

Extending Failure Modes and Effects Analysis Approach for Reliability Analysis at the Software Architecture Design Level*

Hasan Sozer, Bedir Tekinerdogan, and Mehmet Aksit

Department of Computer Science, University of Twente,
P.O. Box 217 7500 AE Enschede, The Netherlands
{sozerh,bedir,aksit}@ewi.utwente.nl

Abstract. Several reliability engineering approaches have been proposed to identify and recover from failures. A well-known and mature approach is the Failure Mode and Effect Analysis (FMEA) method that is usually utilized together with Fault Tree Analysis (FTA) to analyze and diagnose the causes of failures. Unfortunately, both approaches seem to have primarily focused on failures of hardware components and less on software components. Moreover, for utilizing FMEA and FTA very often an existing implementation of the system is required to perform the reliability analysis. We propose extensions to FMEA and FTA to utilize them for the reliability analysis of software at the architecture design level. We present the software architecture reliability analysis approach (SARAH) that incorporates the extended FMEA and FTA. The approach is illustrated using an industrial case for analyzing reliability of the software architecture of a Digital TV.

Keywords: reliability analysis, FMEA, FTA, software architecture evaluation.

1 Introduction

A number of important trends can be observed in the development of embedded systems. First, due to the high industrial competition and the advances in hardware and software technology, there is a continuous demand for products with more functionality. Second, the functionality provided by embedded systems is shifting from hardware to software. Third, the functionality of embedded systems is not solely developed by just one manufacturer only but it is host to multiple parties. Finally, embedded systems are more and more integrated in networked environments that affect these systems in ways that might not have been foreseen during their construction. Altogether, these trends complicate the design and implementation of embedded systems. As a result, major steps in technology are required to keep the *reliability* [2] of these systems at the current level. Obviously, since embedded

* This work has been carried out as part of the TRADER project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Bsik program.

systems are now largely defined and controlled by software, it is also expected that the software failures form a major threat for reliability. Therefore, appropriate reliability analysis and design techniques should be provided to support the anticipation and prevention of potential failures.

In the literature, Failure Mode and Effect Analysis (FMEA) together with Fault Tree Analysis (FTA) are well-known and mature approaches to identify failure modes of system components and evaluate their impact on the system reliability. It appears though that these approaches have primarily focused on failures in hardware components and less on software components. In fact this is not so strange because historically, software formed only a small part of embedded systems and as such hardware components primarily defined the quality of the system. The developments and trends in embedded systems have provided an increasing awareness that reliability analysis should not be limited to hardware components but should also cover software components [6,7,15,16,21]. Further, for applying FMEA and FTA very often a running system is required and less focus has been given to reliability analysis before the system is actually implemented. However, it is of importance to analyze the failures earlier in the life cycle, at the software architecture design level. In this way, potential risks can be identified earlier, before committing organizational resources for implementing the system.

Obviously, FMEA and FTA have provided their clear merits for reliability analysis and it is worthwhile to extend and integrate these mature approaches in reliability analysis for software components. Hence, we propose extensions to FMEA and FTA and integrate them in our novel software architecture reliability analysis method (SARAH). Hereby, we first construct a *failure domain model* by analyzing the domain of failures and the relevant categories. The failure domain model is utilized to derive general failure scenarios that can be defined as possible types of failures. Failure scenarios are described using (adapted) FMEA worksheets. The prioritization of failure scenarios is different from the conventional use of FMEA and FTA where safety has been usually the main criteria for prioritizing failure scenarios. In the industrial project that we are working in [19], we focus on the consumer electronics domain, where safety is less or not an issue. Instead, it is the perception of the user for a failure that defines the severity of that failure. Therefore, we prioritize the failure scenarios based on their severity from the end-user perspective. Failure scenarios are then utilized to derive a *Fault Tree Set (FTS)*, which shows the causal and logical connections among the failures. FTS is used to perform *severity analysis*, in which we measure the impact and contribution of the other failures to the user perceived failures. In FTA, severity analysis uses the probabilities of failure occurrences [6] that are usually obtained from an already developed system. We introduce an impact model to estimate the overall impact of a failure even if the probability values are not known. To sum up, the extensions that we propose for FMEA and FTA relate to using a *failure domain model* for systematic derivation of failures, the prioritization of failure scenarios based on *user perception*, and finally an impact analysis model for FTA that does not explicitly require a running system. The extended approaches are utilized to perform *sensitivity analysis*, where we identify the sensitive components of the architecture. These are the components that are associated with the most severe failures. SARAH results in a failure analysis report that defines the sensitive

components of the architecture and provides information on the type of failures that might happen frequently.

The remainder of this paper is organized as follows. Section 2 provides the background information on conventional FMEA and FTA methods. In Section 3, we give an overview of the analysis method. In Section 4, we present the industrial case, in which software architecture for a Digital TV is introduced. This example will be used throughout the paper to illustrate the steps of SARAH and to discuss our experiences in the industrial context. Section 5 presents the specification and derivation of the failure scenarios. In section 6, severity analysis based on fault trees is explained. Section 7 presents the analysis of the architecture. Section 8 provides the related work. Finally, in Section 9 we provide the conclusions.

2 FMEA and FTA

FMEA [17] is a well-known and mature approach for reviewing the causes and effects of system failures systematically. The basic operations of the method are 1) to question the ways that each component fails (failure modes) and 2) to consider the reaction of the system to these failures (effects). The analysis results are organized by means of a worksheet, which comprises information about each failure mode, its causes, its local and global effects (concerning other parts of the product and the environment), the associated component and its severity. A simplified FMEA worksheet template is presented in Table 1.

Table 1. An example FMEA worksheet based on MIL-STD-1629A [12]

System:		<i>Car Engine</i>			
Date:		<i>10-10-2000</i>			
Compiled by:		<i>J. Smith</i>			
Approved by:		<i>D. Green</i>			
ID	Item ID	Failure Mode	Failure Causes	Failure Effects	Severity Class
<i>1</i>	<i>CE5</i>	<i>fails to operate</i>	<i>Motor shorted</i>	<i>Motor overheats and burns</i>	<i>V</i>
<i>2</i>	<i>...</i>	<i>...</i>	<i>...</i>	<i>...</i>	<i>...</i>

FMEA can be employed for risk assessment and for discovering potential single-point failures. Systematic analysis increases the insight in the system and the analysis results can be used for guiding the design, its evaluation and improvement. At the downside, some failure modes can be overlooked [14] and some information (e.g. failure probability, risk) regarding the failure modes can be incorrectly estimated at early design phases. Since the technique focuses on individual components at a time, combined effects and coordination failures can also be missed. In addition, FMEA process is effort and time consuming.

FMEA is usually applied together with FTA. FTA [6] is based on a graphical model (fault tree), which defines causal relationships between faults (a set of example fault trees can be seen in Figure 5). Faults, which are assumed to be provided, are

defined as undesirable system states or events that can lead to a system failure. The top node (i.e. root) of the fault tree represents the system failure and the leaf nodes represent faults. The nodes of the fault tree are interconnected with logical connectors (e.g. AND, OR gates) that infer propagation and contribution of faults to the failure. Once the fault tree is constructed, it can be processed in a bottom-up manner to calculate the probability that a failure would take place. This calculation is done based on the probabilities of fault occurrences and interconnections between the faults and the failure [6]. Additionally, the tree can be processed in a top-down manner for diagnosis to determine the potential faults that may cause the failure.

