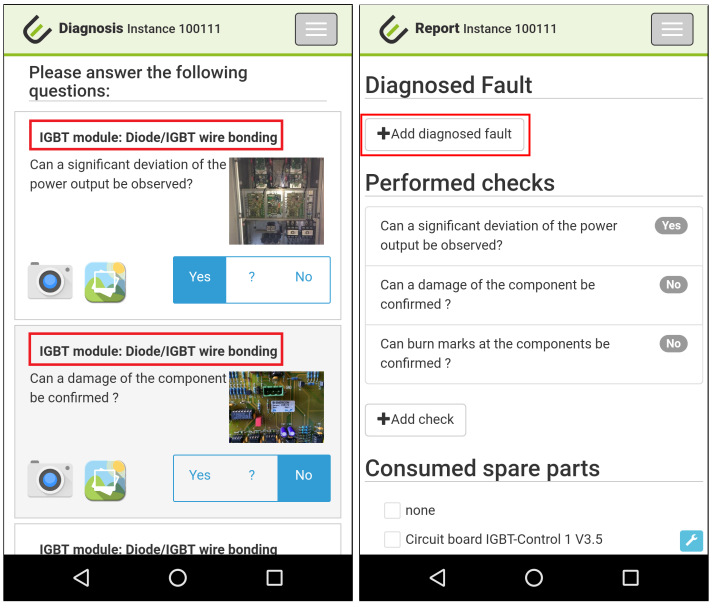
Highlight of selected list items: In order to add the diagnosed fault to the report, users had to select the fault from a list (see Figure 6.10a) and click on the *OK* button. Users got confused when they selected the identified fault from the list since the background of the item changed to gray, which was not easy to recognize. Rather than changing the background color, this issue can be resolved by inserting the fault to the report once the user clicks on the list item.

Adding a serial number is not obvious: During the report task, users were asked to enter a serial number to a consumed spare part. The icon (depicted in Figure 6.10b) which enabled this function was missing a label, therefore users could not associate the icon with the underlying functionality.

6.5.3. Minor Issues

The two low prioritized issues consist of minor design aspects.

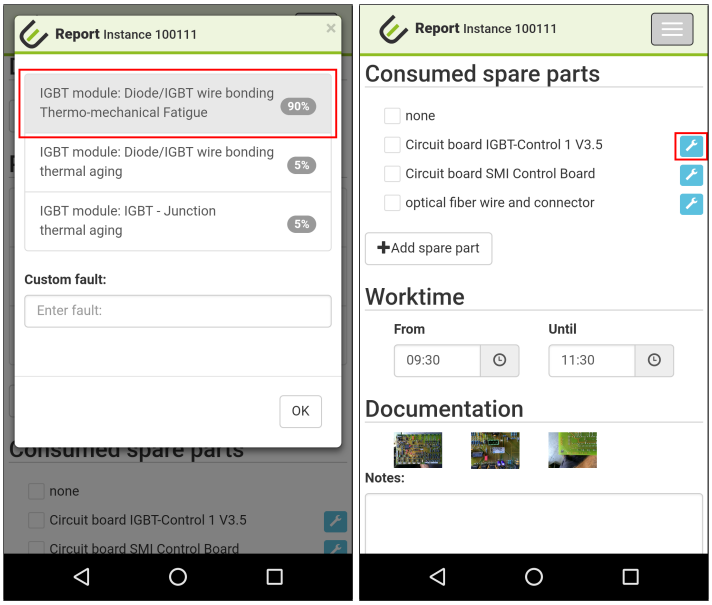
Misinterpretation of colored calendar days: A caption needs to be added to clarify the maintenance schedule calendar entries of the desktop interface shown in Figure 6.11a.



(a) Invisible information (b) Button gets overlooked

(c) Button has wrong label (d) Button hard to find

Figure 6.9.: Critical issues [29]



(a) Highlight color too bright (b) Icon is not understandable

Figure 6.10.: Major issues [29]

Creation date missing: The diagnosis results in the desktop interface included the creation date in the headline which was not present in the mobile interface (see Figure 6.11b).

6.5.4. Issues Identified as Non-usability Problems

Probability not obvious: One user had problems identifying the increased probability after refining the diagnosis results.

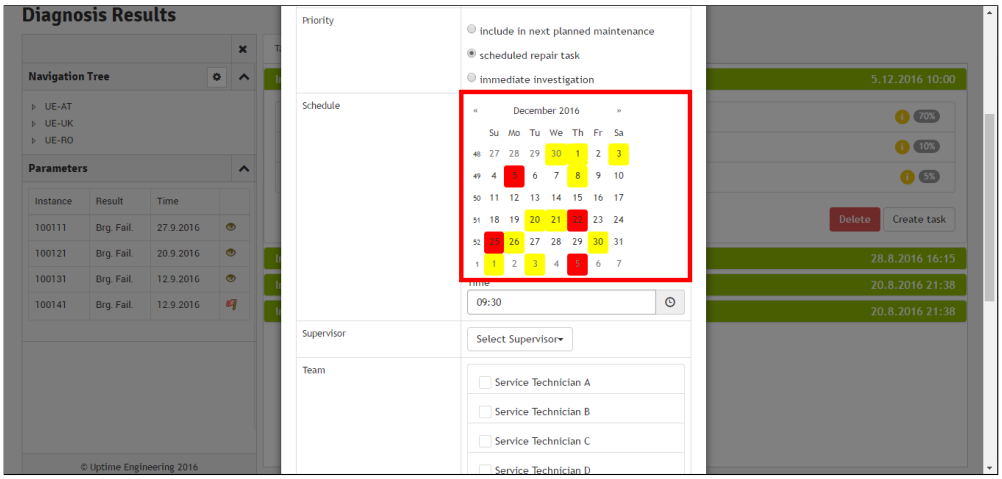
Interaction with time picker is hard: One user was unable to interact with the time picker.

