Gradus Vol 8, No 2 (2021) 123-128
ISSN 2064-8014

SOFTWARE QUALITY IMPROVEMENT BY FAULT TREE ANALYSIS

Zsolt Csaba Johanyák[*]

[1] Department of Information Technology, GAMF Faculty of Engineering and Computer Science, John von Neumann University, Hungary
https://doi.org/10.47833/2021.2.CSV.003

Keywords:
safety risk analysis
FTA
software quality

Article history:
Received 17 Aug 2021
Revised 22 Aug 2021
Accepted 28 Aug 2021

Abstract
It is common problem with complex software systems that although usually they work as intended, after some time of usage security-critical flaws poop up. In order to alleviate this problem, this paper aims to demonstrate the application potential and benefits of the Fault Tree Analysis (FTA) which is a widely used method in quality assurance. FTA is used as an integral part of the software quality management process to identify the causes of suspected security vulnerabilities by complementing the widely used testing procedures.

1 Introduction

It is a quite usual scenario that a software system meets the requirements of the customers, i.e. all the demanded functionality is built-in, but the product is not secure. For example, a Web browser perfectly downloads and displays the pages requested by the user, runs the scripts placed on them, but as a result of an unforeseen malicious instruction sequence, it makes the information stored on the hard drive accessible. Often, due to the complexity of the software system, developers are not able to identify all the security risks before the actual deployment. When these flaws are cost-intensive in addition to the usual verification techniques it could be beneficial to use well-established methods in the traditional areas of quality assurance, such as Fault Tree Analysis (FTA) [2][10] or Failure Mode and Effects Analysis (FMEA) [6].

In this paper, we discus some application details of the FTA in case of a software project illustrating the concepts by a practical example. The rest of this paper is organized as follows. Section 2 gives a short introduction to the ideas of FTA including the preliminary risk analysis and introducing a short program written in C, followed by the exploration of cause and effect relationships and the construction and explanation of the fault tree. The conclusions are drawn in Section 3

2 Fault Tree Analysis

FTA, by focusing on catastrophic events, allows the identification of environmental conditions under which an otherwise correct system state (mode of operation) may become safety critical. The method was originally developed in the 1960s in the United States by Bell Labs for the safety analysis of the Minuteman missile system [2], and has subsequently been widely used in the mechanical and electronics industries to assess the reliability of various systems.

In our case the aim of the analysis is to evaluate the safety of a software design or implementation and to eliminate risks. As a result, the design can be modified to achieve the required level of safety. Fault tree analysis allows

- to identify all errors and combinations of errors leading to a main event and their causes,
- detection of critical events and event chains,
- build test cases to identify the most dangerous modules,

* Corresponding author
E-mail address: johanyak.csaba@gamf.uni-neumann.hu

- clear and transparent documentation of the propagation mechanisms.

The design of software systems can be described as a forward chaining (data-driven) inference process, where developers build incrementally refined components from initial data, expectations and previously created components, not necessarily for the same conditions.

The method described below, in contrast, uses a backward chaining (goal-driven) technique. The analyst looks at the system from a completely different perspective to the designer. It starts from a hypothetical system failure (main event) and progressively explores the component and subsystem failure modes that lead to the occurrence of that event. The clear work is supported by a fast-structured graphical representation (Fig. 1 and 2).

