Integrated Methodology for Software Reliability Analysis

Marian Pompiliu CRISTESCU[1], Laurenţiu CIOVICĂ[2], Liviu CIOVICĂ[3]
[1, 3]"Lucian Blaga" University of Sibiu, Romania
[2]Academy of Economic Studies, Bucharest, Romania
marian.cristescu@ulbsibiu.ro, laurentiu.ciovica@gmail.com, liviu.ciovica@gmail.com

The most used techniques to ensure safety and reliability of the systems are applied together as a whole, and in most cases, the software components are usually overlooked or to little analyzed. The present paper describes the applicability of fault trees analysis software system, analysis defined as Software Fault Tree Analysis (SFTA), fault trees are evaluated using binary decision diagrams, all of these being integrated and used with help from Java library reliability.

Keywords: *Software Reliability, Binary Decision Diagram, Fault Trees, Combinatorial Models, Analytical Techniques*

1 Introduction

Safety critical software systems refer to those systems blackouts that may cause catastrophic consequences, implying human and substantial material losses [10]. A variety of techniques is available and used to ensure safety and reliability to the system. Some of those include: HAZOP method (Hazard and Operability Analysis), which deals with risk study and system operability; fault tree analysis (FTA), used to analyze potential danger causes and FME(C)A technique (Failure Modes Effects (and Criticality) Analysis), used to check the proper system functionality.

Most of the techniques mentioned before, are applied together as a whole, and in most cases, the software components are usual overlooked or very little analyzed.

This paper describes an integrated framework for software reliability analysis using the following analyses and modeling techniques:

- software fault tree analysis (SFTA);
- binary decision diagram;
- Java technology for reliability.

In use of this analysis framework are fallowed these steps:

- it starts from a structural/functional analysis of the software system, in order to determine software's failure mechanisms (failures), based on which the fault tree is constructed. The program is seen from a structural point of view, taking in consideration his components and the relations between them;
- after the structural - functional analysis is made, the primary events group (primary defects) and the group of critical events (failures) are identified, with the help of which the fault tree is built;
- using the resulted tree, different computation operations are made, such us tree minimization, through known methods, to ease the implementation process using the Java language;
- for the automated decision binary diagram generation an open source Java library is used, called JReliability (Java-based Reliability Library), library developed for modeling and analyzing the reliability of complex systems using BDD. Thus the most of the research was made on hardware systems in this paper the authors extends the library utility towards software systems;
- at last, after the BDD generation a quantitative analysis is made in order to evaluate the reliability indicators put at disposal by the JReliability, through the analytic results and through graphic representations.

First are presented the techniques used until the present and well researched in the field's literature: software fall tree analysis, SFTA and binary decision diagrams, BDD.

2 Combinatorial Models

Combinatorial models represents a structured approach of systems, forming a class of reliability models in which the causes of software failure can be expressed in terms of combinations of software components (modules) faults (failures) [4]. Combinatorial models include models with graphs used in network reliability analysis, fault trees and reliability block diagrams. These models do not require the fulfillment of constant failure rate assumption and have been successfully applied for reliability analysis of mechanical and electrical systems. In particular, failure trees were implemented also for reliability software analysis.

An alternative to SFTA [1] (a traditional approach of models with solutions based on minimal cuts) uses the so called binary decision diagrams (BDD) [2]. Binary decision diagram were used initial as a checking technique in electric circuit theory, but recently were adapted to resolve a failure tree model for quantitative and qualitative reliability analysis [12]. Because it seems that there is no connection between the number of modules (components) from a system and the dimension of the appropriate BDD, the BDD based solutions can offer efficient solutions techniques for big systems.

2.1. Failure Trees

An often used technique proven to be efficient for the software reliability analysis is the software failure tree analysis (SFTA). This technique borrows the failure trees concept (FTA) from the hardware field.

FTA technique represents a deductive approach, having the system failure mechanism as a base [17]. FTA starts with an unwanted event, such as "obtaining unauthorized access on a server", and then determines (deducts) its causes using a backward systematic process, step by step. To determine the causes, a failure tree is built as a logical representation of events and the relations between them, which are necessary and sufficient to detect the unwanted event, called the top tree event (top event).

Failure tree represents a qualitative model which provides a useful set of information about the causes which led to the manifestations of the unwanted event - called the critical event. Also, the tree can be quantitative evaluated to offer useful information about the probability of the top tree event to be produced, as well as the defects' importance and the interdependence between them, modeled in the failure tree.

Itself the failure tree represents a graphic model of various parallel and sequential failures combinations, combinations which lead towards the manifestations of the predefined unwanted event. The defects can be events assigned to hardware components failures, human errors, software errors or any other event which can lead to the manifestation of the critical event. Thus, the tree describes the logical interdependency between the primary events (failures) resulting in the production of the top tree event.

It is important to understand that a failure tree is not a model which describes all possible system failures or all possible causes of its failure. A tree is built around a critical event depending on its particular failure mechanism, and the tree will contain only those events (failures) which contribute on making the tree top event possible. More, those defects are not exhaustive; they cover only the failures evaluated to be realistic by the analysts.

Also, it is also know that a fault tree is not by itself a quantitative model, but represents a qualitative model which can be evaluated even from a quantitative point of view.