6.5.5. Post-Test Interview

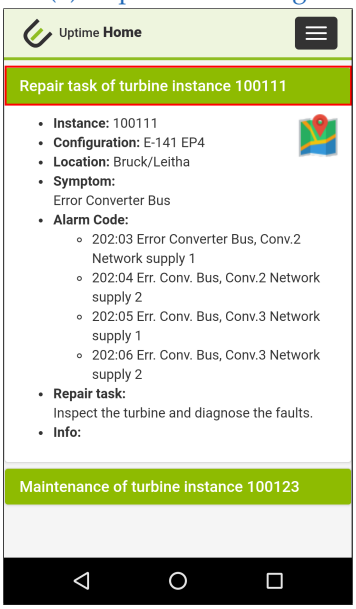
During the interview, all users mentioned that they are curious how the system will perform in the pilot phase. Further, three users remarked that they need more time to interact with the software in order to feel comfortable using it. One user pointed out, that it is hard to operate on a mobile phone with cold fingers. Another user thought that the diagnosis system is too much effort for troubleshooting wind turbines and results of diagnosis systems are often unreliable based on his personal experience.

6.5.6. Evaluation of the System Usability Scale Questionnaire

The results of the SUS questionnaire are depicted in Figure 6.12. Overall, the prototype was rated with a SUS mean score of 71.5 which is slightly above the average score of 68. The semantic differential (depicted in Figure 6.15) shows, that users had some oppositional opinions on the interface. From the evaluation of the SUS questionnaire follows, that the service technicians would like to use the system more frequently, yet not all of them felt confident using it. *User B* found the interface was too inconsistent, but also admitted that other people would learn the system very easily and he personally needs more experience with the software.



(a) Caption is missing



(b) Date is missing

Figure 6.11.: Minor issues [29]

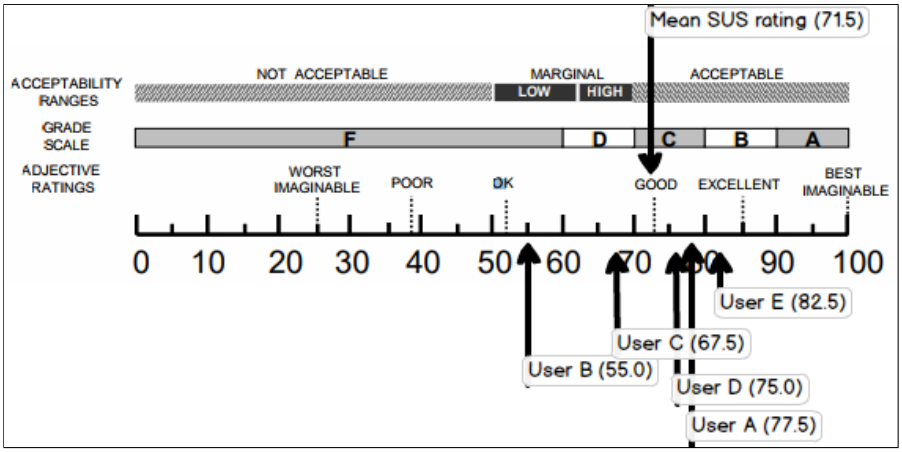


Figure 6.12.: SUS scores (adapted from [2])

6.5.7. Usability Metrics

As Nielsen [39] states, usability metrics can be used to measure the success of a redesigned interface compared with the original design. The user success rate [36] is a percentage value of how many tasks a user was able to complete successfully. Each completed task was rated with a score of 1, partial successful tasks with 0.5 and unsuccessful tasks with 0 points. The sum of the ratings are divided by the amount of tasks in order to get the success rate. The results of each user is depicted in Figure 6.13.

Figure 6.14 depicts the total number of errors the users made while performing the tasks. In order to determine the error rate, the sum of errors is divided by the number of users. The average error rate is shown in Table 6.1. Again, these values can be used to determine the impact of a redesign.