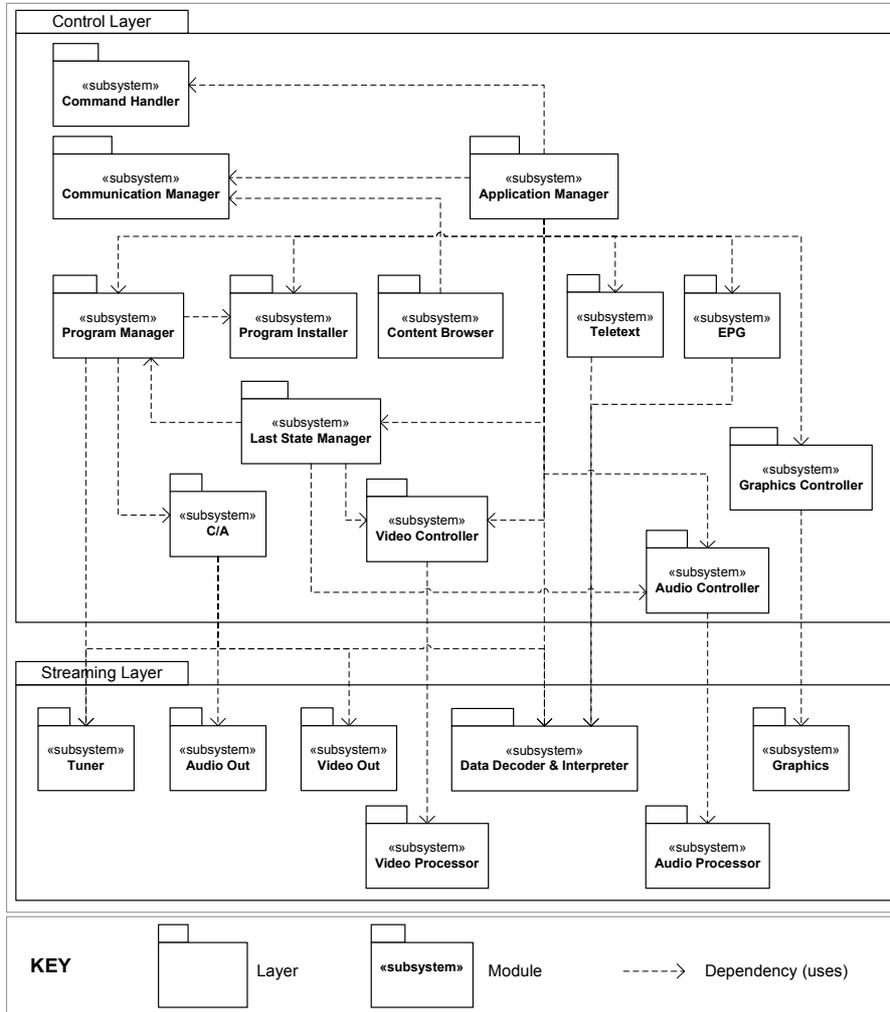
3 Industrial Case: Digital TV (DTV) Software Architecture

At the Embedded Systems Institute (ESI, Netherlands), the TRADER (Television Related Architecture and Design to Enhance Reliability) project is carried out together with NXP Semiconductors and several other academic and industrial partners [19]. The objective of the project is to develop methods and tools for ensuring reliability of digital television (DTV) sets. Due to the increasing size and complexity of software that is embedded in TVs, in practice, it is not feasible to design a perfect system that is fault-free. To cope with the faults that cannot be detected, appropriate fault tolerance mechanisms are required. In the DTV design in the TRADER project, faults of which effects can be directly observed by the user require a special attention. Such faults are considered to be important and they should be tolerated. Because TRADER aims to anticipate also on faults in future releases, it is also important that reliability analysis techniques are defined at the design level. From this perspective one of the key aims in TRADER is the design of fault tolerant software architecture with respect to the perception of TV users.

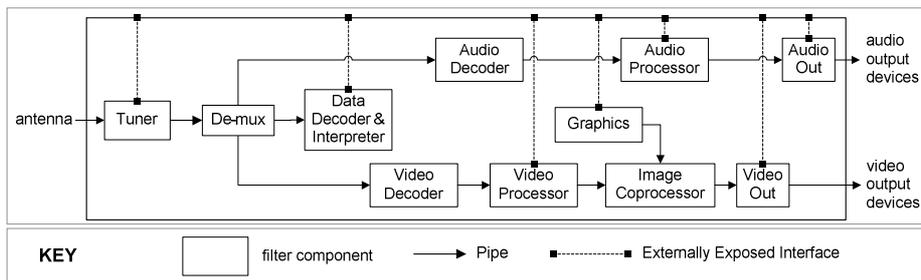
A conceptual architecture of DTV is depicted in Figure 1, which will be referred to throughout the paper. Figure 1(a) presents a module view [4] of the whole DTV structure with implementation units and direct dependencies among them. It mainly comprises two layers. The bottom layer, namely the streaming layer, involves modules taking part in streaming of audio/video information. The upper layer consists of application-related and utility modules that control the streaming process. Figure 1(b) presents a view of the streaming layer in pipe-and-filter style [4]. The streaming layer modules shown in Figure 1(a) correspond to the streaming layer components in Figure 1(b) with the same names.

In the following, we briefly explain some of the important modules and components that are shown in the architecture. For brevity, the components for decoding and processing audio/video signals are not explained here.

Application Manager (AMR) initiates and controls execution of both resident and downloaded applications in the system. It keeps track of application states, user modes and redirects commands/information to specific applications or controllers accordingly. *Audio Controller* (AC), controls audio features like volume level, bass and treble based on commands received from AMR. *Command Handler* (CH) interprets externally received signals (i.e. through keypad or remote control) and sends corresponding commands to AMR. *Communication Manager* (CMR) employs



(a) Module View of the Digital TV Architecture



(b) Component-Connector View (pipe-and-filter style) of the Streaming Layer

Fig. 1. Conceptual Architecture of a Digital TV

protocols for providing communication with external devices. *Conditional Access (C/A)* authorizes information that is presented to the user. *Content Browser (CB)* presents and provides navigation of content residing in a connected external device. *Electronic Program Guide (EPG)* presents and provides navigation of electronic program guide. *Graphics Controller (GC)* is responsible for generation of graphical images corresponding to user interface elements. *Last State Manager (LSMR)* keeps track of last state of user preferences such as the volume level and the selected program. *Program Installer (PI)* searches and registers programs together with the channel information (e.g. frequency). *Program Manager (PM)* tunes to a specific channel. *Teletext (TXT)* handles acquisition, interpretation and presentation of the teletext pages. *Video Controller (VC)* controls the video features like scaling of the video frames based on commands received from AMR.

4 Fundamental Steps of the Analysis Method

Software architecture forms one of the key artifacts in the software development life cycle since it embodies the earliest design decisions and includes the gross-level components that directly impact the subsequent analysis, design and implementation. Accordingly, it is important that the architecture design supports the required qualities of the software system. In general static analysis of formal architectural models is applied or a set of architecture analysis methods as described in [5] are used to analyze and predict the quality of the system. In this paper, we focus on software architecture analysis methods that utilize scenarios for evaluating architectures. In general, these analysis methods take as input the architecture design and measure the impact of predefined scenarios on it to identify the potential risks and the sensitive points of the architecture. This helps to predict the quality of the system before it is built, thereby reducing unnecessary maintenance costs. A scenario is considered to be a brief description of some anticipated or desired use of the system.

We propose a scenario-based architecture analysis method, SARAH that incorporates FMEA and FTA. In contrast to general-purpose architectural analysis methods in which scenarios can refer to different quality factors, we utilize the notion of *failure scenario* to analyze the impact of failures. A failure scenario represents the possible occurrence of a failure in a given component. The steps of SARAH are presented in a UML activity diagram in Figure 2. The approach consists of two basic processes: (1) *Definition* (2) *Analysis*.

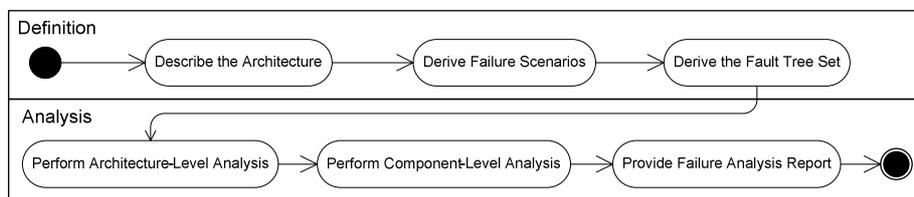


Fig. 2. The Steps of the Analysis Method

In the *definition process*, first the software architecture is described (section 3). Failure scenarios are then derived from a failure domain model that includes a categorization of failures. Failure scenarios are described using a template like FMEA worksheet (section 5). As a subsequent step the failure scenarios in the FMEA are utilized in the FTA (section 6). Here, we introduce the Fault Tree Set (FTS) concept, which is basically a set of possibly connected fault trees. We use an impact model based on FTS to estimate the impact of a failure scenario on the occurrence of the user-perceived failures. By combining the impact model together with the failure prioritization, we measure the severity of a failure scenario based on the type of user-perceived failures it can lead to and its contribution on the occurrence of these failures. Section 6 describes in detail our analysis based on fault trees.