The result of the failure tree is a binary event, i.e. either towards success or towards failure. The graphic model of the tree is compound from an entities' complex called "gates" which serve to allow or to deny the logic path towards the top event. The gates point to the necessary relations for producing a "superior" event. The symbol of a gate indicates the relation type between the input methods necessary for producing the output event.

In Figure 1 is represented an example of failure tree.

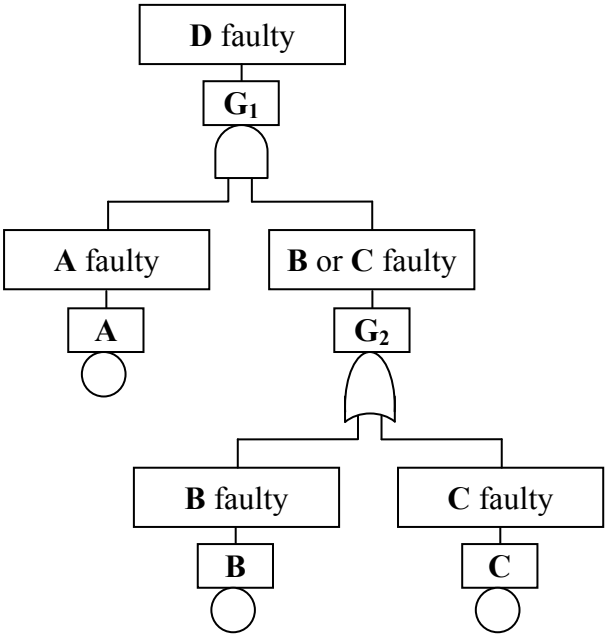


Fig. 1. Simplified example of failure tree

2.1.1 Software failure tree analysis (SFTA)

Software failure tree analysis technique represents a method of identification and documentation of inferior level software events which allows to an event on a superior level (root node) to manifest. If the root event is a critic one, the FTA technique helps the requests' defining process, describing the ways in which the system can reach that uncertain state [8].

To analyze the software reliability using the SFTA technique, first we must define the unwanted event as a root event. All the root events define the set of root events. Each root event has its own failure tree. Using logic relations between program modules we can analyze which modules or complex events (middle events) can cause the manifestation of the root event. The complex events analysis is continued until a stopping condition it reached, more exactly on a primary (failed) event. So, based on this analysis it can be represented the software failure tree.

A set of cuts represents a group of primary events (failures) whose manifestations lead to the falling of the system [11]. A set of cuts is called a minimal set of cuts only when they cannot be reduced and can cause the system failure. The minimal set of cuts provides a minimal set of successful events required to satisfy the root node. To obtain a set of minimal cuts, we use an algorithm "top-down", more precise the Fussell - Vesely algorithm [3]:

1) Down from the root event, events are listed taking in consideration the various logical relationships between them.
2) If the gate underneath the root node is an OR logical gate, input events are showed on different rows. Even if it is a logical gate the input events are showed on a single row.
3) Taking in consideration the complex event, G_i, it is also treated as a root event (same as the step 1) and the step 2 is repeated until the primary (failure) event it is reached.
4) Finally, a few sets of events are obtained representing all the cutting sets for the failure tree.
5) To obtain the minimal cut set from all those cuts obtained on 4th step the next operations are made: first of all the cuts are arranged depending of their number of events where every cut is been viewed as a product of primary events. Then the fallowing relations are used to absorb the redundant cuts. Finally, the minimal cut set it's obtained.

$$A + A = A$$

$$A + A \cdot B = A$$

$$A \cdot A = A$$

Figure 2 shows the graph for obtaining all the minimal cuts form the failure tree from figure 1. Minimal cut set is composed from {A,B} and {A,C}

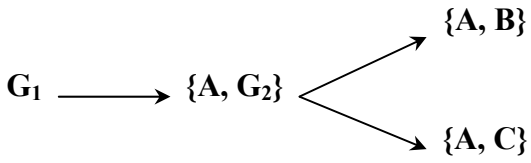


Fig. 2. Minimal Cuts Generation

It's been also showed that the SFTA technique can be realized on different levels and steps of the software developing process. The highest analyze level it's represented by the functional description. On the lowest level of

investigation, the SFTA technique analyzes the source code of the program. In this paper, it is used the SFTA methodology on the functional modules level, in order to analyze the reliability relations and the intrinsic' requests of the system modules.
In this paper, taking in consideration a software system, the modules (components) are viewed as primary events, and the root node is associated with the system failure event. Then, resorting to SFTA technique, we can find out which are the important modules to which a higher reliability and security can be allocated, being more likely to lead the system to failure, if they are unreliable.