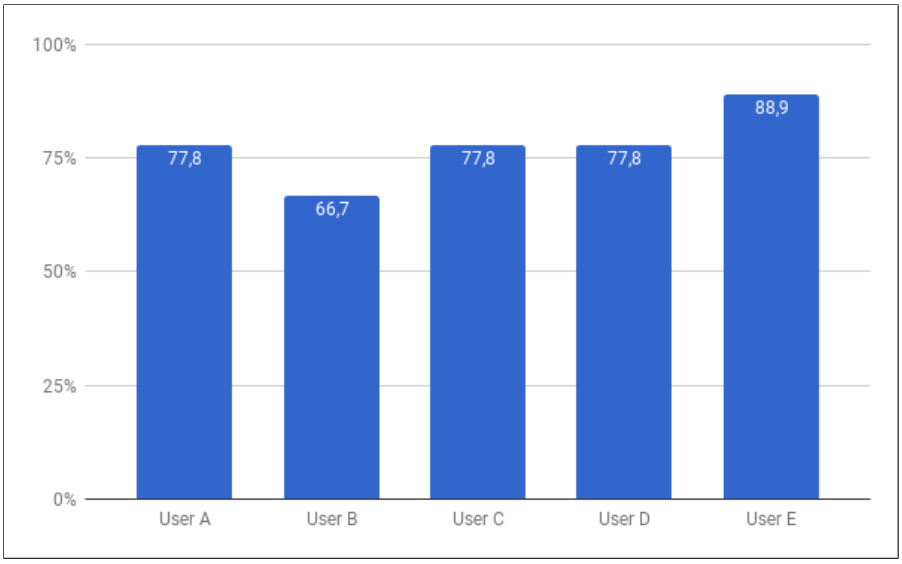


Figure 6.13.: User success rate

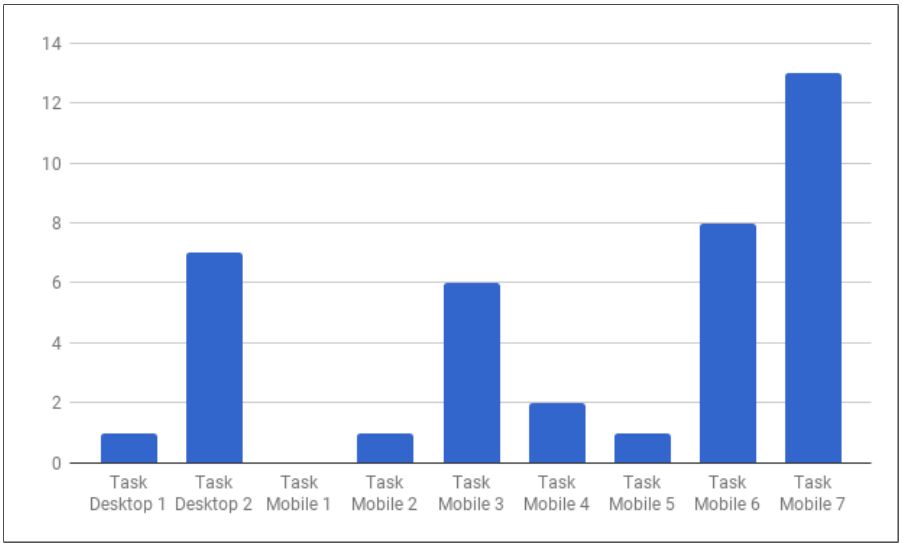


Figure 6.14.: User error rate

Task	Average number of errors per user
Task 1 Desktop	0.2
Task 2 Desktop	1.4
Task 1 Mobile	0.0
Task 2 Mobile	0.2
Task 3 Mobile	1.2
Task 4 Mobile	0.4
Task 5 Mobile	0.2
Task 6 Mobile	1.6
Task 7 Mobile	2.6

Table 6.1.: Average error rate

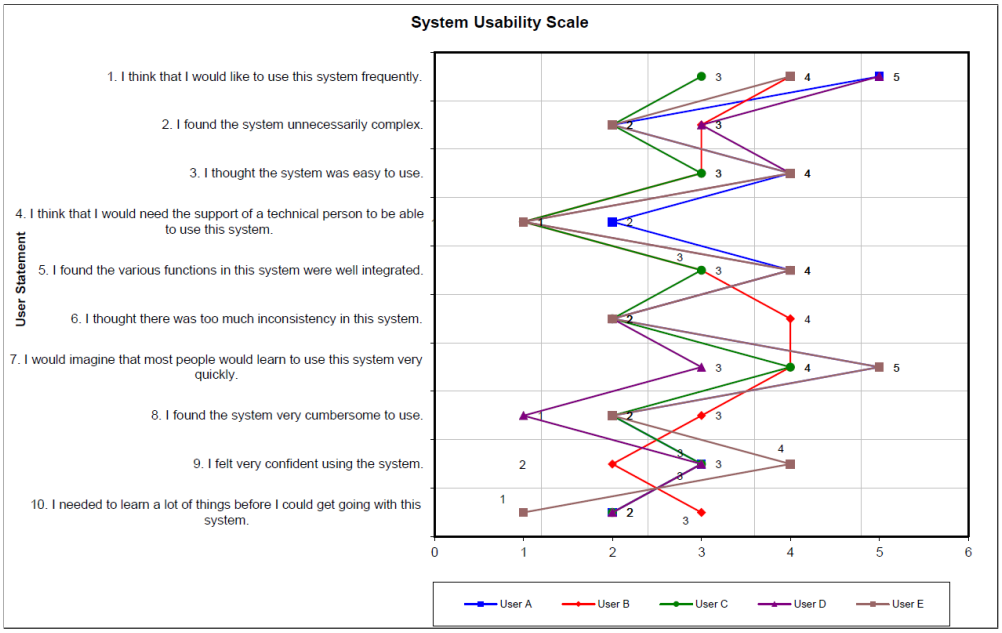


Figure 6.15.: Results of the SUS questionnaire

Issue	Description	Priority	Users
1	Users overlook location or components and get confused since the asked questions are the same, but belong to different components	5	5 / 5
2	Users overlook or do not recognize how to enter a repair task to a fault	5	5 / 5
3	Users think that the *Continue* button links to a location	5	4 / 5
4	Users do not know where to acquire enforcement permit	5	4 / 5
5	The used highlight color of selected list items is too bright and hard to see	4	3 / 5
6	Users have problems finding the button to enter the serial number for a replaced component	4	3 / 5
7	Users have problems interpreting the colored days in calendar	3	2 / 5
8	When creating a new repair task, users claimed that they were looking for a date but could not find any	3	1 / 5
9	User cannot immediately spot the increased probability	0	1 / 5
10	Users have problems using the time picker to insert a specific time	0	1 / 5

Table 6.2.: Identified usability issues

7. Conclusions and Future Work

The main purpose of this work was to design and evaluate an interface for model-based diagnosis applications. Since industrial model-based applications are sparse and research in terms of interface design is missing, knowledge-based systems and model-based research tools were investigated and their user interfaces analyzed. Based on the results of the interface inspections, a list of best practices for a new GUI design was created. In order to validate the usability of an interface, several usability inspection and testing methods were presented.
Diagnosis applications can be used in various different domains. Therefore, a scenario for the vast majority of users was developed. The *Smart Home* domain was viable to develop an interface supporting the needs of novice, casual, and expert users. Given the scenario and the target user groups, a general interface for a model-based diagnosis application was created with respect to the previously defined best practices. The performed *Thinking aloud* test revealed several usability issues and the continuous comments from the users during the test were used to identify the interface design errors. The evaluation of the SUS questionnaire showed that the interface is practicable for all three user groups.
The results of the usability study confirmed, that the designed user interface is suitable even for users with less experience in digital technologies. The participants of the test pointed out that they liked the design, layout, and interaction speed with the application. With an overall score of 86.4, the evaluation of the SUS questionnaire could confirm the positive feedback. Novice users rated the interface with 75.5, casual users with 93.4 and expert users with 86.1 points, which are all above the average score of 68. Therefore, the general interface proposal can be used as foundation for the successful integration of MBD applications in industrial environments.

