Applied Soft Computing xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Identifying critical architectural components with spectral analysis of fault trees

Tolga Ayav[a,*], Hasan Sözer[b]

[a] Izmir Institute of Technology, İzmir, Turkey
[b] Ozyegin University, İstanbul, Turkey

ARTICLE INFO

Article history:
Received 10 December 2015
Received in revised form 22 June 2016
Accepted 30 June 2016
Available online xxx

Keywords:
Hardware/software architecture evaluation
Reliability analysis
Fault trees
Fourier analysis
Sensitivity analysis
Importance analysis

ABSTRACT

We increasingly rely on software-intensive embedded systems. Increasing size and complexity of these hardware/software systems makes it necessary to evaluate reliability at the system architecture level. One aspect of this evaluation is sensitivity analysis, which aims at identifying critical components of the architecture. These are the components of which unreliability contributes the most to the unreliability of the system. In this paper, we propose a novel approach for sensitivity analysis based on spectral analysis of fault trees. We show that measures obtained with our approach are both consistent and complementary with respect to the recognized metrics in the literature.

1. Introduction

We increasingly rely on software-intensive systems such as modern embedded systems employed in telecommunication, consumer electronics, automotive, avionics and health application domains. Software plays a central role in defining the functionality and the quality for these systems. As a result, both hardware and software faults constitute a threat for system reliability. This threat gets amplified as systems continue to grow in size and complexity.

For a long period, software reliability has been basically addressed at the source code level. However, the increasing size and complexity required a special focus on higher abstraction levels as well. In particular, early reliability evaluation at the software architecture design level became essential [1,2]. Software architecture represents the gross level structure of the system, consisting of a set of components, connectors and configurations [3,4]. This structure has a significant impact on the reliability of the system [5]. Hence, it is important to evaluate software architecture with respect to reliability risks [6]. By this way, the quality of the system can be assessed early to avoid costly redesigns and reimplementations.

In the case of software-intensive embedded systems, both hardware and software faults have to be taken into account. To analyze the propagation and interaction of these faults, system level abstract models have been developed. These models include state/path-based models [5,7,8] and (dynamic) fault trees [9,10]. In this work, we employ fault tree models, which depict logical interrelationships among faults that cause a system failure. They have been integrated as part of AADL (Architecture Analysis and Design Language) [11]. There also exist tools for synthesizing them automatically based on UML models [12]. Fault trees can be used for estimating the reliability of the overall system based on individual component failures. Another goal is to estimate the sensitivity of system reliability with respect to reliabilities of system components [5,13,14]. This goal is achieved with so-called *sensitivity analysis* [15] or *importance analysis* [16,17] to identify critical components [18]. These are the components of which unreliability contributes the most to the unreliability of the system.

An established measure for sensitivity/importance was introduced by Birnbaum [19], which is basically defined as the partial derivative of the system reliability with respect to the corresponding component reliability. Hereby, the system reliability is defined as a function of reliabilities of the involved components. There have also been other measures introduced for assessing component importance; however, it was later observed that they provide counterintuitive or inconsistent results [16,20].

In this paper, we propose a novel approach for sensitivity analysis. The approach is based on the spectral analysis of Boolean functions. Spectral (or Fourier) analysis is widely used in mathematics and engineering for decomposing a signal into a sum

* Corresponding author at: Computer Engineering Department, Izmir Institute of Technology, Urla, 35430 İzmir, Turkey. Tel.: +90 232 750 7878.
E-mail address: tolgaayav@iyte.edu.tr (T. Ayav).

http://dx.doi.org/10.1016/j.asoc.2016.06.042
1568-4946/

Please cite this article in press as: T. Ayav, H. Sözer, Identifying critical architectural components with spectral analysis of fault trees, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.06.042

of periodic functions. Representing a function as a sum of simpler functions allows for a sort of probabilistic reasoning about the various parameters of the system. In our approach, we use fault tree models as input, which are commonly used for sensitivity/importance analysis [2,13,14,16,18,21]. We apply spectral analysis on these models to identify critical components of the architecture. We evaluate our approach based on a benchmark fault tree model and two additional subject models derived from a software architecture description of a Pick and Place Unit (PPU) of a factory automation system [22]. We show that the measures obtained with our approach are both consistent and complementary with respect to the recognized metrics in the literature. Moreover, to the best of our knowledge, this is the first study that applies spectral analysis methods to the evaluation of fault trees.

The remainder of this paper is organized as follows. In the following two sections, we provide background information on fault tree analysis and spectral analysis. In Section 4, we introduce our approach for sensitivity analysis. In Section 5, we present an evaluation of our approach. In Section 6, we discuss the results and limitations. In Section 7, we summarize the related studies. Finally, in Section 8, we conclude the paper.