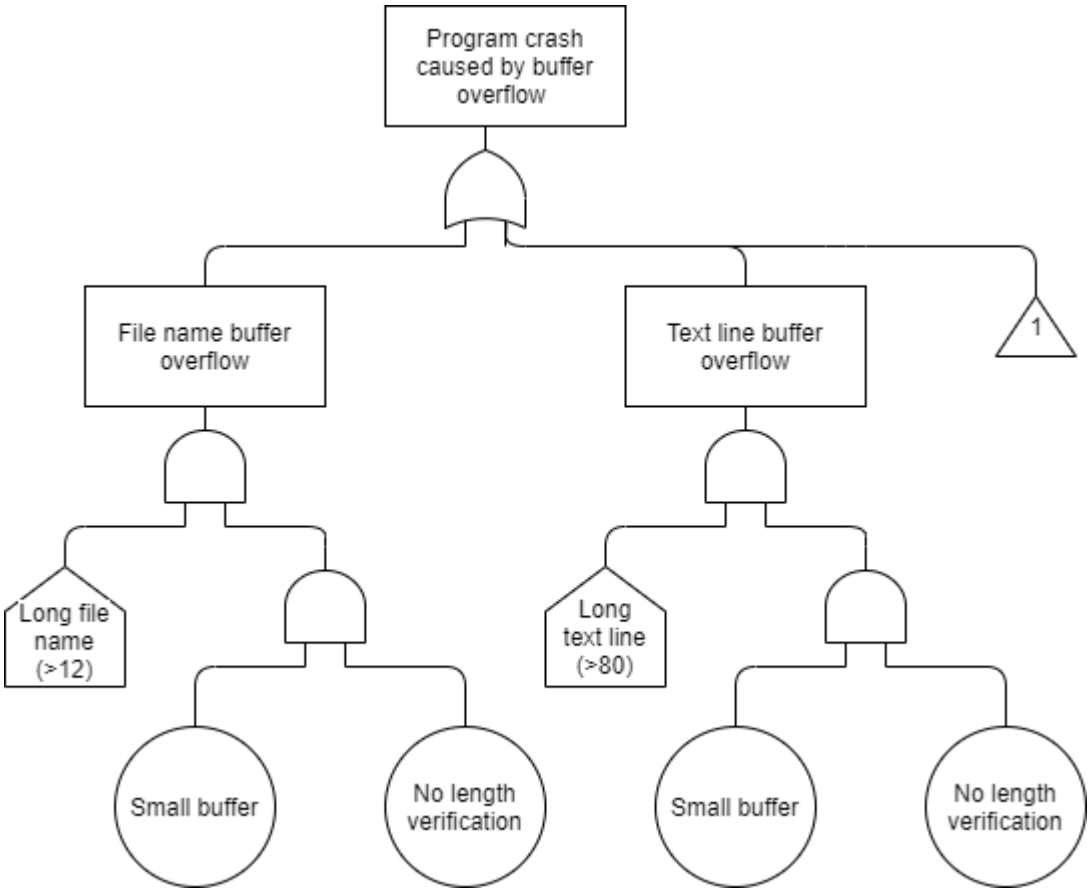


Figure 1. Fault tree – part 1

Fault tree analysis is independent of the programming language and technique used. However, in most cases cannot be independent from the hardware and operating system. The human factor, user inattention or incompetence is often an important factor of the analysis.

FTA can be applied at all stages of the software life cycle, from the preliminary design to maintenance operations, but it is recommended to be performed primarily at the end of software design or coding phases.

2.1 Preliminary risk analysis

The software fault tree analysis is preceded by a risk analysis of the whole system, which identifies undesirable events that could have serious consequences. It is important not to get lost in the details here, the analysis team must have a clear boundary to identify which issues are safety critical. However, one needs to be aware of the fact that in a complex system, not all threats can be identified in advance, and thus the effectiveness of the method strongly depends on the knowledge and expertise of the FTA team.