2.1.2 Using the SFTA technique for software reliability allocation [7]

It is consider P as being the maxim failure probability accepted and the software system is composed from n modules: $m_1, m_2, ..., m_n$. Using the SFTA technique will obtain a number x of minimal cuts, after which the fallowing algorithm is applied.
***Algorithm 1*:** if a minimal cut contain a number of i modules, then the probability of failure of each module from the cut it's been given by the relation:

$$P_{m_j} \le \left(\frac{P}{x}\right)^{1/i}, (j = 1,2, \dots ,n) \qquad (1)$$

If there are intersections in the minimal cut set, the failure probabilities P_{m_j} can have k different values, noted with $y_1, y_2, ..., y_k$. Case in which can define

$$P_{m_j} = min(y_1, y_2, ..., y_k) \qquad (2)$$

It must be mentioned that the algorithm use the geometric mean, in a certain way [7], the process being a reversed one regarding the classical analysis of system unavailability using SFTA [9,pp 619 - 627]. Using the classic method, by giving the failure probabilities to each module (data gather in the testing step), we can find out the failure probability of the entire system. The methodology used here, represents the reversed method [7]: knowing the reliability' request of the software system, from the planning or designing phase, the reliability of each module it's been investigated and determined, along with interdependency between modules.
According to the present algorithm, the probabilities P_{m_j} are obtained by applying the geometric mean which can have unimportant small errors. So, we can offer, to software engineers, a comprehensive understanding regarding the structure of the software system and the reliability request of each module; information which can help to determine the most important modules (key modules), allocating a higher reliability to the last ones.

Example 1:
For the practical explanation of the algorithm, the failure tree from Fig. 3 is examined for a software system made from six module $m_1, m_2, ..., m_6$.

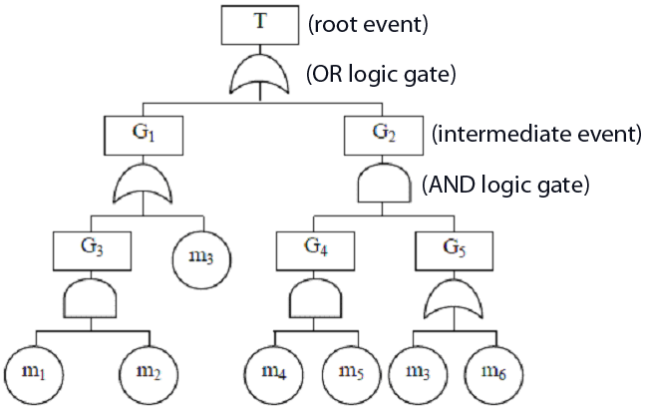


Fig. 3. An example of failure tree

The requirement is that the probability of the system failure to be $P \le 0,03$. In which way the reliability must be allocated for each module?
Using Fussel-Veseley [3] algorithm, all the cuts are first obtained in Fig. 4, for the tree from fig. 3. So, the generated cuts are:
$K_1 = \{m_3\}$, $K_2 = \{m_1, m_2\}$, $K_3 = \{m_3, m_4, m_5\}$ and $K_4 = \{m_4, m_5, m_6\}$

But, $m_3 = m_3 + m_3 \cdot m_4 \cdot m_5$, with other words K_1 it's subset from K_3, finally obtaining the minimal cuts: $\{m_3\}$, $\{m_1, m_2\}$, $\{m_4, m_5, m_6\}$.

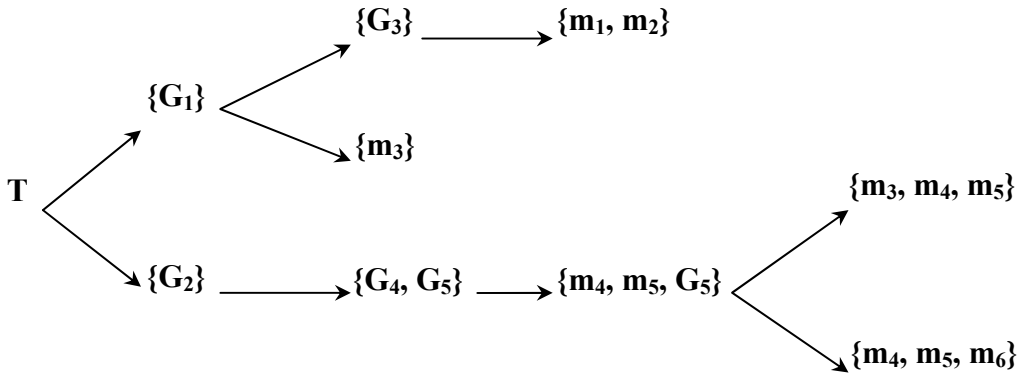


Fig. 4. Generating the minimal cuts for the tree from Figure 3

Then, appling the 1st relation, the fallowing results are obtained:

$$P_{m_3} \le \left(\frac{P}{3}\right)^{1/1} = 0,01 \quad (3)$$

$$P_{m_1} = P_{m_2} \le \left(\frac{P}{3}\right)^{1/2} = 0,1 \quad (4)$$

$$P_{m_4} = P_{m_5} = P_{m_6} \le \left(\frac{P}{3}\right)^{1/3} = 0,215 \quad (5)$$

In other words, the maximum failure probability of each module is:

(0.1, 0.1, 0.01, 0.215, 0.215, 0.215)

So that the reliability of modules must be

(0.9, 0.9, 0.99, 0.785, 0.785, 0.785).

Based on the minimal cut definition and the analysis made above can reach an important conclusion: if the number of modules from a cut is low, then for those modules the highest reliability must be allocated.

In example 1, because the number of modules from the minimal cut set follows the relation *K1 < K2 < K3,* the fallowing conclusion is asserted: m_3 is the key module, meaning that is the most important component from the system, thus must receive the most reliability, and the m_4, m_5, m_6 modules can receive a lower reliability than m_3.