2. Fault tree analysis

A fault tree is a graphical model, which defines causal relationships among faults leading to a system failure. An example fault tree model is depicted in Fig. 1. Hereby, the top node (i.e., root) or the *top event* of the tree represents the system failure. The leaf nodes of the tree (labeled as *a*, *b*, and *c* in Fig. 1) are named as *basic events*. In our modeling approach, each basic event represents a failure of an individual component of the software architecture. We can also see an *intermediate event* in Fig. 1. Such events represent undesirable system states that can lead to a system failure. Logical connectors, which interconnect the set of events, infer the propagation and contribution of these events to other events and eventually to the system failure. For example, we can see in Fig. 1 that basic events *b* and *c* are connected with an AND-gate (depicted with symbol), which in turn is connected to the intermediate event. This means that the intermediate event occurs if both *b* and *c* occur. This can be the case, for instance, if these basic events represent the failures of functionally equivalent software components employed for N-version programming [21]. Another basic event, *a* is connected with the intermediate event through an OR-gate (depicted with symbol), which in turn is connected to the top event. This means that the top event occurs if one or both of *a* and the intermediate event occur. This can be the case, for instance, if *a* represents the failure of a (critical) component, of which failure directly leads to the system failure regardless of the states of other components.

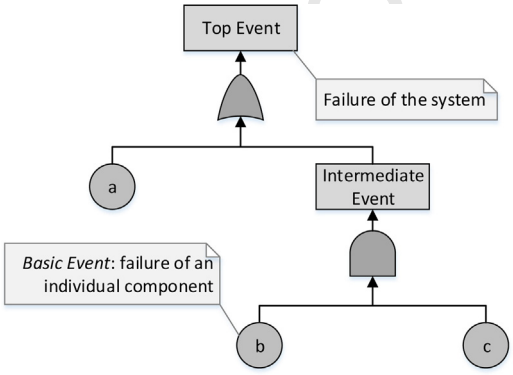

Fig. 1. An example fault tree model.

Occurrence of a set of k events can be represented as a vector of Boolean variables, $\mathbf{x} = [x_0, x_1, \ldots, x_{k-1}]$, of length k. Hereby, $x_i = \mathtt{T}$ and $x_i = \mathtt{F}$ indicate the existence and absence of event i, respectively. Boolean variables and operations are noted and defined as follows:

$$x := F \mid T \mid x' \mid x_1 \Theta x_2, \quad \text{where } \Theta = \{\wedge, \vee\}.$$

An AND-gate represents the intersection of the events attached to the gate. All events must exist for the output event of the gate to occur. For k input events, the equivalent Boolean expression would be

$$f_{\text{AND}}(x) = x_0 \wedge x_1 \wedge x_1 \wedge \cdots \wedge x_{k-1}$$

Let $p_0, \ldots, p_{k-1}$ denote the probabilities of the input events. Under the assumption that these events are independent, the probability of the output event can be defined as

$$p = \prod_{i=0}^{k-1} p_i \tag{1}$$

Similarly, an OR-gate represents the union of the input events. There must exist at least one input event for the output event to occur. The equivalent Boolean expression is

$$f_{\text{OR}}(x) = x_0 \vee x_1 \vee x_1 \vee \cdots \vee x_{k-1}$$

The probability of the output event can be written as

$$p = 1 - \prod_{i=0}^{k-1} (1 - p_i) \tag{2}$$

Let us assume that n different potential failures are identified for a given software architecture. These failures are considered as basic events. Then, a vector of Boolean variables, $\mathbf{x} = [x_0, x_1, \ldots, x_{n-1}]$ can represent the occurrence of these events. So, $x_i = \mathtt{T}$ and $x_i = \mathtt{F}$ indicate the existence and absence of failure i, respectively. The occurrence of the top event can be represented as a Boolean function of $\mathbf{x}$, $f(\mathbf{x})$, where $f(\mathbf{x}) = \mathtt{T}$ and $f(\mathbf{x}) = \mathtt{F}$ indicate the failure and the correct functioning of the overall system, respectively. For the example fault tree model depicted in Fig. 1, this function can be defined as $f(\mathbf{x}) = a \vee b \wedge c$.

Note that in reliability engineering, failure probabilities depend on time, *i.e.* they are expressed as $p(t)$, $t \in [0, T]$ where T is the mission time and they are assumed to be generated from a failure distribution. Therefore all equations depending on the probabilities are also time dependent. For the sake of clarity, we prefer to use a simpler notation throughout the paper by simply omitting the time dependency. This means that all equations presented in this study are applicable to any time $t \in [0, T]$.

2.1. Coherent and non-coherent systems

Fault tree analysis techniques commonly assume that the analyzed system is a *coherent system*, which is defined as follows:

Definition 1 *(Coherent system).* Given a system with n possible component failures, and its fault tree, where the occurrence of the top event is defined by $f : \mathbb{B}^n \to \mathbb{B}$, the system is said to be coherent iff:

1. (Relevancy) $\text{Inf}_i > 0 : \quad \forall i \in \{0, 1, 2, \ldots, n-1\}$,
2. (Monotonicity) $f(x) \geq f(y)$ whenever $x \geq y$ pointwise.

The first requirement states that each component must have an influence on whether or not the system works. Second, $f(x)$ is required to be monotone, i.e., a non-decreasing function. In other words, fixing a component cannot make the system worse. Note

Please cite this article in press as: T. Ayav, H. Sözer, Identifying critical architectural components with spectral analysis of fault trees, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.06.042