In collaboration with *Uptime Engineering Gmbh*, a prototype for wind turbine

troubleshooting was created. During several meetings with all stakeholders, the requirements for this application were identified and the design refined after each iteration. Since the diagnosis system should support the service technicians, their current workflow was analyzed and taken into account. Although the service technicians were involved in the development process of the interface, the performed *Thinking aloud* test uncovered several usability issues. During the interview session, all users mentioned their skepticism on the diagnosis system as they had had negative experience with such systems in the past. They pointed out that the system will need to provide valid diagnosis results during the pilot phase under real working conditions in order to be useful. The results of the SUS questionnaire showed, that the mean SUS score of 71.5 points is slightly above the average value. The user success rate shows, that on average 77.7% of the tasks could be performed successfully. The calculated error rate underlines the importance of usability testing. Once the identified usability issues are resolved, the provided data can be used to benchmark the impact of the redesign.

The general interface presented in this work can be used as foundation for diagnosis applications for different domains. However, opportunities for future work exist. In certain scenarios, a web-application might not be practicable since a connection to a server is required. Depending on the target device, alternative solutions (e.g. native applications for mobile devices or desktop programs) can be built based on the interface specifications. Since style guidelines of other platforms might conflict with the presented design, an additional usability test will be required. Although the diagnosis engine is capable of determining multiple fault diagnoses, only single fault solutions were taken into account. If this additional feature is needed, a reevaluation of the the interface is required.
The interface prototype for wind turbine maintenance is currently implemented and integrated into *Uptime Engineering's* condition monitoring suite. Since the service technicians were very skeptical about the prototype, the feedback of the users during the pilot stage needs to be evaluated to confirm the users' acceptance.

Not covered within the interface proposal is the representation of the root-cause effect chain, which, for example, provides a possibility for the users to understand how the system computes the diagnoses. This could be an important factor for users to trust the results of the system and therefore

increase user acceptance. In order to ensure the correctness of the knowledge base, there is currently no possibility for users to update the abductive model. Additional data could be used to refine and improve the diagnosis process. Also, if a failure remains in the system after a repair task, the diagnosis needs to be repeated and the failure report should be updated accordingly.

Appendix

Appendix A.

General Diagnosis Interface Study

A.1. Background Questionnaire for the General Diagnosis Interface Study

Date: _____ Time: _____ Test Nr.: _____ User Nr.: _____

Background questionnaire

Thank you for participating this usability test.
Please answer the following questions:

1. General Information:

Gender: [ ] male [ ] female

Age: _____

Job: _____

Education: [ ] None [ ] Apprenticeship [ ] Master [ ] High school [ ] Studies

else _____

2. Sight Impairment

1. Do you use a sight aid when working on a computer?

[ ] None [ ] Glasses [ ] Contact lenses [ ] else _____

2. Are you color blind?

[ ] No [ ] Yes, namely _____

3. Experience with computers:

1. How long have you been using personal computers?

_____ *Years*

2. In a typical week, how many hours do you spend on a computer?

_____ *hours*

3. Which computer programs do you use at work?

_____

4. Which kind of personal computer do you use most?

[ ] Microsoft Windows [ ] Apple Macintosh [ ] Unix [ ] else _____

5. How many hours do you use mobile devices (Smartphone, PDA, eBook-Reader, ...)?

_____ *hours*

6. What kind of smartphone do you use?

[ ] Android Phone [ ] iPhone [ ] Windows Phone [ ] else _____

7. How many apps do you have installed on your smartphone?

[ ] < 5 [ ] 5-10 [ ] 11-15 [ ] > 15

4. Web Experience

1. How many hours per week do you use the World Wide Web??

_____ *hours*

2. Which kind of device do you use most often to access the World Wide Web?

[ ] Desktop PC [ ] Laptop [ ] Tablet
[ ] Smartphone [ ] else _____

3. Which web browser do you normally use?

[ ] Microsoft Internet Explorer [ ] Firefox [ ] Safari
[ ] Chrome [ ] Opera [ ] else _____

6. Experience with Usability tests

1. Have you ever participated in a usability study?

[ ] participant [ ] member of test team

If so, what kind of study did you participate in?

[ ] Thinking Aloud [ ] Formal Experiment [ ] else _____

	Novice users	Casual users	Expert users
Highest education qualification			
Compulsory education	1	0	0
Apprenticeship	2	1	0
A Levels	0	1	3
Specialized secondary school	0	2	0
University degree	2	4	4
Current job			
Student	0	1	6
Employee	1	6	0
Entrepreneur	2	1	0
Retiree	2	0	0
Sight Impairment			
Glasses	3	3	4
Contact lenses	0	2	1
None	2	3	2
PC experience (in years)			
< 10	1	0	0
10 - 15	1	3	1
16 - 20	1	3	3
> 20	2	0	2
PC usage per week (in hours)			
< 10	3	2	0
10 - 15	1	3	0
16 - 20	0	0	0
> 20	1	3	7
Applications used at work (multiple selections)			
Browser	3	4	3
Office	2	6	3
Job specific software	2	2	7
PC operating system			
Windows	5	7	1
Apple	0	1	2
Unix	0	0	4
Smartphone usage per week (in hours)			

< 10	4	1	3
10 - 15	0	5	3
16 - 20	0	1	0
> 20	1	1	1
Smartphone			
Android	5	6	5
iPhone	0	2	2
Number of installed apps			
< 5	4	0	0
5 - 10	1	3	2
11 - 15	1	0	0
> 15	0	5	5
Internet usage per week (in hours)			
< 10	4	3	0
11 - 15	1	4	0
16 - 20	0	1	6
> 20	0	0	1
Preferred device for surfing the web			
Laptop	1	2	5
Desktop	3	1	2
Smartphone	1	5	0
Used browser (multiple selections)			
Chrome	0	4	4
Firefox	4	2	4
Opera	0	1	0
Safari	0	1	1
Internet Explorer 1	0	0	
Experience with usability tests			
Participant	0	2	0
Member of test team	0	0	5
None	5	6	2