2.2 Binary decision diagrams

The concept of binary decision diagram (BDD) was used, in the beginning, in the design and verification of VSSI (very large scale integration), as being an efficient method for the manipulation of Boolean expressions [14,9]. Bryant [2] and other researchers showed that, in most cases, the BDD technique use less memory to represent large Boolean expressions, instead of the common representation.

The binary decision diagrams are based on the Shannon decomposition, which makes the reliability evaluation to be accomplished very easy from the BDD form. A series of researchers successfully used this method in the reliability analysis of the failure trees [11, 15, 15 and 7].

The methodology presented in this papers it's based on the analysis of failure trees using binary decision diagram. Besides the technique already presented in the field's literature, the technique presented in this paper goes one step further, integrating this technology with the Java advanced programming techniques and technologies, used to automate the generation of failure trees and the generation of some reliability graphic indicators.

2.2.1 Shannon Decomposition in ITE (If – Then - Else)

Shannon decomposition theorem (or the developing theorem) is presented as fallowing: f is a function that represents a Boolean expression defined on the X set, and $x \in X$. According with the theorem, the relation could be written as:

$$f = x \cdot f_{x=1} + \overline{x} \cdot f_{x=0} \qquad (6)$$

where f is evaluated in $x = v$ and get noted with $f_{x=v}$.

Shannon decomposition is behind the use of binary decision diagram. To precise describe this theorem, it's defined as If-Then-Else (ITE) as fallowing:

$$f = ite(x, F_1, F_2) = x \cdot F_1 + \overline{x} \cdot F_2 \qquad (7)$$

where $F_1 = f_{x=1}$ and $F_2 = f_{x=0}$.

A BDD diagram represents a directed acyclic graph which has the Shannon decomposition at its base. The graph has a single root node and two terminal node, noted with 0 and 1, representing the two constant expresions. Each complex node is noted with a *x* boolean variable and has two exit branches. These two branches are called 0-branch (*else* branch) and 1-branch (*then* branch). The node connected with the 1-branch represents a boolean expresion when $x = 1$, i.e., $f_{x=1}$ from the equation (6), while any node connected to a 0-branch indicates a Boolean expressions when x = 0, meaning $f_{x=0}$ in equation 6. In fact, each complex node from BDD encrypts if-the-else instructions.

An ordered binary decision diagram (OBDD) fallows the rule thrugh which the variables (nodes) are ordered, and the chart depth crossing is done in an ascending order following the nodes order. An reduced and ordered binary decision diagram (ROBDD) is a OBDD diagram in which each node represents a different boolean expresion.

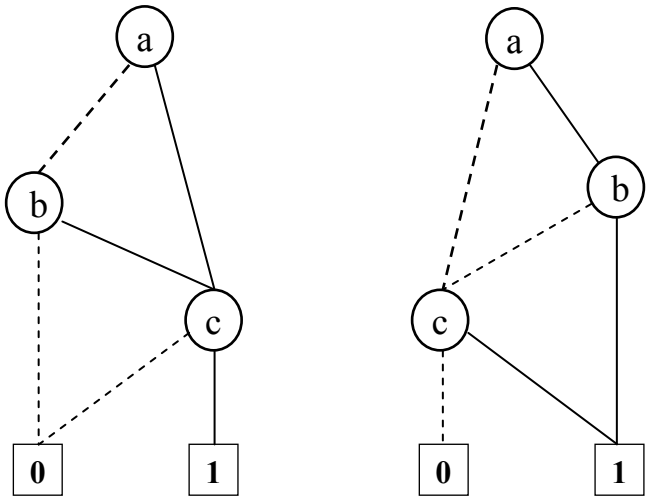
In practice, ROBDD diagrams are often used. To be able to generate a ROBDD diagram, first we must ordonate the variables, and this order must remain the same during the diagram generation process. In this paper, we note with $x_i < x_j$ in case in which the x_j variable is behind the x_i variable, in the ordered node list.

Figure 5, describes two examples of ROBDD diagram generated for two different boolean expresion.

Appling the equation (7), the *ite* forms of the *g* and *h* expressions are:

$$g = ite(x, g_{x=1}, g_{x=0}) = ite(x, G_1, G_2) \qquad (8)$$

$$h = ite(x, h_{x=1}, h_{x=0}) = ite(x, H_1, H_2) \qquad (9)$$



(a) $g = a \cdot c + b \cdot c$ (b)

$h = a \cdot b + c$

Fig. 5. BDD examples for two Boolean expressions

In practice it's been shown that the BDD are generated using rather logical operations on the variables instead of applying directly the Shannon development.

2.3 Quantitative Analysis

Quantitative analysis is performed on the binary decision diagram (BDD) [12] representing an exact and efficient method allowing us to determine many reliability characteristics of a hardware/software system.

Furthermore we will refer to methods which use the BDD's containing complex or/and

modular events. The analysis purpose [2] is that to obtain probability of the critic event and the intensity of unconditioned failure.

2.3.1 System Unavailability

The probability of the critical events manifestation, noted with Q_{sys} it's obtained through the sum of probabilities obtained on disjoint branches from crossing the primary BDD diagram. An algorithm of depth crossing (depth first) can effectively perform this calculation; in paper [15] an elaborate description of this method is presented.