Based on the input from the definition process, in the *analysis process*, an architectural level analysis (section 7.1) and a component level analysis (section 7.2) is performed. The results are presented in the failure analysis report (section 8). In the following sections the main steps of the method will be explained in detail using the industrial case study.

5 Specification and Derivation of Failure Scenarios

In this section we define the process for deriving and specifying failure scenarios. Section 5.1 introduces the scenario template that is used for describing failure scenarios. Section 5.2 explains the derivation of the failure domain model, which provides a categorization for failure scenarios. Section 5.3 explains the derivation of general failure scenarios. Finally, Section 5.4 explains the derivation of concrete failure scenarios based on general failure scenarios.

5.1 Failure Scenario Template

We define the concept of *failure scenario* to analyze the architecture with respect to reliability. Reliability is the ability of the system to function without a *failure*, which is as an event that occurs when the delivered service of a system deviates from a correct service [2]. A correct service is delivered when the service implements the required system function. An *error* is defined as the system state that may lead to a failure and the cause of an error is called a *fault* [2]. Figure 3 depicts the fundamental chain of threats that leads to a failure.



Fig. 3. The Fundamental Chain of Reliability Threats Leading to a Failure

A failure scenario specification includes a description of the *fault*, the *error* and the *failure* (See Table 2). Additionally, the *failure ID* is used to uniquely identify a scenario and the *component ID* defines the component for which the failure scenario applies. Note that this template is similar to the FMEA worksheet (See Table 1). To specify failure scenarios, we use *FID*, *CID*, *fault*, *error* and *failure* instead of the

Table 2. Template for Defining the Failure Scenarios

Attribute	Explanation
FID	A numerical value to identify the failures (i.e. Failure ID).
CID	An acronym defining the component for which the failure scenario applies (i.e. Component ID).
Fault	The description and the features of the cause of the error.
Error	The description and the features of the component state that leads to the failure.
Failure	The description and the features of the deviation of the component function from the required function.

concepts *failure ID*, *related component*, *failure cause*, *failure mode* and *failure effect*, respectively.

5.2 Derivation of the Failure Domain Model

The failure scenario template can be used to describe scenarios. However, there exist too many fault, error and failure types that can be considered. Hence, there is a high risk that several potential and relevant failure scenarios are missed or that other irrelevant failure scenarios are included. To define the relevant failures SARAH defines relevant domain model for faults, errors and failures using a systematic domain analysis process. This domain model provides a first scoping of the potential scenarios. In fact, several researchers have already focused on modeling and classifying failures for embedded systems. Avizienis et al [2], for example, provide a nice overview of this related work and provide a comprehensive classification of faults, errors and failures. The provided domain classification by Avizienis et al., however, is rather broad¹, and one can assume that for a given reliability analysis project not all the potential failures in this overall domain are relevant. Therefore, the given domain is further contracted by focusing only on the faults, errors and failures that are considered relevant for the actual project. Figure 4, for example, defines the derived domain model that is considered relevant for the DTV project. Figure 4 includes three feature diagrams that depict a categorization for faults, errors and failures.

In the feature diagram of fault (see Figure 4), faults are identified according to their *source*, *dimension* and *persistence*. In SARAH, failure scenarios are defined per component. For that reason, the source of the fault can be either (1) internal to the component in consideration, (2) caused by other component(s) of the system or (3) caused by the external entities with respect to the system. Faults can be introduced by software or hardware, and be transient or persistent. The relevant features of an error comprise the *type* of error together with its *detectability* and *reversibility* properties. The features for failure include the *type* and the *target*. The *target* of a failure can be the user or the other component(s) of the system.

¹ Due to space limitations we do not show this domain model and refer the interested reader to the corresponding publication.

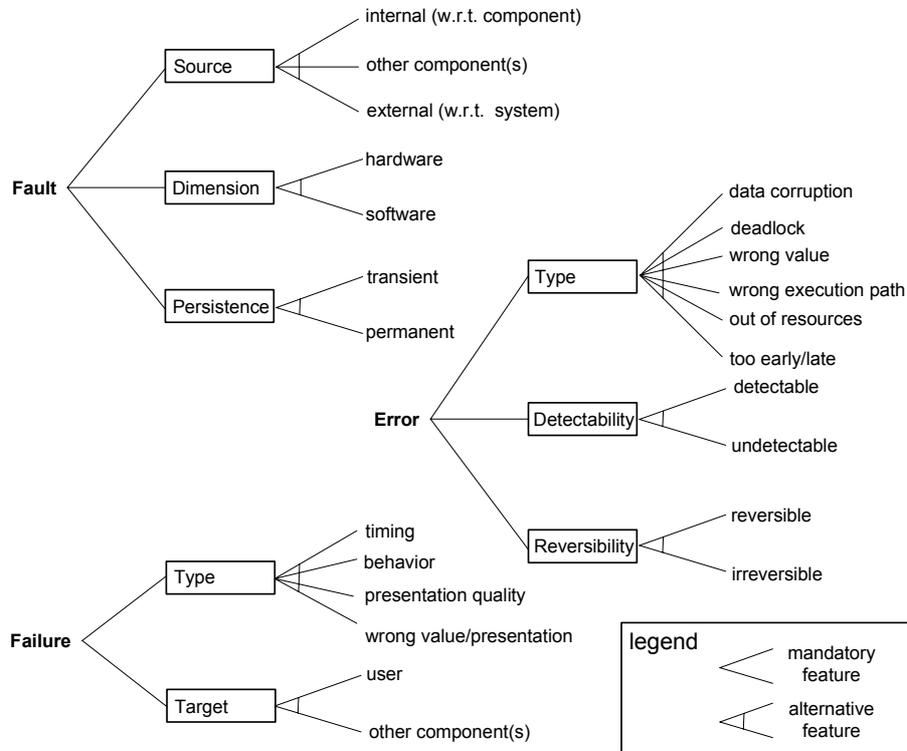


Fig. 4. Failure Domain Model: Classification of (relevant) Faults, Errors and Failures

In principle, for different project requirements one may come up with a slightly different domain model, but as we will show in the next sections this does not impact the steps in the analysis method itself. The key issue here is that the failure domain model is derived in accordance with the project requirements and the elements of a failure scenario are defined as instances of this model.

5.3 Derivation of the General Failure Scenarios

The domain model defines a system-independent specification of the type of faults, error and failures that could occur and is utilized for deriving so-called *general failure scenarios*². A general failure scenario is a system-independent failure scenario that includes selected sub-features from the fault, error and failure features. The number and type of general failure scenarios is implicitly defined by the failure domain model.

In the fault classification (see Figure 4), for example, we can define faults based on three features, namely *Source*, *Dimension* and *Persistence*. The feature *Source* can

² The notion of general failure scenario is a specialization of the general scenario as defined by Bachmann et al. [3].

have 3 different values, the features *Dimension* and *Persistence* 2 values. This means that the fault classification captures $3 \times 2 \times 2 = 12$ different faults. Similarly, from the error classification we can derive $6 \times 2 \times 2 = 24$ different errors, and $4 \times 2 = 8$ different failures are captured by the failure classification. Since a general scenario is a composition of selection of features from the failure domain model, we can state that for the given failure domain model in Figure 4, $12 \times 8 \times 24 = 2304$ general failure scenarios can be in principle defined. One of these general failure scenarios is, for example, shown in Table 3.

Table 3. Example General Failure Scenario Derived from the Failure Domain Model

Fault	Error	Failure
<i>source</i> : internal	<i>type</i> : too early/late	<i>type</i> : timing
<i>dimension</i> : software	<i>detectability</i> : detectable	<i>target</i> : user
<i>persistence</i> : transient	<i>reversibility</i> : irreversible	

5.4 Derivation of the Concrete Failure Scenarios

Although general failure scenarios provide an indication of possible failure scenario *types*, these do not reflect instantiated real failure scenarios. For example, the failure scenario specified in Table 3 does not specify the associated component, the actual fault, the error and the failure. SARAH includes additional steps to identify the relevant failure scenarios and to change the general failure scenarios to *concrete failure scenarios*. Unlike general failure scenarios, the concrete failure scenarios relate to specific components in the architecture. The steps for deriving concrete failure scenarios include (1) analyzing the domain (2) analyzing problems of existing systems, and (3) consulting domain experts. We can illustrate this for the adopted example case on the DTV architecture.