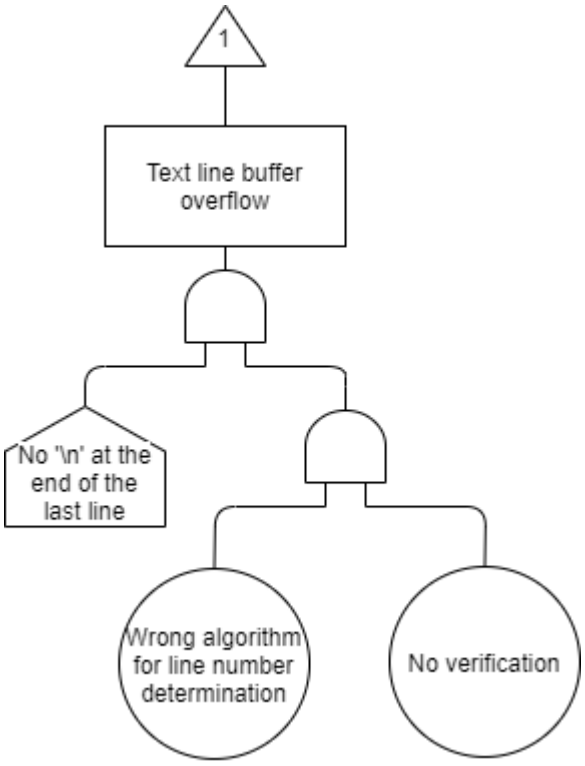


Figure 2. Fault tree – part 2

To illustrate the concepts that arise in explaining the method, consider a simple program that contains file operations and dynamic memory allocation. The task of the software is to read the content of a text file and display it on the console. The *main()* function first asks the user for the file name and then calls the *Load()* function, which opens the file, counts the new line characters, allocates memory for an array of pointers that will store the addresses of the character arrays storing the individual text lines. They are read from the file in a while loop. The *Load()* function returns with the address of the pointer array and its size (number of lines) becomes also available thanks to the reference type first argument of the function.

```
#include <stdio.h>
#include <string.h>

typedef char* text;

text* Load(int& i, char *fn)
{
   char z[81];
   FILE* fp;
   errno_t err;
   text* T = NULL;
   err = fopen_s(&fp, fn, "r");
   if (err == 0)
   {
      int no = 0;
      int c;
      while ((c = fgetc(fp)) != EOF)
         if (c == '\n') no++;
      T = new text[no];
      i = 0;
      rewind(fp);
      while (fgets(z, 81, fp) != NULL)
      {
         T[i] = new char[81];
```

```
            strcpy(T[i], z);
            i++;
         }
         fclose(fp);
      }
      return T;
   }

   int main()
   {
      printf("File name: ");
      char fn[13];
      gets_s(fn);
      int no=0;
      text* T = Load(no,fn);
      if(T!=NULL)
      for(int i=0;i<no;i++)
        printf("%s", T[i]);
      getchar();
      return 0;
   }
```

During the preliminary risk analysis, the team conducting the analysis highlighted the threat described as "*Program crash caused by buffer overflow*". This threat will be considered as main event (root of the fault tree) in the next step.

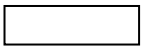
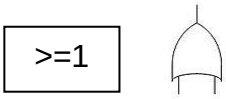
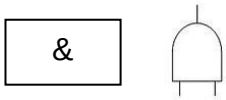
2.2 Exploring cause and effect relationships

The starting point for the analysis is the list of hazards identified in the preliminary risk analysis. In the general case it can contain several threats. These are processed one by one, assuming the occurrence of the main event they define. For each hazard on the list an individual fault tree will be constructed, and so several fault trees are developed in parallel or in sequence. Starting from the root cause (the main event), the events or deficiencies that caused the main event are identified and then, one by one, these are explained, using a recursive technique, to arrive at a detailed understanding of all the causes. In more complex situations, a fishbone (Ishikawa) diagram can be used to make the work systematic and transparent.

An event can be triggered by several conditions. If any of these conditions alone can cause the event, they are linked to the event through a logical "or" gate. If the failure occurs only when all conditions are satisfied simultaneously, the linkage is made through a logical 'and' gate. This extension shall continue until further explanation/development of the event is no longer possible. The extension will also stop if the occurrence of an event is the effect of a hardware failure that is independent of the software. The graphical representation of the fault tree applies the symbols presented in Table 1.

In our example, a careful analysis of this small software shows that a buffer overflow can occur in the following three scenarios: (1) after the user enters the file name, (2) after reading a text line from the file, and (3) after reading the last text line from the file.

Table 1. Graphical symbols [5]

Symbols	*Description*
	Description of the subsystems and events
	OR gate, the output occurs if any input occurs
	AND gate, the output occurs only if all inputs occur
	Basic (primary) failure event
	External (normal) event
	Undeveloped event
	Transfer symbol

The events represented by the leaves of the tree, can be divided into three classes:

- Basic (primary) failure event. It can be a random event. It does not require any further expansion. It is a leaf node in the tree.
- External (normal) event. Is does not represent a fault, it is normally expected from the system.
- Undeveloped event. It has no consequences or there is not enough information about it. It is not further considered in the analysis.

A primary event is a failure that occurs under the prescribed operating conditions. Its cause lies in the design or coding of the software module (component). The identification of all primary events is one of the most important goals of FTA.