To make this calculation we need to know the unavailability of each coded event. For this, the probability of complex events and of modular ones must be obtained from the primary events data obtained experimental.

The determination of unavailability complex events is straightforward, these events being a combination of two component events. The calculation depends on how events are combined, under a "AND" logic gate or under a "OR" gate, thus, if x_c is a complex event and its constituent events are x_1 and x_2 , then fallowing will exists:

$$\text{logic gate "AND" : } q_c = q_1 \cdot q_2 \qquad (10)$$

$$\text{logic gate "OR": } q_c = q_1 + q_2 - q_1 \cdot q_2 \qquad (11)$$

where: q_c is the probability of achieving the complex event, and q_1 and q_2 are the probabilities of realization of x_1 and x_2 events.

The probabilities of complex events are calculated according to their construction, making the process more efficient.

The calculation of modular events probabilities it's made by determining the probabilities of the "top tree event" manifestation for each separate module. Also, in this case depth crossing algorithm is used. Thereby, first the unavailability of those modules must be evaluated, which encrypts only the primary and complex events.

After the probabilities of the complex events were obtained along with those of the modular ones, the system unavailability can be easily determined.

2.3.2 Unconditioned Failure Intensity of the System

Unconditioned failure intensity of the system (called the system failure frequency) $W_{sys}(t)$, is defined as the probability with which the critic event (root node) manifest itself at the t moment and it's calculate using the fallowing relation:

$$w_{sys}(t) = \sum_i G_i\left(q(t)\right) \cdot w_i(t) \qquad (12)$$

where $G_i(q(t))$ it's the critically function (or the risk function associated to each failure node) for each component, and $w_i(t)$ represents the unconditioned failure intensity of one component.

Explicitly, the risk function is defined as the system probability to be in a critical state, in relation with the component i and, so, failure of this component would make the system to pass from a functionality state to a failure one. So, it's result the relation:

$$G_i\left(q(t)\right) = Q\left(1_i, q(t)\right) - Q\left(0_i, q(t)\right) \qquad (13)$$

where: $Q(1_i,q(t))$is the probability of system failure with $q_i(t)=1$, and $Q(0_i,q(t))$ is the probability of system failure with $q_i(t)=0$.

An efficient method for calculating criticality takes in consideration the probability of diagram branches before and after the respective nodes, leading to the fallowing expression:

$$G_i\left(q(t)\right) = \sum_n pr_{x_i}\left(q(t)\right) \cdot \left[po^1_{x_i}\left(q(t)\right) - po^0_{x_i}\left(q(t)\right) \right] \qquad (14)$$

where: $pr_{x_i}\left(q(t)\right)$ - is the branch probability which leaves from the root node towards the x_i node, $po^1_{x_i}\left(q(t)\right)$ - the path probability on branch '1' of the node x_i towards terminal node 1,

$po^0_{x_i}(q(t))$ - the path probability on branch '0' of node x_i towards the terminal node 1, and

n - all nodes corresponding to variable x_i from DDB.

For a single BDD, which codes only the primary events is necessary a single crossing of diagram in order to calculate $pr_{x_i}(q)$, $po^1_{x_i}(q)$ and $po^0_{x_i}(q)$ for each node, from where it can be obtained the risk function of each primary event, leading to the unconditioned failure intensity evaluation of the system.

However, this method doesn't take in consideration the modular and complex events. It is possible the calculation of W_{sys} taking in consideration only the primary events coded in the primary diagram, but this presumes the knowledge of both the risk of modular and complex events and the intensities of unconditioned failures. Even if these are not hard to gather, they represent some values which don't present any utility in the performed analysis. Rather than those the risk functions of the primary events will be calculated which will be used together with their unconditioned failures intensities to determine W_{sys}.

These risk functions of primary events, from the primary BDD diagram, can be also calculated while obtaining the path probabilities for the primary diagram nodes.

3 Using Java Language in Software Reliability Analysis

Within the methodology for integration of more reliability techniques in a global framework it's been included a Java library, with the help of which it can be generated a binary decision diagram (BDD) and different reliability indicators can be evaluated for a considered software system

JReliability represents a library written in Java programming language which has as purpose the reliability evaluation of a system and have been developed in the Hardware/Software Co-Design Department of the Computer Science Faculty from Erlangen-Nuremberg University from Germany, [18]. The library is adapted (but not limited to) modeling and analysis based on BDD of complex systems, such us network integrated systems. Till now various materials have been published at internationals conferences, researches based on analysis and modeling for this library [4, 9, 11].

This paper extends the *JReliability* library utilization in order to analyze the system software reliability.

JReliability library is based on previous developed libraries, such as: JavaBDD - a Java library for binary decision diagram manipulation; Ptolemy Plot (java library for plotters).

JReliability allows gathering reliability indicators, such as MTTF, or mission time (MT) of complex systems which are used using Boolean functions, efficient represented through binary decision diagrams.

The library is very extensible so various special evaluators for gathering others reliability indicators can be included along with designing other system representation.

More, *JReliability* offers a graphic user interface (GUI) to visualize the obtained reliability indicators, also the possibility to view the Boolean representation of the system (the BDD), using DOT graphic language. Figure 6 presents the result of reliability evaluation for the failure – tolerant structure of TMR (Triple-Modular-Redundancy) type.