In the first step, we have analyzed the domain on corresponding digital TVs, including requirements specifications, design documentation of existing systems, literature on digital TV systems and real-time embedded systems.

In the second step, we have made use of the available DTV *Problem Database* (PRDB) that had been developed to report on failures of earlier TV systems. The primary goal of PRDB by recording failures was to solve the existing problems in current TV sets and not to support the analysis of the system. This had a clear impact on the PRDB. First of all, the reported problems were not in the format that we required, and we had to interpret these first. Further, the PRDB also included outdated information and we had to filter these out. Some problems were related to concrete and older products but since existing TV systems share common properties with next generation products, they have to cope with similar type of failures. For example, *out-of-spec signal* is a typical fault leading to numerous errors in extraction of Teletext information. Usually, specific instances of this type of fault have been reported in PRDB and appropriate solutions were provided. However, in general, reception of out-of-spec signals can pop up as a problem for every new TV system. By collecting such regular problems we could derive useful failure scenarios. Despite of the problems that we encountered in analyzing this PRDB it definitely provided valuable input.

Finally, in the last step for deriving concrete failure scenarios we have consulted digital TV domain experts to derive additional scenarios and to cross-validate the previously identified scenarios.

Based on these steps, we have derived and specified 44 concrete failure scenarios for the reliability analysis of the DTV. As an example, Table 4 presents a list of nine selected concrete failure scenarios that have been derived. In Table 4 the five attributes *failure ID*, *component ID*, *fault*, *error* and *failure* are represented as columns headings. Failure scenarios are represented in rows. The failure ID (*FID*) does not imply a specific ordering but it is only used to identify the concrete failure scenarios. The column component ID (*CID*) includes acronyms of component names from Figure 1 to which the identifiers refer. Note that the separate features of the failure domain model are represented as keywords in the cells. For example, *Fault* includes the features *source*, *dimension* and *persistence* as defined in Figure 4. Such features are represented as keywords, which distinguish the fault, error and failure categories. For example, failure scenario F5 indicates a *permanent* fault (wrong implementation of a protocol). It leads to a *timing* error since a communicating party can not receive an expected response on time. This error turns out to be a failure that leads to a *transient* fault in F2 because the communication temporarily ceases. We consider *timing* errors as *irreversible* whereas a *wrong value*, for instance, can be rolled back (*reversible*). Apart from the different features, every column also includes the keyword *description*, which is used to denote the domain specific details of the concrete failure scenarios. Typically these descriptions are derived from domain experts.

6 Fault Tree Set and Severity Analysis

A close analysis of the failure scenarios in Table 4 shows that they are connected to each other. The columns *Fault* and *Failure* include the fields *source* and *target* respectively. These fields represent the propagation and the links between failure scenarios. For example, in failure scenario F2, the fault source is defined as CMR(F5), indicating that the fault in F2 occurs due to a failure in component CMR as defined in F5. The source of the fault can be caused by a combination of failures. This is expressed by logical connectives. For example, the source of F1 is defined as CH(F4) OR CMR(F6) indicating that F1 occurs due to a failure in component CH as defined in F4 or due to a failure in component CMR as defined in F6. To make all these connections explicit, in SARAH *fault trees* are defined. A fault tree is a model for representing the cause-effect relations of failures and faults. The root of a fault tree represents a failure and the leaf nodes represent faults. Since a failure can be logically caused by a set of faults, the nodes of the fault tree are interconnected with logical connectors.

Normally, a fault tree has one root (top) node, which represents the failure of the system. However, a system may fail in many different ways each of which has a different effect (i.e. annoyance caused) on the user. For example, a failure can be transient, permanent or catastrophic. A failure scenario can contribute to one or more user perceived failures. Hence, we define a *Fault Tree Set (FTS)* based on a given set

Table 4. Selected Failure Scenarios Derived for the DTV Architecture

FID	CID	Fault	Error	Failure
F1	AMR	<i>description:</i> Reception of irrelevant signals. <i>source:</i> CH(F4) OR CMR(F6) <i>dimension:</i> software <i>persistence:</i> transient	<i>description:</i> Working mode is changed when it is not desired. <i>type:</i> wrong path <i>detectability:</i> undetectable <i>reversibility:</i> reversible	<i>description:</i> Switching to an undesired mode. <i>type:</i> behavior <i>target:</i> user
F2	AMR	<i>description:</i> Can not acquire information. <i>source:</i> CMR(F5) <i>dimension:</i> software <i>persistence:</i> transient	<i>description:</i> Information can not be acquired from the connected device. <i>type:</i> too early/late <i>detectability:</i> detectable <i>reversibility:</i> irreversible	<i>description:</i> Can not provide information. <i>type:</i> timing <i>target:</i> CB(F3)
F3	CB	<i>description:</i> Can not acquire information. <i>source:</i> AMR(F2) <i>dimension:</i> software <i>persistence:</i> transient	<i>description:</i> Information can not be presented due to lack of information. <i>type:</i> too early/late <i>detectability:</i> detectable <i>reversibility:</i> irreversible	<i>description:</i> Can not present content of the connected device. <i>type:</i> behavior <i>target:</i> user
F4	CH	<i>description:</i> Software fault. <i>source:</i> internal <i>dimension:</i> software <i>persistence:</i> permanent	<i>description:</i> Signals are interpreted in a wrong way. <i>type:</i> wrong value <i>detectability:</i> undetectable <i>reversibility:</i> reversible	<i>description:</i> Provide irrelevant information. <i>type:</i> wrong value/presentation <i>target:</i> AMR(F1)
F5	CMR	<i>description:</i> Protocol mismatch. <i>source:</i> external <i>dimension:</i> software <i>persistence:</i> permanent	<i>description:</i> No communication with the connected device. <i>type:</i> too early/late <i>detectability:</i> detectable <i>reversibility:</i> irreversible	<i>description:</i> Can not provide information. <i>type:</i> timing <i>target:</i> AMR(F2)
F6	CMR	<i>description:</i> Software fault. <i>source:</i> internal <i>dimension:</i> software <i>persistence:</i> permanent	<i>description:</i> Signals are interpreted in a wrong way. <i>type:</i> wrong value <i>detectability:</i> undetectable <i>reversibility:</i> reversible	<i>description:</i> Provide irrelevant information. <i>type:</i> wrong value/presentation <i>target:</i> AMR(F1)
F7	DDI	<i>description:</i> Out-of-spec signals. <i>source:</i> external <i>dimension:</i> software <i>persistence:</i> transient	<i>description:</i> Scaling information can not be interpreted from meta-data. <i>type:</i> wrong value <i>detectability:</i> detectable <i>reversibility:</i> reversible	<i>description:</i> Can not provide data. <i>type:</i> wrong value/presentation <i>target:</i> IC(F8)
F8	IC	<i>description:</i> Inaccurate scaling ratio information. <i>source:</i> DDI(F7) AND VC(F9) <i>dimension:</i> software <i>persistence:</i> transient	<i>description:</i> Video image can not be scaled appropriately. <i>type:</i> wrong value <i>detectability:</i> undetectable <i>reversibility:</i> reversible	<i>description:</i> Provide distorted video image. <i>type:</i> presentation quality <i>target:</i> user
F9	VC	<i>description:</i> Software fault. <i>source:</i> internal <i>dimension:</i> software <i>persistence:</i> permanent	<i>description:</i> Wrong scaling ratio calculation. <i>type:</i> wrong value <i>detectability:</i> detectable <i>reversibility:</i> reversible	<i>description:</i> Provide inaccurate information. <i>type:</i> wrong value/presentation <i>target:</i> IC(F8)

of failure scenarios. Here, we introduce a set of definitions related to FTS, which will be further used in the remainder of the paper. FTS is a graph $G(V,E)$ with the following properties:

1. $V = F \cup A$
2. F is the set of failure scenarios each of which is associated with an architectural component.
3. $F_u \subseteq F$ is the set of failure scenarios comprising failures that are perceived by the user (i.e. system failures). Vertices residing in this set constitute root nodes of fault trees.
4. A is the set of gates representing the logical connectors.
5. $\forall g \in A,$
 $\text{outdegree}(g) = 1 \wedge$
 $\text{indegree}(g) \geq 1$
6. $A = A_{\text{AND}} \cup A_{\text{OR}}$ such that,
 A_{AND} is the set of AND gates,
 A_{OR} is the set of OR gates.
7. E is the set of directed edges (u, v)
 where $u, v \in V$.

