

# Automated Failure-modes-and-effects Analysis of Embedded Software

Extended Abstract

Peter Struss  
Comp. Sci. Dept.  
Tech. Univ. of Munich  
Garching, Germany  
struss@in.tum.de

**Abstract**—The paper presents work in progress aiming at extending Failure-modes-and-effects Analysis (FMEA) to include embedded software. It is based on the insight that requirements on Software FMEA in this context are only dependent on the required functionality of the physical system, which, in particular, allows the consideration of a finite set of qualitatively characterized software failures.

**Keywords:** software quality, Failure-modes-and-effects Analysis, embedded software

## I. INTRODUCTION

Failure-modes-and-effects analysis (FMEA) of physical systems is a well-established process and subject to standardization, esp. in the aeronautics and automotive industries. The core of the task is inferring the impact of classes of failures of components and subsystems on the functionality of the entire system. More specifically, this impact is described in terms of a set of specified effects, which represent different types of violations of the intended function.

FMEA is performed as a worst-case analysis, based on characterizing both failures and effects in a purely qualitative, rather than numerical, way (e.g. “leakage”, “reduced pressure generation”). Current practice is mainly confined to the physical components of the system, despite the fact that the considered systems and subsystems are mechatronic systems, whose behavior is strongly influenced, or even determined, by embedded software.

Based on our work on automated model-based FMEA of physical systems [1], [2], we propose an integrated solution to FMEA of mechatronic systems and argue that, in this context, software FMEA becomes feasible. This is due to the qualitative nature of the analysis and the fact that the (mis)behavior of the physical system alone determines the goal and granularity of the analysis and, especially, a finite set of failure models. It also restricts the input combinations for which the software behavior needs to be analyzed.

We sketch the foundations of model-based FMEA and the models used and discuss the extension to software, using an illustrative example of a braking system.

## II. AUTOMATED MODEL-BASED FMEA

Besides different FMEA tools, which mainly provide facilities for editing, storing, and retrieving the content of FMEA tables or supporting constructs, such as fault trees, there exists successful work on automating the key inference of FMEA.

To support FMEA, it is necessary to determine whether the effects of a certain component fault violate an intended function of the system. If the function is considered as part of GOALS, then the task might mean to check whether the fault model  $FM_1$  is inconsistent with the function:

$$FM_1 \cup GOALS \vdash ? \perp$$

Often, the analysis is carried out for particular mission phases (such and “cruising” or “landing” of an aircraft) or scenario  $S_k$ :

$$FM_1 \cup S_k \cup GOALS \vdash ? \perp$$

In practice, FMEA is not carried out this way, but by specifying effects  $E_i$ , which are specific violations of the intended function (GOALS), for instance too high and too low deceleration of a wheel, i.e. underbraking and overbraking:

$$S_k \cup E_i \vdash \neg GOALS,$$

and the analysis determines the effects that may occur under a particular failure mode:

$$FM_1 \cup S_k \cup E_i \not\vdash \perp.$$

This solution can be and has to be based on a compositional and qualitative model.

- **Compositionality** is a prerequisite for cheap automatic configuration of a system model with an injected fault.
- An abstract and **qualitative** level of **modeling** is mandatory for inferring the associations between classes of faults and classes of effects, with both cover ranges of rather numerically specified instances. Furthermore, re-usability of the models increases, since irrelevant distinctions vanish at the qualitative level.

A behavior model is regarded as a relation R over a set of variables that characterize a component or a system:

$R \subset \text{DOM}(\underline{v})$ , where  $\underline{v}$  is a vector of system variables with the domain  $\text{DOM}(\underline{v})$ , which is the Cartesian product

$$\text{DOM}(\underline{v}) = \text{DOM}(v_1) \times \text{DOM}(v_2) \times \dots \times \text{DOM}(v_n).$$

So, a relation  $R$  (i.e. a *constraint*) is a subset of the possible behavior space.

If elementary model fragments  $R_{ij}$  are related to behavior modes  $\text{mode}_i(C_j)$  of the component  $C_j$ , then an aggregate system (under correct or faulty conditions) is defined by a mode assignment  $MA = \{\text{mode}_i(C_j)\}$  which specifies a unique behavior mode for each component of this aggregate, whose model is obtained as the join of the mode models, i.e. the result of applying a (complete version of) constraint satisfaction to  $\{R_{ij}\}$ :

$$R_{MA} = \bowtie R_{ij}.$$

Since models, scenarios, and effects can all be represented by relations, we can characterize and compute the effects of the  $FM_1$  as follows:

- $FM_1 \bowtie S_k \subset E_1$   
if the failure mode is included in effect, then the effect will definitely occur (case  $E_1$  in Figure 1)
- $FM_1 \bowtie S_k \cap E_2 = \emptyset$   
if the intersection is empty, the effect does not occur (case  $E_2$ )
- *otherwise*  
the effect may occur:  $E_3$

### III. DEVIATION MODELS

FMEA is about inferring deviations from nominal system function from a deviation of nominal component behavior. Hence, not the magnitude of certain quantities matters, but the fact whether or not they deviate from what is expected under normal or safe behavior.