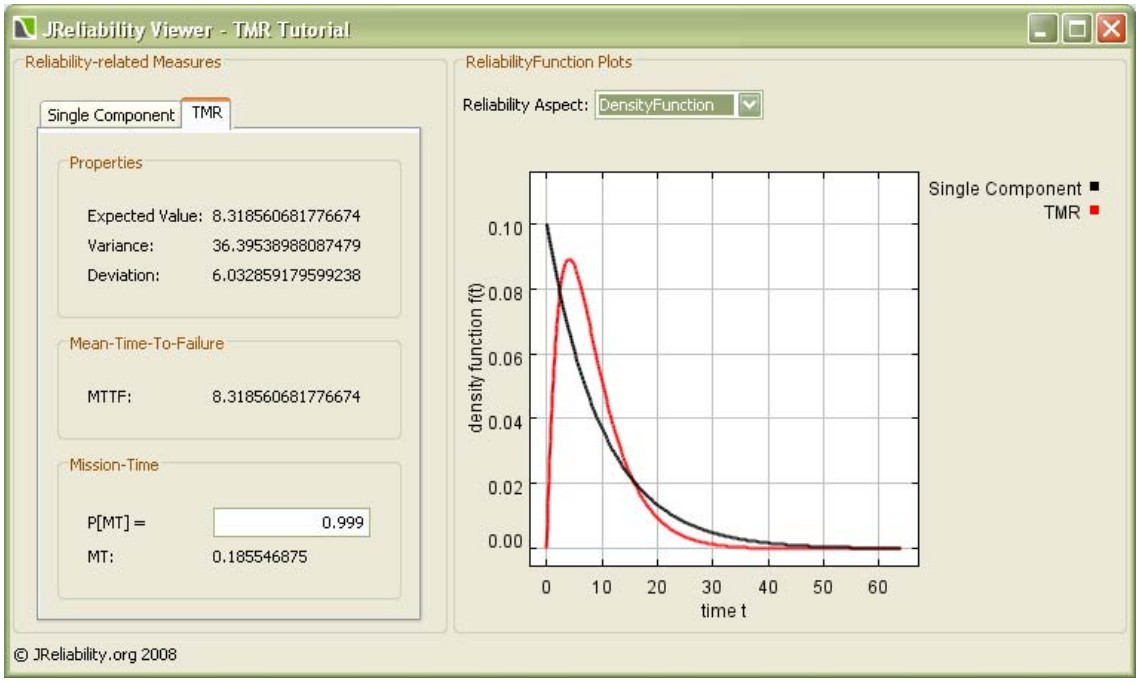


Fig. 6. Analysis of the TMR reliability structure using *JReliability*

3.1 Operation of *JReliability*

The idea which underlines the use of JReliability library is that there is a Model of the system which describes the system behavior in the case of failure manifestation, failures or repairs of the system component. Having this Model it can generate the overall *Reliability Function*. In the last step, a set of evaluators will obtain reliability indicators, based on the Model and *Reliability Function* or, if possible, only on the *Reliability Function*. The principle is schematically described in Figure 7.

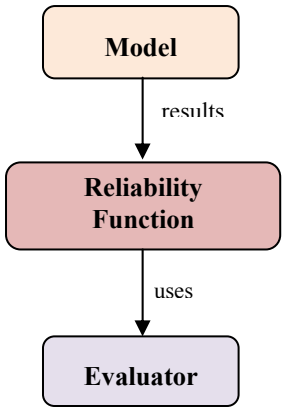


Fig. 7. Functional principle of *JReliability*

3.2 Reliability Modeling with *JReliability*

The behavior of a system (failure mechanism) is usually represented by failure trees, reliability block diagrams (RBD), automated, Markov chains or Boolean functions. All of this techniques have in common the fact that exist, implicitly or explicitly, a data structure which can observe if the system function properly, to a certain point, or if is unavailable, i.e. doesn't work properly.

JReliability uses a Boolean function to represent system structural function; function encrypted with binary decision diagrams (BDDs)

To obtain the *Reliability Function* of the whole system, it must be known the Reliability Function of each component from the system. For this purpose, *JReliability* is using a data structure called *Transformer Function*, which represents mapping of each element (component) corresponding to his own *Reliability Function*. *JReliability* comes with a wide set of *Reliability Function* known and predefined, based on exponential distributions, Weibull distributions and many others.

Having this knowledge, *JReliability* structure can be refined like in the Figure 8.

Note that the BDD which encrypts the Boolean functions combined with the *Transformer Function* represents an approach to obtain the global *Reliability Function*. The user can implement other methods which will lead to expanding the use of *JReliability* library.

3.3 Reliability Evaluation

Once the global *Reliability Function* of the system has been obtain, it can evaluate different reliability indicators such us: average lifetime until the failure – MTTF, mission timing – MT, failure rate, etc., using the so called *Evaluators*. An evaluator receive from input a *Reliability Function* and evaluate the wanted indicator either analytic, either by gathering evidence or simulating.

Some Evaluators requires, also, the access to *Reliability Function* of each component from the system and to structure Boolean functions of whole system, encrypted in BDD. To reach this purpose, was developed a special *Reliability Function*, called *BDD Reliability Function*, which offers the access to *Reliability Function* of each component, and to whole BDD.