For the example case, based on the failure scenarios in Table 4 we can derive the FTS as depicted in Figure 5.

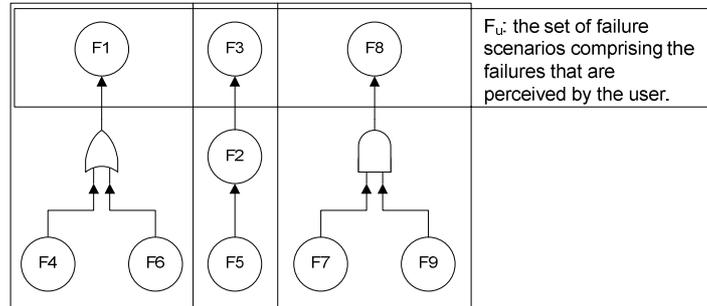


Fig. 5. Fault Trees Derived from the Failure Scenarios in Table 4

Here, the FTS consists of three fault trees. On the left the fault tree shows that F1 is caused by F4 or F6. The middle column indicates that failure scenario F3 is caused by F2 which is on its turn caused by F5. Finally, in the last column the fault tree shows that F8 is caused by both F7 and F9.

In conventional FMEA and FTA, every fault is assigned a severity value to denote how severe a fault is (e.g. faulty component can be repaired or not). In our model, we take a user-centric approach and define *severity* based on the user-perception. We assign severities to all failures based on their impact on the user. System failures that we consider are not restricted to complete crash-down of the system. For instance, a minor distortion in the brightness level of the video image and absence of any image

are both failures. However, the impact on the user perception for both failures is different. For example, a complete black screen will upset the user more than a temporary distortion in the image.

Severities based on user perception can be only defined for user perceived failures (elements of the set F_u). We need to propagate these severity values to the other failures (elements of the set F) as well. For this, we need to consider the impact of such intermediate failures to the user perceived failures. In conventional FTA, this impact is known as the *sensitivity* of the system failure (i.e. user perceived failure) with respect to a fault (i.e. intermediate failure). Sensitivity analysis is based on cut-sets in a fault tree and the probability values of fault occurrences [6]. However, this analysis leads to complex formulas and it requires that the probability values are known priori. We propose the following impact model by means of which one can reason about and estimate the overall impact of a failure even if the probability values are not known.

$$\begin{aligned}
 & root \in F_u, node \in F, \\
 & \forall n \in F \text{ s.t. } n \neq node \wedge n \neq root, P(n) = p, \\
 & P(node) = p', P(root) = f(p', p), \\
 & impact(node) = \int_0^1 \left(\frac{\partial}{\partial p'} P(root) \right) dp
 \end{aligned} \tag{1}$$

In equation (1) above, the impact calculation of an intermediate failure (*node*) on a user-perceived failure (*root*) is presented. The probability of occurrence of *root* ($P(root)$) is represented in terms of the occurrence probabilities of all other nodes as it is done in FTA [6]. For example, the following probabilities can be defined for user-perceived failures considering the fault trees presented in Figure 5: $P(F1) = P(F4) + P(F6) - (P(F4) \times P(F6))$, $P(F3) = P(F2) = P(F5)$, $P(F8) = P(F7) \times P(F9)$. We assign p' to the probability of occurrence of *node* and fix the probability values of all other nodes to p in $P(root)$. Thus $P(root)$ turns out to be a function of p and p' . Recalling the example in Figure 6 again, if we are interested in impact of F4 on F1, we transform $P(F1)$ to $P(F1) = p' + p - (p' \times p)$. Then, we take a partial derivation of $P(root)$ with respect to p' . This gives the rate of change of $P(root)$ with respect to p' . For our example, this will yield to $\partial/\partial p' P(F1) = 1 - p$. Finally, the result of the partial derivation is integrated with respect to p for all possible probability values ([0-1]) to calculate the overall impact. For the example case, $\int (\partial/\partial p' P(F1)) dp = \int (1 - p) dp = p - p^2/2$. So, the result of the integration from 0 to 1 will be 0,5. When the probability values are known, they can be included in the model instead of fixing them to p and ranging them from 0 to 1. The basic shortcoming of this model is the equality assumption for the probability values that are fixed to p . There exist similar approaches in the literature where sensitivity/impact analyses are applied by changing a parameter one at a time and fixing the others [7, 21]. However, such analyses are applied for a specific architecture and the associated fault tree without providing a generic model. In those studies, basically the parameters are varied and results obtained from the reliability model are observed to assess the sensitivity.

Combining the impact model introduced above together with the severity degrees assigned to the user perceived failures leads us to the following severity model, which assigns severity to each node of a fault tree.

$$\begin{aligned}
 & root \in F_u, node \in F, \\
 & \forall n \in F \text{ s.t. } n \neq node \wedge n \neq root, P(n) = p, \\
 & P(node) = p', \forall root \in F_u : P(root) = f(p', p), \\
 & severity(node) = \sum_{root \in F_u} severity(root) \times \int_0^1 \left(\frac{\partial}{\partial p'} P(root) \right) dp
 \end{aligned} \tag{2}$$

In equation (2), $severity(root)$ gives the severity of a user-perceived failure. This can be based on a set of values in ordinal scale that defines user annoyance levels or it can be based on a more sophisticated model. The severity of other failure ($severity(node)$) is calculated based on the impact model introduced in equation (1) and the severity values assigned to the user perceived failures they lead to. As a result, the severity of a failure depends on the type of failures it leads to and to how much it contributes (the impact) to these failures. Note in equation (2) that if there is no connection between $node$ and a $root \in F_u$ in the fault tree, then the result of the partial derivation and hence the whole severity calculation will yield to 0 (since there will be no term in the function with p' as the multiplier).

For the analysis presented in this paper, we adopt a simpler model for propagation of severity values. As a diversion from the usual approach, we process the fault trees in a top-down manner to assign severities to intermediate failures based on severities of user-perceived failures. Given the FTS, the calculation of severities of failure scenarios can be defined as follows.

$$\begin{aligned}
 & \forall f \in F_u, s(f) = s_u \text{ s.t. } 1 \leq s_u \leq 5 \\
 & \forall f \in F \wedge f \notin F_u, \\
 & s(f) = \sum_{\substack{\forall v \text{ s.t.} \\ (u,v) \in E \wedge \\ (v \in F \vee v \in A_{OR})}} s(v) + \sum_{\substack{\forall v \text{ s.t.} \\ (u,v) \in E \wedge \\ v \in A_{AND}}} \frac{s(v)}{INDEGREE(v)}
 \end{aligned} \tag{3}$$

In the first part of the equation, we take into account the failure scenarios with the target *user*. A severity value (s_u) is assigned for each such failure based on their severities with respect to the user-perception. These are the failures F1, F3 and F8. We apply a prioritization of the failure scenarios based on the severity values that range from 1 to 5 (See Table 5).

Table 5. The Severity Levels for the Prioritization of Failure Scenarios

Severity	Type	Annoyance Description
1.	Very low	User hardly perceives failure.
2.	low	A failure is perceived but not really annoying.
3.	moderate	Annoying performance degradation is perceived.
4.	high	User perceives serious loss of functions.
5.	very high	Basic functions fail. System locks up and does not respond.