This is why deviation models [2] offer the basis for a solution: they express constraints on the deviations of system variables and parameters from the nominal behavior and capture how they are propagated through the system.

For each system variable and parameter  $v_i$ , the deviation is defined as the difference between the actual and a reference value:

$$\Delta v := v_{\text{act}} - v_{\text{ref}}$$

Then algebraic expressions in an equation can be transformed to deviation models according to rules such as

$$\begin{aligned} a + b = c &\Rightarrow \Delta a + \Delta b = \Delta c \\ a * b = c &\Rightarrow a_{\text{act}} * \Delta b + b_{\text{act}} * \Delta a - \Delta a * \Delta b = \Delta c \end{aligned}$$

Furthermore, for any monotonically growing (section of  $a$ ) function  $y = f(x)$ , we obtain  $\Delta y = \Delta x$  as an element of a qualitative deviation model.

For instance, the deviation model of a valve is given by a constraint:

$$\Delta Q = A * (\Delta P_1 - \Delta P_2) + \Delta A * (P_1 - P_2) - \Delta A * (\Delta P_1 - \Delta P_2)$$

on the signs of the deviations of pressure ( $\Delta P_i$ ), flow ( $\Delta Q$ ), and area ( $\Delta A$ ). This constraint allows, for instance, to infer that an increase in  $P_1$  ( $\Delta P_1 = +$ ) will lead to an increase in the flow ( $\Delta Q = +$ ), if  $P_2$  and the area remain unchanged ( $\Delta P_2 = 0, \Delta A = 0$ ) and the valve is not closed ( $A = +$ ). Such qualitative deviation models can be constructed from equational component models, if they exist.

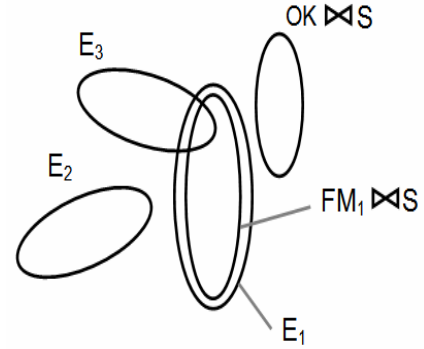


Figure 1 Computation of effects

### IV. EXAMPLE: BRAKING SYSTEM

To illustrate the approach, we use an example from one of our recent case studies [1]. A standard braking system is split into two diagonal wheel pairs, one of them shown in Figure 2.

The pedal actuation block (to the right of JointT2\_1, not shown in the figure) is directly affected by pushing the brake pedal. The inlet-valves (M\_VI11, 12, 21, 22) behave as piloted check valves; during standard braking (i.e. with no command), they are open, while the outlet-valves (M\_VO11, 12, 21, 22) are closed if no command is present. This way, pushing the brake pedal causes pressure to build up in the wheel brakes.

When operated under the Anti-lock-braking system (ABS), the valves are controlled by commands from the electronic control unit (ECU). The pressure-build-up phase is the scenario described above. For pressure maintenance, the inlet valve is closed. If the speed sensors indicate that the wheels tend to lock up, the outlet valves are opened to release pressure, let the wheels spin again and, thus, enable steering of the vehicle. Then the cycle is entered again.

Typical inferences required for FMEA of the brake is moving would be

- If an inlet valve is stuck closed under normal braking, the respective wheel will be underbraked (reduced deceleration).
- If an outlet valve is stuck closed in the pressure release phase, the respective wheel will be overbraked.

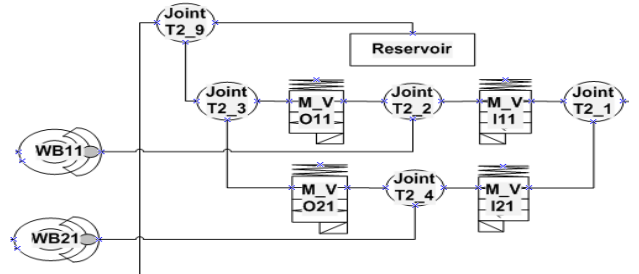


Figure 2 - Braking system (for one pair of wheels). Pressure generated by the pedal reaches the wheel brakes, WB<sub>ij</sub>, via open inlet valves, M\_VI<sub>ij</sub>, while outflow is blocked by closed outlet valves, M\_VO<sub>ij</sub>.