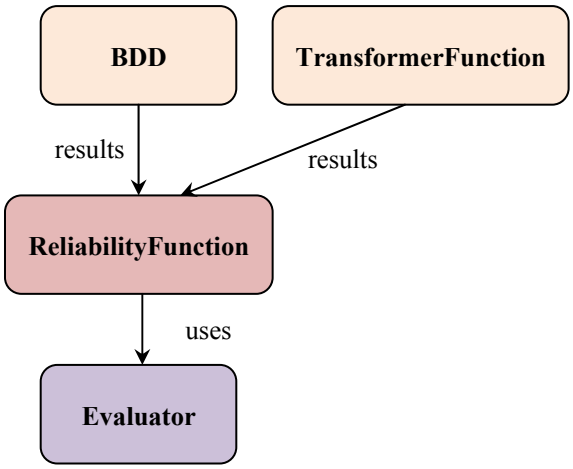


Fig. 8. *JReliability* extended structure *Model = {BDD, TransformerFunction}*

3.4. Binary decision diagrams role in *JReliability*

The basic model of system behavior, current in *JReliability*, is the Boolean function. Data structure used to efficient encrypt this kind of functions in *JReliability* is been given by binary decision diagrams. An important characteristic of the *JReliability* library is that it offers a generic special interface for a series of BDD libraries, which allows directly using BDD variables of Java objects to model components of the real system

BDD allows canonic representation of Boolean functions. BDD represents an acyclic direction graph with root and is made from decision nodes, which corresponds to variables and two terminal nodes, 0 and 1, which corresponds to returned value of Boolean function. Since each is a binary variable which can make only 0 and 1 value, each decision node presents two output ramifications, low and high, corresponding the case in which the variable take 0 and 1 values. This process is presented in fig. 9. Any values attribution for variables which lead to terminal node 1 corresponds of a functional system, while any attribution of values for variables which lead to terminal node 0, corresponding to a system in a failure state.

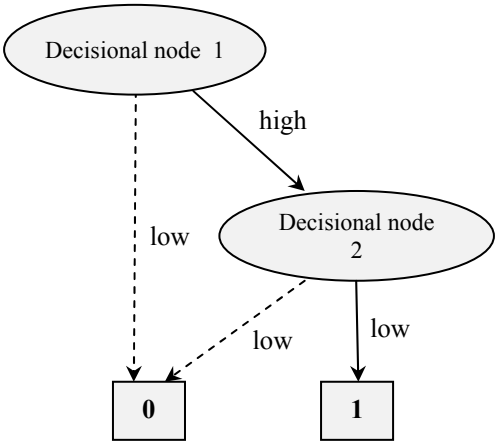


Fig. 9. BDD data tip structure example

In this reliability library, each variable corresponds to one component, being represented by a Java object. Value 0 corresponds to a failed component and value 1 to a component in a good state of functionality. Objects which models the components are directly connected between them by logic operators, AND or OR, with which all structure functions is encrypted in a single BDD.

4 Conclusions

SFTA technique used for allocation software reliability is a clear, simple and efficient. IT can be used not only at the level of system analysis, but also at modules and subsystem analysis level.

Because the construction process of failure tree offers a comprehensive understanding of the system, it's been asked from the software engineer a better control of relations between

the modules, to remove any factor with an unsafely potential, so the result of reliability to be more rational.

More, SFTA represents a deductive graphic method, describing in a clear way the relations from the system framework, what makes easy the identification process of key modules, with the purpose of reliability allocation.

This paper introduces a new integrated framework for software reliability analysis, using analysis and modeling techniques:

- software failure tree method, SFTA;
- binary decision diagram, BDD, for the SFTA analysis;
- Java evaluation techniques for Java reliability, *JReliability*

With the help of Java library, *JReliability*, we could automate the generation of binary decision diagram for a considered system, after applying the SFTA technique. The new methodology is simple, efficient, having a considerable development factor, limited only at software engineer knowledge level which analyzes the systems reliability.

Other analysis and modeling approaches (Markov model, simulation models) are necessary for the cases in which a system failure depends even by the order in which the events (errors) appears, and not only by their simple combination.

References

[1] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *DAC '90: Proceedings of the 27th ACM/IEEE Design Automation Conference*, New York, NY, USA, 1990, pp. 40–45.

[2] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Comput. IEEE Transactions on Computers*, vol. 35, no. 8, pp. pp. 677-691, Aug. 1986.

[3] M.H. Chen, M. R. Lyu, and W. E. Wong, "Effect of code coverage on software reliability measurement," *IEEE Trans. Rel. IEEE Transactions on Reliability*, vol. 50, no. 2, pp. 165-170, 2001;

[4] S. A. Doyle, "Dependability Assessment Using Binary Decision Diagrams (BDDs)," in *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, 1995, p. 249;

[5] M. Glass, M. Lukasiewycz, F. Reimann, C. Haubelt, and J. Teich, "Symbolic reliability analysis and optimization of ECU networks," in *Proceedings of the conference on Design, automation and test in Europe*, Munich, Germany, 2008, pp. 158-163;

[6] M. Glass, M. Lukasiewycz, F. Reimann, C. Haubelt, and J. Teich, "Symbolic Reliability Analysis of Self-healing Networked Embedded Systems," in *Proceedings of the 27th international conference on Computer Safety, Reliability, and Security*, Newcastle upon Tyne, UK, 2008, pp. 139-152;