The severity values for failures F1, F3 and F8 are depicted in Figure 6. These values are then used in order to determine the severities of other failures as shown in the second part of the equation (3). If a failure directly leads to another one or if it is connected through an OR gate in which some other failures may exclusively cause the same failure also, we add the severity of the resulting failure to the severity of failure under consideration. If there is a connection through an AND gate where a combination of failures results in another one, we share the severity value of the resulting failure among contributing failures. In other words, we add the severity of resulting failure divided by the number of contributing failures to the severity of each contributing failure. For example, since F1 has the assigned severity value of 3, this is also assigned to the failures F6 or F8 that are connected through an OR logic to F1. In case of F7 and F9, the severity value is $4/2$ because F8 has the severity value of 4 and it has an AND gate with 2 failure scenarios connected to it. A failure scenario can be connected to multiple gates and other failures in which the severity is derived as the sum of severities calculated for all these connections.

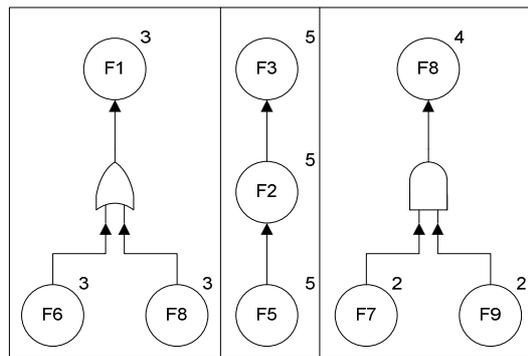


Fig. 6. Fault Trees with the Severity Values

7 Analysis of the Software Architecture

In our example case, we defined a total of 44 failure scenarios including the scenarios presented in Table 4. We completed the definition process by deriving the corresponding fault tree set and calculating severity values as explained in Section 6. The results that were obtained during the definition process are utilized by the analysis process as described in the subsequent sub-sections.

7.1 Architecture-Level Analysis

The first step of the analysis process is the architecture-level analysis in which we pinpoint the sensitive points of the architecture with respect to reliability. As a primary and straightforward means of comparison, we consider the percentage of failures (PF) that are associated with the components. For each component c the value for PF is calculated as follows.

$$PF_c = \frac{\#of\ failures\ associated\ with\ c}{\#of\ failures} \times 100 \tag{4}$$

This means that simply the number of failures related to a component is divided by the total number of failures (in this case 44). The results are shown in Figure 7. A first analysis of this figure already shows that the *Application Manager (AMR)* and *Teletext (TXT)* components have to cope with a higher number of failures than the other components.

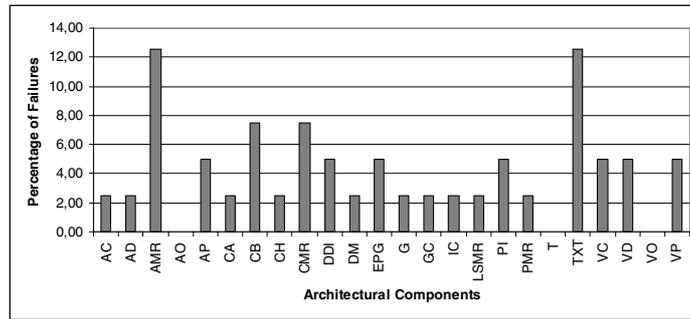


Fig. 7. Percentage of the Failure Scenarios Impacting the Components

This analysis treats all failures equally. To take the severities of the failures into account we define the Weighted Percentage of Failures (WPF) as the following.

$$WPF_c = \frac{\sum_{\forall u \in F \text{ s.t. } comp(u)=c} s(u)}{\sum_{\forall u \in F} s(u)} \times 100 \tag{5}$$

For each component, we collect the set of failures associated with it and we add up their severity values. After averaging this value with respect to the all failures and calculating the percentage, we obtain the WPF value. The result of the analysis is shown in Figure 8.

The weighted percentage presents different results compared to the previous one. Two components can be associated with the same amount of failures but these failures can have different severities. For example, AP and EPG have the same values in Figure 7 but EPG has a greater value than AP in Figure 8. Nevertheless, the *Application Manager (AMR)* and *Teletext(TXT)* components again appears to be the most critical.

From the project perspective it is not always possible to focus on the total set of possible failures due to the time constraints and the cost of fault tolerance. To optimize the cost usually one would like to consider the failures that have the largest impact on the system. For this, in SARAH the architectural components are ordered in

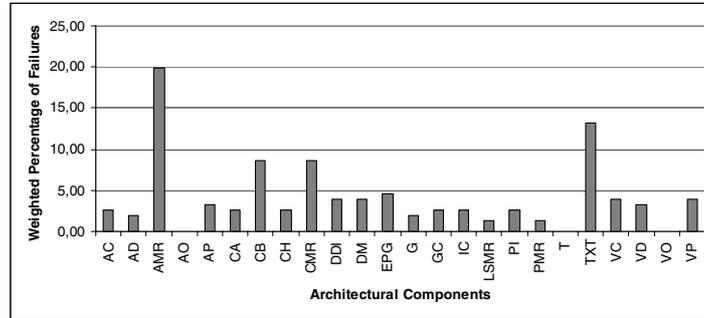


Fig. 8. Percentage of the Failure Scenarios Impacting the Components

ascending order with respect to their WPF values. The components are then categorized based on the proximity of their WPF values. Accordingly, components of the same group have WPF values that are close to each other. The results of this prioritization and grouping are provided in Table 6, which also shows the sum of the WPF values of the components for each group. Here we can see that, for example, group 4 consists of two components AMR and TXT. The reason for this is that their WPF values are the highest and close to each other. The sum of their WPF values is $20+13=33\%$.

Table 6. Components Grouped Based on the WPF Values

Group #	Components	WPF
1	AC, CA, CH, GC, IC, PI, AD, G, LSMR, PMR, AO, T, VO	23%
2	EPG, VC, VD, DDI, DM, VD, AP	27%
3	CB, CMR	17%
4	AMR, TXT	33%

To highlight the difference in impact of the component groups we define a Pareto chart as presented in Figure 9.

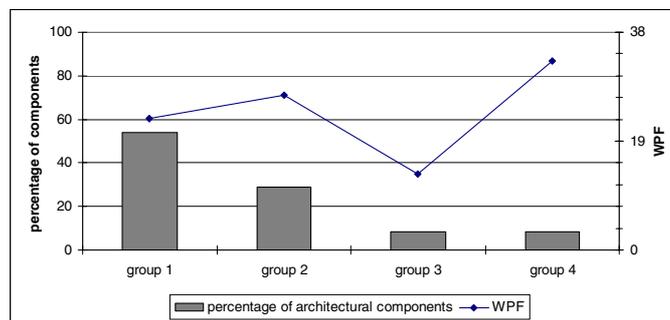


Fig. 9. Pareto Chart Showing the Largest Impact of the Smallest Set of Components

In the Pareto chart, the component groups shown in Table 6 are ordered along the x-axis with respect to the number of components they include. The percentage of components that each group includes is depicted with bars. The y-axis on the left hand shows the percentage values from 0 to 100 and is used for scaling the percentages of the architectural components whereas the y-axis on the right hand side scales the WPF values. The plotted line represents the WPF value for each group. In the figure we can, for example, see that group 4 (consisting of two components) represents about 8% of the components but has a WPF of 33%. Here, we group the components according to their WPF values and assign importance to these components according to the amount and severities of failures they are associated with (not according to their functionality or other aspects). The components with the highest WPF values are considered to be the most important ones to focus on.