After identifying of all the conditions that can trigger the main event the next step is a qualitative and/or quantitative analysis of the fault tree. The term quantitative analysis means that based on the probabilities of occurrence of the basic, external and undeveloped events one determines the corresponding probabilities of the intermediate events as well as the probability associated to the main event. This helps to determine the real threat represented by the individual events. Unfortunately, the needed probabilities are seldom available especially in case of new software products. Therefore, a qualitative analysis is usually carried out where the main focus is on identifying the minimal cut set, i.e. the smallest set of basic events that can trigger the main event.

The last step of FTA is the reduction or elimination of the risks that lead to the main event. Here the usual solution is either to include verification measures into the code to detect the existence of conditions in time to allow intervention or to redesign the code. Verifications can be done through condition checks and feedback, which are commonly used in traditional programming, or through exception handling, which is a popular technique in object-oriented approach.

3 Conclusions

Fault tree analysis provides a systematic, deductive approach to identify all the flaws of a program that can lead to the occurrence of a non-desired safety-critical event, the so called main event. This kind of search for safety-critical code represents a very small fraction of the whole work invested in the software project, and the associated costs are negligible compared to the costs of testing and verification. The resulting code is also much more robust.

Further research will consider the application of different computational intelligence methods (e.g. [3][7][9][11]) during the analysis.

Acknowledgment

This research is supported by EFOP-3.6.1-16-2016-00006 "The development and enhancement of the research potential at John von Neumann University" project. The Project is supported by the Hungarian Government and co-financed by the European Social Fund.

References

[1] ***: NASA-GB-8719.13: NASA Software Safety Guidebook - 8000 - Safety, Quality, Reliability, Maintainability. Available at: https://standards.nasa.gov/standard/nasa/nasa-gb-871913 [Accessed: Aug. 9, 2021].

[2] Ericson, C. "Fault Tree Analysis - A History" Proceedings of the 17th International Systems Safety Conference. 1999. Available at: https://standards.nasa.gov/standard/nasa/nasa-gb-871913 [Accessed: Aug. 9, 2021].

[3] E.H. Guechi, J. Lauber, M. Dambrine, G. Klančar and S. Blažič (2010): PDC control design for non-holonomic wheeled mobile robots with delayed outputs, Journal of Intelligent and Robotic Systems, vol. 60, no. 3-4, pp. 395-414, Dec. 2010. https://doi.org/10.1007/s10846-010-9420-0

[4] Helmer, G., Wong, J., Slagell, M., Honavar, V., Miller, L., and Lutz, R.: A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System, 1st Symposium on Requirements Engineering for Information Security, Indianapolis USA, 2001. pp. 63 - 74.

[5] Kabir, S. (2017). An overview of fault tree analysis and its application in model based dependability analysis. Expert Systems with Applications, 2017, 77, 114–135. https://doi.org/10.1016/j.eswa.2017.01.058

[6] Koncz, A., Pokorádi, L., Szabó, G.: Failure Mode and Effect Analysis and Its Extension Possibilities, Repüléstudományi Közlemények, 2018, 30 : 1 pp. 247-254.

[7] Sz. Kovács, D. Vincze, M. Gácsi, Á. Miklósi and P. Korondi, "Fuzzy automaton based Human-Robot Interaction," 2010 IEEE 8th International Symposium on Applied Machine Intelligence and Informatics (SAMI), 2010, pp. 165-169, https://doi.org/10.1109/SAMI.2010.5423746

[8] Leveson, N. G.: Safeware: System Safety and Computers, Addison-Wesley, New York, 1995.

[9] R.-E. Precup and S. Preitl, Development of fuzzy controllers with non-homogeneous dynamics for integral-type plants, Electrical Engineering, vol. 85, no. 3, pp. 155-168, Jul. 2003. https://doi.org/ 10.1007/S00202-003-0157-7

[10] Signoret J.P., Leroy A. (2021) Fault Tree Analysis (FTA). In: Reliability Assessment of Safety and Production Systems. Springer Series in Reliability Engineering. Springer, Cham., pp 209-225, https://doi.org/10.1007/978-3-030-64708-7_16

[11] J. Vaščák and M. Rutrich: Path planning in dynamic environment using fuzzy cognitive maps, Proceedings of 6th International Symposium on Applied Machine Intelligence and Informatics (SAMI 2008), Herľany, Slovakia, 2008, pp. 5-9. https://doi.org/ 10.1109/SAMI.2008.4469153