The deviation model of the inlet valve predicts  $\Delta Q = -$  from  $\Delta A = -$ . The model of the wheel brake yields a reduced pressure and reduced deceleration and positive deviation of the wheel speed, i.e. underbraking. [1] reports the results of the case study, which produced a complete FMEA table for the system.

#### V. INTEGRATING FMEA OF EMBEDDED SOFTWARE

FMEA **needs** to be extended to **include embedded software** as a component for several reasons:

- Control software may **modify the impact of failures** of physical components and even mask these failures.
- **Sensor faults**, a common source of disturbances, cannot be analyzed without considering software, as the medium that transforms a (wrong) sensor signal into an actuator signal, thus determining the impact of the sensor fault on the behavior of the physical system.
- The **interaction** of several **subsystems** may happen at a physical level, but also through communication between their embedded software modules.
- Of course, **software malfunctions** can be the origin of improper system performance.

Automated FMEA of embedded software is made feasible in the context of FMEA of physical systems:

- The **only criterion** for assessing the performance of the embedded software and the importance of software failures is the desired behavior of the **physical system**. There is no intrinsic quality criterion of the software itself. In other words, the effects already defined for the FMEA of the physical components are also the touchstones for FMEA of the embedded software and provide a focus for the modeling and analysis of the software.
- The context of the **physical system confines the analysis** of the software: the combinatorics of the theoretical input space is reduced to the set of physically possible inputs.
- This space is further reduced due to the **qualitative nature of the analysis**. Hence, the required models of the software functions can and need to be stated with the same coarse granularity of the models of physical components.

More specifically, models of embedded software have to be stated in terms of relations of sensor and actuator signals, including, for nominal software behavior, how deviating sensor signals are transformed into (deviations of) actuator signals (or their timing) and how relevant software faults result in deviating actuator signals.

The brake system can be used to illustrate this. In order to investigate the impact of a failure of a sensor that measures the rotational speed of a wheel we need a model of the intended behavior of the ECU, more precisely the software functions that control the valves based on the measured wheel speed: it has to issue a command,  $cmd=I$ , when the wheel speed drops below a certain threshold. The command

causes an inlet valve to close and an outlet valve to open (for different thresholds). In our context, the only interesting aspect is how the function propagates a deviation of a sensor value (or a missing one), if it works correctly.

Slightly simplified, this can be stated as

$$\Delta cmd = -\Delta v_s,$$

where  $v_s$  is the sensor signal and  $\Delta cmd$  is defined on the domain  $\{0, 1\}$  of  $cmd$ . If the  $v_s$  is too low (high), i.e. deviates negatively (positively) and, hence, reaches the threshold too early (too late), this causes the command to be set too early (too late), i.e. deviate positively (negatively). The (OK) model of the inlet valve contains

$$\Delta A = -\Delta cmd,$$

while the outlet valve includes

$$\Delta A = \Delta cmd.$$

Hence, the impact of the sensor failure will be the same as for the respective valve failures, in particular overbraking and underbraking.

The relevant failures of the software itself are

- untimely command (which includes command too early, e.g. due to a high threshold value, and command always):  $\Delta cmd = +$  and
  - missing command (too late or never):  $\Delta cmd = -$ ,
- triggering the same effects as above.

The example supports a number of insights:

- Software functions can be modeled in the same way as physical components, in particular
- qualitatively and
- by deviation models.
- The abstract level of the required analysis and the finite number of relevant effects yields a finite space of software failures.
- Re-usable models of software functions (and model libraries for compositional modeling) seem possible: the described model cannot only be used for the eight functions that control the valves. Sending a command or switching a mode based on some threshold for a sensor (or computed) value is quite common.
- Also, at least some aspects of communication between ECUs and its (hardware and software) failures can be modeled in similar way (timeouts, missing or wrong signals, etc.).

[1] P. Struss, A. Fraracci: FMEA of a Braking System - A Kingdom for a Qualitative Valve Model!. To appear in: To appear in: 25th International Workshop on Qualitative Reasoning, Barcelona, Spain, 2011

[2] [Picardi *et al.*, 2004] C. Picardi, L. Console, F. Berger, J. Breeman, T. Kanakis, J. Moelands, S. Collas, E. Arbaretier, N. De Domenico, E. Girardelli, O. Dressler, P. Struss, B. Zilbermann. *AUTAS: a tool for supporting FMECA generation in aeronautic systems*. In proceeding of the 16th European Conference on Artificial Intelligence. August 22nd - 27th 2004 Valencia, Spain, pp. 750-754

[3] Struss, P. *Models of Behavior Deviations in Model-based Systems*. In Proceeding of the 16th European Conference on Artificial Intelligence. August 22nd - 27th 2004 Valencia, Spain, pp. 883-887