Table A.1.: Results of the background questionnaire

A.2. Tasks from the Evaluation of the General Interface

Task 0
Starten Sie das Programm und verschaffen Sie sich einen Überblick über die vorhandenen Funktionen.
Task 1a
Schalten Sie Lampe 1 ein. Sollte ein Problem auftreten, starten Sie bitte das Diagnose-System und diagnostizieren Sie das beobachtete Problem.
Task 1b
Sie wollen das Problem beheben. Finden Sie Informationen über die Ursache und Optionen zur Behebung des Fehlers.
Task 2a
Schalten Sie Lampe 2 ein. Sollte ein Problem auftreten, starten Sie bitte das Diagnose-System und diagnostizieren Sie das beobachtete Problem.
Task 2b
Verbessern Sie die Diagnose mittels Beobachtungen, bis Sie ein zufriedenstellendes Ergebnis erzielen.
Task 3a
Schalten Sie Lampe 1 ein und ändern Sie die Farbe. Sollte ein Problem auftreten, starten Sie bitte das Diagnose-System und diagnostizieren Sie das beobachtete Problem.
Task 3b
Verbessern Sie die Diagnose mittels Beobachtungen, bis Sie ein zufriedenstellendes Ergebnis erzielen.
Task 4
Senden Sie einen Bericht an den Hersteller des Lampensystems, um den Austausch des defekten Gerätes zu veranlassen. Fügen Sie den diagnostizierten Fehler in den Bericht ein. Fügen Sie ein Bild der Base in den Bericht ein, um die Bearbeitung in der Servicestelle des Herstellers zu beschleunigen.
Task 5
Interface Vergleich

Appendix B.

Wind Turbine Diagnosis Interface Study

B.1. Background Questionnaire for the Wind Turbine Diagnosis Interface Study

Datum: _____ Uhrzeit: _____ Test Nr.: _____ User Nr.: _____

Hintergrundbefragung

Danke, dass Sie sich als Freiwilliger für unseren Test zur Verfügung stellen.
Bitte beantworten Sie die folgenden Fragen:

1. Angaben zur Person

Geschlecht: [ ] männlich [ ] weiblich

Alter: _____

Beruf: _____

Abgeschlossene Lehre: _____

Bei Verbund seit: _____

2. Sehvermögen

1. Verwenden Sie eine Sehhilfe bei der Arbeit am Computer?

[ ] Keine [ ] Brille [ ] Kontaktlinsen [ ] sonstige _____

2. Sind Sie farbenblind?

[ ] Nein [ ] Ja, und zwar _____

3. Umgang mit Computern

1. Wie lange benutzen Sie bereits einen Personal Computer?

_____ *Jahre*

2.Wie viele Stunden pro Woche verwenden Sie einen Personal Computer?

_____ *Stunden*

3. Welche Computer-Programme verwenden Sie regelmäßig am Arbeitsplatz?

_____

4. Welches Betriebssystem verwenden Sie am häufigsten?

[ ] Microsoft Windows [ ] Apple Macintosh [ ] Unix [ ] sonstige _____

5. Wie viele Stunden pro Woche verwenden Sie mobile Geräte (Smartphone, PDA, eBook-Reader, ...)?

_____ *Stunden*

6. Welches Smartphone verwenden Sie?

*[ ] Android Phone [ ] iPhone [ ] Windows Phone [ ] sonstiges*_____

7. Wie viele Apps haben Sie auf Ihrem Smartphone installiert?

[ ] < 5 [ ] 5-10 [ ] 11-15 [ ] > 15

4. Umgang mit dem Internet und Web

1. Wie viele Stunden pro Woche benutzen Sie das World Wide Web?

_____ *Stunden*

2. Welches Gerät verwenden Sie am häufigsten zum Surfen?

[ ] Desktop PC [ ] Laptop [ ] Tablet
*[ ] Smartphone [ ] sonstiges*_____

3. Welchen Web-Browser verwenden Sie normalerweise?

[ ] Microsoft Internet Explorer [ ] Firefox [ ] Safari
[ ] Chrome [ ] Opera [ ] sonstiges _____

6. Erfahrung mit Usability Tests

1. Haben Sie schon an einer Usability Studie teilgenommen?

[ ] als Testperson [ ] als Mitglied des Testteams

Wenn ja, was war das für eine Studie?

[ ] Thinking Aloud [ ] Formal Experiment [ ] sonstiges _____

	User A	User B	User C	User D	User E
Gender	m	m	m	m	m
Age	32	26	36	42	32
Job	wind turbine service technicians				
Education	Apprenticeship			Electr. engineer	Master's degree
Sight aid	no	glasses	no	no	no
Color blindness	no	no	red/green	no	no
PC experience (years)	15	12	15	16	20
PC usage per week (hours)	15	12	20	24	30
Applications used	Office, Harvest, Internet Explorer				
PC	Windows				
Weekly mobile usage (hours)	20	2	40	10	8
Mobile	iPhone				
Installed apps	< 5	5 - 10	< 5	5 - 10	< 5
Weekly internet usage (hours)	10	6	10	8	4
Device used for surfing	Laptop	Laptop	Desktop	Smartphone	Laptop
Preferred browser	IE	Chrome	IE	Safari	IE

Table B.1.: Results of the background questionnaire

B.2. Tasks from the Evaluation of Wind Turbine Fault Diagnosis Interface

B.2.1. Operations Center Tasks

Task 1: Sie erhalten einen Anruf von ihrem Vorgesetzten, der sich gerade im Urlaub in der Karibik befindet. Er teilt Ihnen mit, dass er sich gerade mit seinem Smartphone in der Hotel-Lobby befindet und über die „Mobile Ansicht“ von Uptime Harvest bemerkt hat, dass eine Diagnose für Turbinen-Instanz 100111 aus der Gruppe „Uptime 1“ berechnet wurde.

- Navigieren Sie zu dem Diagnosebereich.

Task 2: Erstellen Sie einen Störungsbehebungsauftrag für die Instanz 100111.

- Der Auftrag soll so bald wie möglich von den Service Technikern A, B und C durchgeführt werden.
- Arbeitsverantwortlicher soll Service Techniker B sein.

B.2.2. Mobile Application Tasks

Task 1: Sie haben einen neuen Störungsbehebungsauftrag bekommen. In der Übersicht sehen Sie die verfügbaren Aufträge.

- Öffnen Sie das Menü und verschaffen Sie sich einen Überblick über die Menüpunkte.

Task 2: Navigieren Sie zum Vorbereitungs-Bereich und bereiten Sie die Ersatzteile und Werkzeuge vor. Bestätigen Sie die Vorbereitung von Ersatzteilen und Werkzeugen.