7.2 Component-Level Analysis

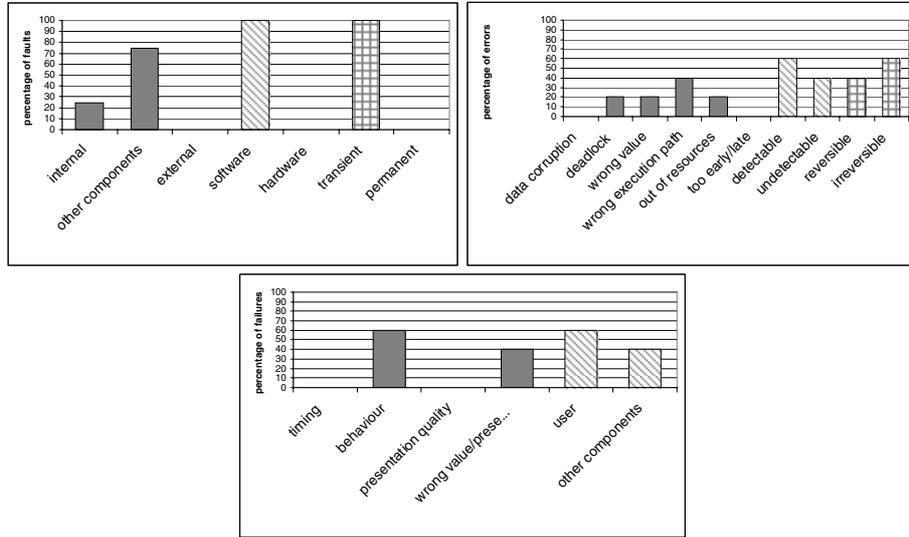
The architectural level analysis provides only a quantitative analysis of the impact of failure scenarios on the given architecture. However, for failure management and recovery it is also necessary to define the *type* of failures that might occur in the identified sensitive components. This is analyzed in the component-level analysis in which the features of faults, errors and failures that impact the component are determined. For the example case, in the architectural-level analysis it appeared that components residing in the 4th group (see Table 6) had to deal with largest set of failure scenarios. Therefore, in the component-level analysis, we will focus on the members of this group, namely *Application Manager* and *Teletext* components.

Following the derivation of the set of failure scenarios impacting the component, we group them in accordance with the features presented in Figure 4. This grouping results in the distribution of fault, error and failure categories of failure scenarios associated with the component.

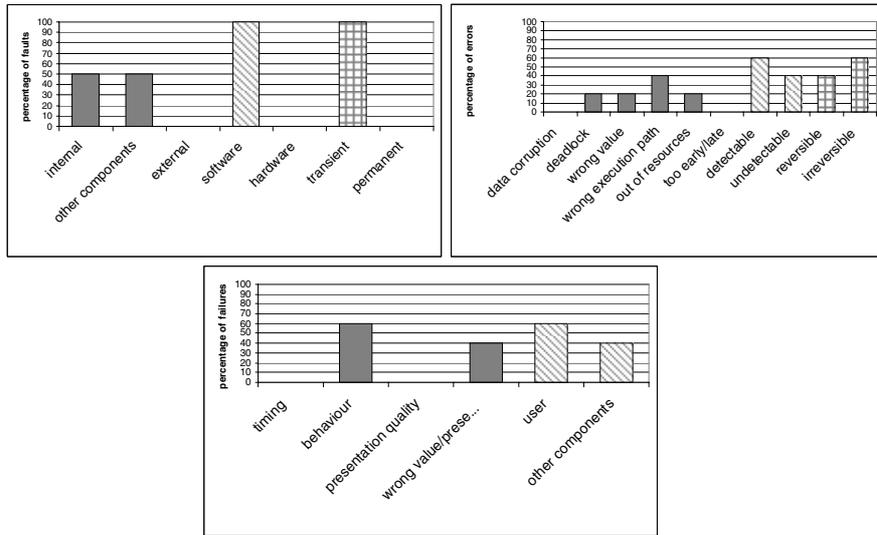
For example, the results obtained for the *Application Manager* and *Teletext* components are shown in Figure 10(a) and Figure 10(b), respectively. If we take a look at the fault features presented on those figures for instance, we see that most of the faults impacting the *Application Manager* Component are caused by the other components. On the other hand, *Teletext* Component has internal faults as much as faults stemming from the other components. As such, distribution of features reveals characteristics of faults, errors and failures associated with the individual components of the architecture.

7.3 Failure Analysis Report

SARAH defines a detailed description of the fault tree set, the failure scenarios, the architectural level analysis and the component level analysis. These are described in the *failure analysis report* that summarizes the previous analysis results and provides hints for recovery. Sections comprised by the failure analysis report are listed in Table 7, which are in accordance with the steps of SARAH.



(a) Fault, Error and Failure Features of the Failure Scenarios Associated with the *Application Manager* Component



(b) Fault, Error and Failure Features of the Failure Scenarios Associated with the *Teletext* Component

Fig. 10. Fault, Error and Failure Features of the Failure Scenarios Associated with the Components in the 4th Group of the Pareto Chart

Table 7. Sections of the Failure Analysis Report

Section #	Heading
1	Introduction
2	Software Architecture
3	Failure Domain Model
4	Failure Scenarios
5	Fault Tree Set
6	Architecture Level Analysis
7	Component Level Analysis

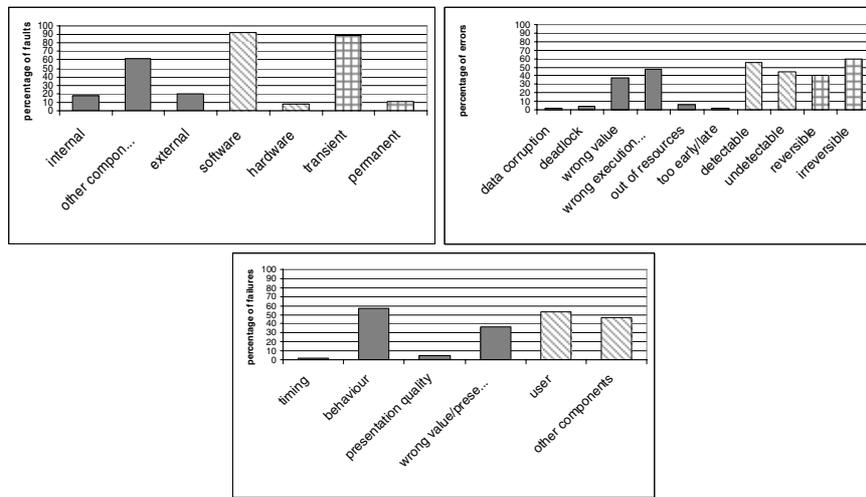


Fig. 11. Feature Distribution of Fault, Error and Failures for all Failure Scenarios

The first section describes the project context, information sources and specific considerations (e.g. cost-effectiveness). The second section describes the software architecture that is used as an input for the analysis. The third section presents the failure domain model including the fault, error and failure features that are of interest to the project. The fourth section contains list of failure scenarios annotated based on this domain model. The fifth section depicts the fault tree set generated from the failure scenarios together with the severities assigned to each. The sixth and seventh sections include analysis results as presented in sections 7.1 and 7.2 of this paper. Additionally, the sixth section includes the distribution of fault, error and failure features for all failure scenarios as depicted in Figure 11.

8 Related Work

FMEA has been widely used in various industries such as automotive and aerospace. It has also been extended to make it applicable to the other domains or to achieve more analysis results. For instance, Failure Modes, Effects and Criticality Analysis

(FMECA) is a well-known method that is built upon FMEA. Additionally, FMECA [12] incorporates severity and probability assessments for faults. The probabilities of occurrence of faults (assumed to be known) are utilized together with the fault tree in order to calculate the probability that the system will fail. On the other hand, a severity is associated with every fault to distinguish them based on the cost of repair. Note that the severity definition in our method is different from the one that is employed by FMECA. We take a user-centric approach and define the severities for failures based on their impact on the user.

Almost all reliability analysis techniques have primarily devised to analyze failures in hardware components. These techniques have been extended and adapted to be used for the analysis of software systems. For instance, in his book [15], Redmill shows how Hazard and Operability Studies (HAZOP) can be applied to computer-based systems. Application of FMEA to software has a long history [16]. Both FMEA and FTA have been employed for the analysis of software systems and named as Software Failure Modes and Effects Analysis (SFMEA) and Software Fault Tree Analysis (SFTA), respectively. In SFMEA, failure modes for software components are identified such as computational, logic and data I/O. This classification resembles the failure domain model of SARAH. However, SARAH separates fault, error and failure concepts and provides a more detailed categorization for each. Also, note that the failure domain model can vary depending on the project requirements and the system. In [11], SFTA is used for the safety verification of software. However, the analysis is applied at the source code level rather than the software architecture design level. It utilizes a set of failure-mode templates that outlines the failure modes of programming language elements like assignments and conditional statements. These templates are composed according to the control flow of the program to derive a fault tree for the whole software.