[7] X. Jianwen, F. Kokichi, and H. Yanxiang, "Fault Tree Analysis of Software Reliability Allocation," in *Proceedings of 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003)*, Orlando, USA, 2003, vol. Volume II - Computer Science and Engineering, pp. 460-465;

[8] N. G. Leveson, SafeWare: system safety and computers. MA, USA: Addison-Wesley, 1995;

[9] M. R. Lyu, Handbook of software reliability engineering. Los Alamitos, Calif.; New York: IEEE Computer Society Press; McGraw Hill, 1996;

[10] M. Towhidnejad, L. Shen, and T. B. Hilburn, "Application of Software Fault Tree Analysis to an Airport Ground Control System," in *Software Engineering Research and Practice*, 2008, pp. 67-71;

[11] D. Raheja, Assurance technologies: principles and practices. New York: McGraw-Hill, 1991;

[12] K. A. Reay and J. D. Andrews, "A fault tree analysis strategy using binary decision diagrams," *Reliability engineering & system safety.*, vol. 78, no. 1, pp. 45-56, 2002;

[13] F. Reimann, M. Glass, M. Lukasiewycz, J. Keinert, C. Haubelt, and J. Teich, "Symbolic voter placement for dependability-aware system synthesis," in *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software co-design and system synthesis*, Atlanta, GA, USA, 2008, pp. 237-242;

[14] R. Remenyte and J. D. Andrews, "Qualitative Analysis of Complex Modularized Fault Trees Using Binary Decision Diagrams," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 220, no. 1, pp. 45-53, 2006;

[15] R. M. Sinnamon and J. D. Andrews, "Quantitative fault tree analysis using Binary Decision Diagrams," *JOURNAL EUROPEEN DES SYSTEMES AUTOMATISES*, vol. 30, no. 8, pp. 1051-1072, 1996;

[16] R. M. Sinnamon and J. D. Andrews, "Improved Accuracy in Quantitative Fault Tree Analysis," *Quality and reliability engineering international.*, vol. 13, no. 5, pp. 285-292, 1997;

[17] W. E. Vesely, J. B. Dugan, and J. Fragola, "Fault Tree Handbook with Aerospace Applications," *Director*, p. 218, 2002;

[18] *** JReliability (Java reliability library) – Web: Available at: http://jreliability.sourceforge.net/

Marian-Pompiliu CRISTESCU has graduated the Faculty of Planning and Economic Cybernetics in 1985, he holds a PhD diploma in Economics from 2003, obtained at the Faculty of Cybernetics, Statistic and Economic Informatics from the Academy of Economic Studies in Bucharest. Between 1985 and 1991 he worked as an analyst – programmer at the I.A.C.M Computation Office in Olt – Slatina and at the Electronic Computation Territorial Center in Sibiu. In 1991 he joined the university teaching system, going through all didactic positions starting with teaching assistant, lecturer in 2001 and assistant professor since 2008. Presently, he is a full assistant professor in Economic Informatics at the Faculty of Economic Sciences - "Lucian Blaga" University of Sibiu. He is the author of 8 books and over 60 scientific articles in the field of software quality, programming environments, data bases and economic informatics systems. He is equally focused on software development, being the author or co-author of over 25 programming systems for economic management. He has received many diplomas to certify his achievements in the scientific research domain. He has participated as a project director or as a leading team member at 8 research contracts. He is an active member of the scientific and editing committee for the following magazines and journals: Economic Informatics, Journal of Applied Quantitative Methods, The Economic Magazine edited by the Academy of Economic Studies of Moldavia – Chişinău and the „Lucian Blaga" University of Sibiu. He has participated in the scientific committee of over 10 national and international conferences, for the Informatics section and has coordinated the editing of 2 volumes with the projects of some international scientific conferences.

Vasile-Laurenţiu CIOVICĂ has graduated the Faculty of Science, in 2008 gaining a Bachelor of Science degree in Information Technology with a thesis on Translators and Interpreters for Code Generation and Software Optimization. In 2010 he gained a Master of Management degree in the field of Cybernetics, Statistics and Economic Informatics with a thesis on Intelligent Agents. He is currently a PhD student at Academy of Economic Studies in Bucharest. Between 2006 and 2010 he worked as a programmer at a company from Sibiu. Since January 2010 he works as an Independent Consultant. He is the author and

co-author of more than 12 scientific articles in the field of software quality and optimization, code generation techniques, collaborative systems, data bases, programming environments and techniques, mobile platforms and economic informatics systems. Besides the scientific activity he is also an active software developer, being the author of few applications. Some of the created applications were presented to different student's scientific conferences where he was distinguished with 1 excellence award, 6 first awards, 1 second award and 2 third awards. His area of interests includes among others: software quality, optimization techniques and algorithms, code generation techniques, economic informatics systems, intelligent and collaborative systems, mobile platforms.

Ion – Liviu CIOVICĂ has graduated the Faculty of Science, in 2009 gaining a Bachelor of Science degree in Information Technology with a thesis on "Software application for medical diagnosis assisted by computer". He is currently a student at the Faculty of Science master program, field of Cybernetics, Statistics and Economic Informatics.