Task 3: Sie befinden sich vor der Windkraftanlage 100111. Bevor Sie mit der Arbeit beginnen, muss die Durchführungserlaubnis vom Anlagenverantwortlichen eingeholt werden.

Task 4: Der Alarm Code der Turbine stimmt mit dem Alarm Code der Auftragsbeschreibung überein. Bestätigen Sie den Alarm Code.

Task 5: Navigieren sie zum Diagnosebereich und verschaffen Sie sich einen Überblick über die möglichen Fehler.

Task 6: Sie machen folgende Beobachtungen:

- Ausgangsleistung am Konverter weicht ab
- Keine Bauteilbeschädigungen erkennbar
- Keine Brandmale an Bauteilen erkennbar

Geben Sie die oben angeführten Beobachtungen ein.

- Aktualisieren Sie die Diagnose-Ergebnisse
- Bei welchem Fehler konnte die Wahrscheinlichkeit erhöht werden?
- Wie würden Sie vorgehen, um den Fehler weiter einzugrenzen?

Task 7: Sie konnten eine thermo-mechanische Ermüdung als Fehler identifizieren und die zugehörige IGBT-Platine mit Seriennummer 313125 wurde getauscht.

- Navigieren Sie zum Report-Bereich.
- Fügen Sie den diagnostizierten Fehler in den Report ein.
- Fügen Sie die durchgeführte Störungsbehebung in den Report ein.
- Geben Sie die Seriennummer des eingebauten Moduls ein.
- Geben Sie die Arbeitszeit ein.
- Wie könnte ein zusätzliches Bild eingefügt werden?
- Wie könnten Sie Text in den Report eingeben?

Appendix C.

SUS Questionnaire

Fragebogen zur System-Gebrauchstauglichkeit

1. Ich denke, dass ich das System gerne häufig benutzen würde.

Stimme überhaupt nicht zu 1	2	3	4	Stimme voll zu 5
○	○	○	○	○

2. Ich fand das System unnötig komplex.

Stimme überhaupt nicht zu 1	2	3	4	Stimme voll zu 5
○	○	○	○	○

3. Ich fand das System einfach zu benutzen.

Stimme überhaupt nicht zu 1	2	3	4	Stimme voll zu 5
○	○	○	○	○

4. Ich glaube, ich würde die Hilfe einer technisch versierten Person benötigen, um das System benutzen zu können.

Stimme überhaupt nicht zu 1	2	3	4	Stimme voll zu 5
○	○	○	○	○

5. Ich fand, die verschiedenen Funktionen in diesem System waren gut integriert.

Stimme überhaupt nicht zu 1	2	3	4	Stimme voll zu 5
○	○	○	○	○

6. Ich denke, das System enthielt zu viele Inkonsistenzen.

Stimme überhaupt nicht zu 1	2	3	4	Stimme voll zu 5
○	○	○	○	○

7. Ich kann mir vorstellen, dass die meisten Menschen den Umgang mit diesem System sehr schnell lernen.

Stimme überhaupt nicht zu 1	2	3	4	Stimme voll zu 5
○	○	○	○	○

8. Ich fand das System sehr umständlich zu nutzen.

Stimme überhaupt nicht zu 1	2	3	4	Stimme voll zu 5
○	○	○	○	○

9. Ich fühlte mich bei der Benutzung des Systems sehr sicher.

Stimme überhaupt nicht zu 1	2	3	4	Stimme voll zu 5
○	○	○	○	○

10. Ich musste eine Menge lernen, bevor ich anfangen konnte das System zu verwenden.

Stimme überhaupt nicht zu 1	2	3	4	Stimme voll zu 5
○	○	○	○	○

Bibliography

[1] Keith Andrews. *Human-Computer Interaction, Course notes*. 2017 (cit. on pp. 52, 59–62, 82, 111).

[2] Aaron Bangor, Philip Kortum, and James Miller. "Determining what individual SUS scores mean: Adding an adjective rating scale." In: *Journal of usability studies* 4.3 (2009), pp. 114–123 (cit. on pp. 63, 98, 123).

[3] Carol M Barnum. *Usability testing essentials: ready, set... test!* Elsevier, 2010 (cit. on p. 79).

[4] Randolph G Bias. "The pluralistic usability walkthrough: coordinated empathies." In: *Usability inspection methods*. John Wiley & Sons, Inc. 1994, pp. 63–76 (cit. on p. 56).

[5] John Brooke et al. "SUS-A quick and dirty usability scale." In: *Usability evaluation in industry* 189.194 (1996), pp. 4–7 (cit. on p. 62).

[6] M-O Cordier, Philippe Dague, François Lévy, Jacky Montmain, Marcel Staroswiecki, and Louise Travé-Massuyès. "Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34.5 (2004), pp. 2163–2177 (cit. on pp. 4, 5).

[7] Beth Crandall, Gary A Klein, and Robert R Hoffman. *Working minds: A practitioner's guide to cognitive task analysis*. Mit Press, 2006 (cit. on p. 25).

[8] Fred D Davis. "A technology acceptance model for empirically testing new end-user information systems: Theory and results." PhD thesis. Massachusetts Institute of Technology, 1985 (cit. on p. 20).

[9] Fred D Davis. "Perceived usefulness, perceived ease of use, and user acceptance of information technology." In: *MIS quarterly* (1989), pp. 319–340 (cit. on pp. 1, 23, 27, 104).

[10] Johan De Kleer. "An assumption-based TMS." In: *Artificial intelligence* 28.2 (1986), pp. 127–162 (cit. on p. 39).

[11] Johan De Kleer. "Problem solving with the ATMS." In: *Artificial Intelligence* 28.2 (1986), pp. 197–224 (cit. on p. 39).

[12] Johan De Kleer and Brian C Williams. "Diagnosing multiple faults." In: *Artificial intelligence* 32.1 (1987), pp. 97–130 (cit. on p. 6).

[13] H Elmqvist, F Boudaud, J Broenink, D Brück, T Ernst, P Fritzson, A Jeandel, K Juslin, M Klose, SE Mattsson, et al. "Modelica - A unified object-oriented language for physical systems modeling." In: *Tutorial and Rationale* (1999) (cit. on p. 30).