In general, efforts for applying reliability analysis to software [10] mainly focus on the safety-critical systems, whose failure may have very serious consequences such as loss of human life and large-scale environmental damage. In our case, we focus on consumer electronics domain, where the systems are usually not safety-critical. FMEA has been used in other domains as well, where the methodology is adapted and extended accordingly. To use FMEA for analyzing the dependability of Web Services, new failure taxonomy, intrusions and various failure effects (data loss, financial loss, denial of service, etc.) are taken into account in [9]. Utilization of FMEA is also proposed in [22] for early robustness analysis of Web-based systems. The method is applied together with the Jacobson's method [18], which identifies three types of objects in a system: 1) boundary objects that communicate with actors in a use-case, 2) entity objects that are objects from the domain and 3) control objects that serve as a glue between boundary objects and entity objects. In our method, we do not presume any specific decomposition of the software architecture and we do not categorize objects or components. However, we categorize failure scenarios based on the failure domain model and each failure scenario is associated with a component.

Jacobson's classification [18] is aligned with our failure domain model with respect to the propagation of failures (`fault.source`, `failure.target`). The target feature of failure, for instance, can be the user (i.e. actor) or the other components. In [20], a modular representation called Fault Propagation and Transformation Calculus (FPTC) is introduced. FPTC is used to specify the failure behavior of each component (i.e.

how a component introduces or transforms a failure type). This facilitates the automatic derivation of the propagation of the failures throughout the system. In our method, we represent the propagation of failure scenarios with fault trees. The semantics of the transformation is captured in the “type” tags of failure scenarios.

In this work, we made use of spreadsheets that define the failure scenarios and automatically calculate the severity values in the fault trees (after initial assignment of the user-perceived failure severities). This is a straightforward calculation and as such we have not elaborated on this issue. On the other hand, there is a body of work focusing on tool-support for FMEA and FTA. In [9], for example, FMEA tables are being integrated with a web service deployment architecture so that they can be dynamically updated by the system. In [13], fault trees are synthesized automatically. Further, multiple failures are taken into account, where a failure mode can contribute to more than one system failure. The result of the fault tree synthesis is a network of interconnected fault trees, which is analogous to the fault tree set in our method.

An advanced Failure Modes and Effect Analysis (AFMEA) is introduced in [8], which also focuses on the analysis at the early stages of design. However, the aim of this work is to enhance FMEA in terms of the number and range of failure modes captured. This is achieved by constructing a behavior model for the system.

9 Conclusion

FMEA and FTA are well known techniques that have been successfully applied in various domains like automotive and aerospace for reviewing the causes and effects of system failures systematically. In this paper we have utilized these techniques for early reliability analysis of software architectures. It appears that the techniques can not be directly used as-is for analyzing reliability of software earlier in the life cycle and as such need to be extended.

For the FMEA, we have utilized scenarios as the basic means to analyze the software architecture [5]. We have separated fault, error and failure concepts in a failure scenario as defined in the software reliability engineering domain [2]. We have provided a systematic means for deriving failure scenarios based on a failure domain model. For the analysis presented in this paper, we have also derived the fault, error and failure categorizations through domain analysis techniques [1].

For the FTA, we have introduced a quantitative impact model based on fault trees to estimate the impact of faults on the occurrence of a system failure. This model can be used to reason about how sensitive the system reliability is with respect to occurrence of a fault, even if the occurrence rates of faults are not known. Furthermore, we have distinguished and prioritize failures based on their effect (i.e. annoyance caused) on the user. We have incorporated this measure to our severity calculation.

We have integrated the extended FMEA and FTA techniques with our software architecture reliability analysis method (SARAH) and illustrated their application for analyzing the reliability of the software architecture of a Digital TV. In our future work we will experiment with the method to identify the potential failures of embedded systems and as such improve their reliability.

Acknowledgments. We thank all members of the TRADER project, for their feedback on this work and their input about the TV domain knowledge and reliability issues. Particularly, we thank Rob Golsteijn from NXP Semiconductors, Paul L. Janson from Philips Research, Iulian Nitescu from Philips TASS for their contribution in deriving the conceptual architecture of DTV and possible failures. We also specially thank Christian Hofmann from University of Twente, Teun Hendriks and Jozef Hooman from ESI, the editors and anonymous reviewers for reviewing and providing useful feedback to this paper.

References

1. Arrango, G.: Domain Analysis Methods. In: Schafer, R., Prieto-Diaz, R., Matsumoto, M. (eds.) *Software Reusability*, pp. 17–49. Ellis Horwood, New York (1994)
2. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. on Dependable and Secure Computing* 1(1), 11–33 (2004)
3. Bachman, F., Bass, L., Klein, M.: *Deriving Architectural Tactics: A Step Toward Methodical Architectural Design*. CMU/SEI-2003-TR-004, Pittsburgh, PA (2003)
4. Clements, P., Bachman, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: *Documenting Software Architectures*. Addison-Wesley, Reading (2002)
5. Dobrica, L., Niemela, E.: A Survey on Software Architecture Analysis Methods. *IEEE Trans. on Software Engineering* 28(7), 638–654 (2002)
6. Dugan, J.B.: Software System Analysis Using Fault Trees. In: Lyu, M.R. (ed.) *Handbook of Software Reliability Engineering*, ch. 15, pp. 615–659. McGraw-Hill, New York (1996)
7. Dugan, J.B., Lyu, M.R.: Dependability Modeling for Fault-Tolerant Software and Systems. In: Lyu, M.R. (ed.) *Software Fault Tolerance*, ch. 5, pp. 109–138. John Wiley & Sons, New York (1995)
8. Eubanks, C.F., Kmenta, S., Ishil, K.: Advanced Failure Modes and Effects Analysis using Behavior Modeling. In: *Proceedings of the ASME Design Theory and Methodology Conference*, New York (1997)
9. Gorbenko, A., Kharchenko, V., Tarasyuk, O.: FMEA- technique of Web Services Analysis and Dependability Ensuring. In: Butler, M., Jones, C., Romanovsky, A., Troubitsyna, E. (eds.) *Rigorous Development of Complex Fault-Tolerant Systems*, LNCS, pp. 153–168. Springer, Heidelberg (2006)
10. Isaksen, U., Bowen, J.P., Nissanke, N.: *System and Software Safety in Critical Systems*. Technical Report RUCS/97/TR/062/A, The University of Reading, UK (1997)
11. Leveson, N.G., Cha, S.S., Shimeall, T.J.: Safety Verification of Ada Programs using Software Fault Trees. *IEEE Software* 8(4), 48–59 (1991)
12. MIL-STD-1629A: Procedures for Performing a Failure Modes, Effects and Criticality Analysis. Department of Defense, Washington, DC (1980)
13. Papadopoulos, Y., Parker, D., Grante, C.: Automating the Failure Modes and Effects Analysis of Safety Critical Systems. In: *Proceedings of HASE'04, FL*, pp. 310–311 (2004)
14. Redmill, F.: Exploring Subjectivity in Hazard Analysis. *Engineering Management Journal (IEE)* 12(3) (2002)
15. Redmill, F., Chudleigh, M., Catmur, J.: *System Safety: HAZOP and Software HAZOP*. John Wiley & Sons Ltd, Chichester (1999)
16. Reifer, D.J.: Software Failure Modes and Effects Analysis. *IEEE Transactions on Reliability* R-28(3), 247–249 (1979)

17. Roland, E., Moriarty, B.: Failure Mode and Effects Analysis. In: System Safety Engineering and Management, 2nd edn. John Wiley & Sons, Chichester (1990)
18. Rosenberg, D., Scott, K.: Use Case Driven Object Modeling with UML: A Practical Approach. Addison-Wesley, Reading (1999)
19. Trader project web site (2006), <http://www.esi.nl/site/projects/trader>
20. Wallace, M.: Modular Architectural Representation and Analysis of Fault Propagation and Transformation. In: Proceedings of FESCA, ENTCS vol. 141(3) (2005)
21. Yakoub, S., Cukic, B., Ammar, H.: Scenario-based Reliability Analysis of Component Based Software. In: Proceedings of ISSRE'99, Boca Raton, FL, pp. 22–31 (1999)
22. Zhou, J., Stalhane, T.: Using FMEA for early robustness analysis of Web-based systems. In: Proceedings of COMPSAC'04, Washington, DC, pp. 28–29 (2004)