[14] Martina Freiberg, Albrecht Striffler, and Frank Puppe. "Extensible prototyping for pragmatic engineering of knowledge-based systems." In: *Expert Systems with Applications* 39.11 (2012), pp. 10177–10190 (cit. on p. 1).

[15] Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. "Hypothesis classification, abductive diagnosis and therapy." In: *Expert Systems in Engineering Principles and Applications* (1990), pp. 69–78 (cit. on p. 10).

[16] Brad Frost. *Creating a Mobile-First Responsive Web Design*. Apr. 2012. URL: https://www.html5rocks.com/en/mobile/responsivedesign/ (cit. on p. 72).

[17] Amir Ghahrai. *Iterative Model*. Nov. 2008. URL: https://www.testingexcellence.com/iterative-model/ (cit. on p. 103).

[18] Dale L Goodhue and Ronald L Thompson. "Task-technology fit and individual performance." In: *MIS quarterly* (1995), pp. 213–236 (cit. on p. 24).

[19] Georg Gottlob, Thomas Frühwirth, and Werner Horn. *Expertensysteme*. Springer Vienna, 1990 (cit. on pp. 1, 14).

[20] Galen Gruman. *Heed these 10 expert tips for mobile app design*. Oct. 2013. URL: https://www.infoworld.com/article/2612190/mobile-apps/heed-these-10-expert-tips-for-mobile-app-design.html (cit. on p. 108).

[21] T Guckenbiehl, L Hotz, and P Struss. "INDIA - Intelligente Diagnose in der industriellen Anwendung." In: (2000) (cit. on p. 13).

[22] O Isaksson. "Model-based diagnosis of a satellite electrical power system with RODON." PhD thesis. Thesis-Linköpings University, 2009 (cit. on pp. 32, 34, 36, 37).

[23] W Iso. "9241-11. Ergonomic requirements for office work with visual display terminals (VDTs)." In: *The international organization for standardization* 45 (1998) (cit. on p. 52).

[24] Michael Jendryschik. *Mobile Usability – Gebrauchstauglichkeit für unterwegs*. Dec. 2011. URL: http://webkrauts.de/artikel/2011/mobile-usability-gebrauchstauglichkeit-fuer-unterwegs (cit. on p. 63).

[25] Annette Kluge and Anatoli Termer. "Human-centered design (HCD) of a fault-finding application for mobile devices and its impact on the reduction of time in fault diagnosis in the manufacturing industry." In: *Applied Ergonomics* 59 (2017), pp. 170–181 (cit. on pp. 23, 26).

[26] Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M Henne. "Controlled experiments on the web: survey and practical guide." In: *Data mining and knowledge discovery* 18.1 (2009), pp. 140–181 (cit. on p. 62).

[27] Roxane Koitz, Johannes Lüftenegger, and Franz Wotawa. "Model-Based Diagnosis in Practice: Interaction Design of an Integrated Diagnosis Application for Industrial Wind Turbines." In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2017, pp. 440–445 (cit. on pp. 66, 107, 108).

[28] Roxane Koitz and Franz Wotawa. "From theory to practice: Model-based diagnosis in industrial applications." In: *Proceedings of the annual conference of the prognostics and health management society*. 2015 (cit. on p. 65).

[29] Roxane Koitz, Franz Wotawa, Johannes Lüftenegger, Christopher S. Gray, and Franz Langmayr. "Wind Turbine Fault Localization: A Practical Application of Model-Based Diagnosis." In: *Diagnosis and Diagnosability of Hybrid Dynamic Systems: Challenges, Methods and Applications*. Springer. 2017 (cit. on pp. 9–11, 109, 119, 120, 122).

[30] Shu-Hsien Liao. "Expert system methodologies and applications - a decade review from 1995 to 2004." In: *Expert systems with applications* 28.1 (2005), pp. 93–103 (cit. on p. 14).

[31] Karin Lunde. "Object-oriented modeling in model-based diagnosis." In: *Proceedings of Modelica Workshop, Lund, Sweden*. 2000, pp. 111–118 (cit. on pp. 30, 31).

[32] Karin Lunde, Rüdiger Lunde, and Burkhard Münker. "Model-based failure analysis with RODON." In: *Proceedings of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29–September 1, 2006, Riva del Garda, Italy*. IOS Press. 2006, pp. 647–651 (cit. on p. 31).

[33] Heiko Milde, Thomas Guckenbiehl, Andreas Malik, Bernd Neumann, and Peter Struss. "Integrating model-based diagnosis techniques into current work processes–three case studies from the INDIA project." In: *AI Communications* 13.2 (2000), pp. 99–123 (cit. on p. 13).

[34] Jakob Nielsen. "Iterative user-interface design." In: *Computer* 26.11 (1993), pp. 32–41 (cit. on pp. 102, 103).

[35] Jakob Nielsen. "Severity ratings for usability problems." In: *Papers and Essays* 54 (1995), pp. 1–2 (cit. on pp. 88, 116).

[36] Jakob Nielsen. "Success rate: the simplest usability metric." In: *Jakob Nielsen's Alertbox* 18 (2001) (cit. on pp. 97, 98, 123).

[37] Jakob Nielsen. *Usability engineering*. Elsevier, 1994 (cit. on pp. 52, 80).

[38] Jakob Nielsen. "Usability inspection methods." In: *Conference companion on Human factors in computing systems*. ACM. 1995, pp. 377–378 (cit. on pp. 53, 62).

[39] Jakob Nielsen. *Usability Metrics*. Jan. 2001. URL: https://www.nngroup.com/articles/usability-metrics/ (cit. on p. 123).

[40] Jakob Nielsen and Raluca Budiu. *Mobile usability*. MITP-Verlags GmbH & Co. KG, 2013 (cit. on p. 108).

[41] Jakob Nielsen and Robert L. Mack. *Usability Inspection Methods*. John Wiley & Sons, 1994 (cit. on pp. 53, 54).

[42] Jakob Nielsen and Thomas K. Landauer. "A Mathematical Model of the Finding of Usability Problems." In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. CHI '93. Amsterdam, The Netherlands: ACM, 1993, pp. 206–213 (cit. on pp. 59, 80).

[43] Jakob Nielsen and Rolf Molich. "Heuristic evaluation of user interfaces." In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 1990, pp. 249–256 (cit. on p. 54).

[44] Divya Mishra Deepak Painuli Nirvikar. "Rule Based Expert System for Medical Diagnosis - A Review." In: *International Journal of Engineering Technology, Management and Applied Sciences* (2016) (cit. on p. 14).

[45] Jukka K Nurminen, Olli Karonen, and Kimmo Hätönen. "What makes expert systems survive over 10 years-empirical evaluation of several engineering applications." In: *Expert Systems with Applications* 24.2 (2003), pp. 199–211 (cit. on p. 1).

[46] Erica L Olmsted-Hawala, Elizabeth D Murphy, Sam Hawala, and Kathleen T Ashenfelter. "Think-aloud protocols: a comparison of three think-aloud protocols for use in testing data-dissemination web sites for usability." In: *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM. 2010, pp. 2381–2390 (cit. on p. 79).

[47] Victor C Osamor, Ambrose A Azeta, and Oluseyi O Ajulo. "Tuberculosis-Diagnostic Expert System: An architecture for translating patients information from the web for use in tuberculosis diagnosis." In: *Health Informatics Journal* 20.4 (2014). PMID: 24448278, pp. 275–287 (cit. on pp. 14–17).

[48] Claudia Picardi. *A Short Tutorial on Model-Based Diagnosis* (cit. on pp. 6, 7, 9).

[49] Ingo Pill, Gerald Steinbauer, and Franz Wotawa. "A practical approach for the online diagnosis of industrial transportation systems." In: *IFAC Proceedings Volumes* 42.8 (2009), pp. 1318–1323 (cit. on pp. 48, 50).

[50] Scott Poll, David L Iverson, and Ann Patterson-Hine. "Characterization of model-based reasoning strategies for use in IVHM architectures." In: *AeroSense 2003*. International Society for Optics and Photonics. 2003, pp. 94–105 (cit. on p. 30).

[51] Belarmino Pulido, Carlos J Alonso-González, Anibal Bregon, Alberto Hernández, and David Rubio. "DxPCs: A software tool for consistency-based diagnosis of dynamic systems using Possible Conflicts." In: *25st Annual Workshop Proceedings, DX-14*. 2014 (cit. on pp. 43–47).

[52] R. Reiter. "A theory of diagnosis from first principles." In: *Artificial Intelligence* 32.1 (Apr. 1987), pp. 57–95 (cit. on pp. 6, 39, 48).

[53] Sirpa Riihiaho. "The pluralistic usability walk-through method." In: *Ergonomics in Design* 10.3 (2002), pp. 23–27 (cit. on pp. 56, 57).

[54] Jeffrey Rubin and Dana Chisnell. *Handbook of usability testing: howto plan, design, and conduct effective tests*. John Wiley & Sons, 2008 (cit. on pp. 58, 62).

[55] Bernard Rummel. *System Usability Scale – jetzt auch auf Deutsch.* Jan. 2015. URL: https://experience.sap.com/skillup/system-usability-scale-jetzt-auch-auf-deutsch/ (cit. on p. 113).

[56] Jeff Sauro. *A Practical Guide to Measuring Usability: 72 Answers to the Most Commom Questions about Quantifying the Usability of Websites and Software*. Measuring Usability LLC, 2010 (cit. on p. 98).

[57] Jeff Sauro. *A practical guide to the system usability scale: Background, benchmarks & best practices*. Measuring Usability LLC, 2011 (cit. on p. 63).

[58] Jeff Sauro and Erika Kindlund. "A method to standardize usability metrics into a single score." In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2005, pp. 401–409 (cit. on p. 16).

[59] Edward H Shortliffe, Randall Davis, Stanton G Axline, Bruce G Buchanan, C Cordell Green, and Stanley N Cohen. "Computer-based consultations in clinical therapeutics: explanation and rule acquisition capabilities of the MYCIN system." In: *Computers and biomedical research* 8.4 (1975), pp. 303–320 (cit. on p. 1).

[60] Fajar Suryani, Izzati Muhimmah, and Sri Kusumadewi. "Preferred model of dialog style in expert system of physical examination of skin disease." In: *Science in Information Technology (ICSITech), 2015 International Conference on*. IEEE. 2015, pp. 247–252 (cit. on pp. 18, 19, 21, 22).

[61] Louise Travé-Massuyès and Robert Milne. "Gaps between research and industry related to model based and qualitative reasoning." In: *Proceedings of the European workshop on Model based systems and qualitative reasoning*. 1998, pp. 54–57 (cit. on p. 13).

[62] Ivo Weevers. *App Design Guidelines For High-Performance Mobile User Experiences*. July 2011. URL: https://www.smashingmagazine.com/2011/07/seven-guidelines-for-designing-high-performance-mobile-user-experiences/ (cit. on p. 108).

[63] Cathleen Wharton, John Rieman, Clayton Lewis, and Peter Polson. "The cognitive walkthrough method: A practitioner's guide." In: *Usability inspection methods*. John Wiley & Sons, Inc. 1994, pp. 105–140 (cit. on p. 56).

[64] Kathryn Whitenton. *Flat vs. Deep Website Hierarchies*. Nov. 2013. URL: https://www.nngroup.com/articles/flat-vs-deep-hierarchy/ (cit. on p. 110).

[65] Franz Wotawa. "Failure Mode and Effect Analysis for Abductive Diagnosis." In: *DARe ECAI*. 2014 (cit. on p. 65).

[66] Franz Wotawa. *Non-monotonic Reasoning in Artificial Intelligence*. 2010 (cit. on p. 39).

[67] Franz Wotawa. *The Logic Reasoning System* (cit. on pp. 7, 8, 38, 39).

[68] Franz Wotawa, Ignasi Rodriguez-Roda, and Joaquim Comas. "Abductive Reasoning in Environmental Decision Support Systems." In: *AIAI workshops*. 2009, pp. 270–279 (cit. on p. 10).