

FTAB -- A New Generation Computer Code For Fault Tree Analysis

Clifton A. Ericson II; The Boeing Company; Seattle, WA.

15th International System Safety Conference, Aug 12-17, 1997

Abstract

Fault Tree Analysis (FTA) is a tool for analyzing and evaluating failure paths in a system. Since the inception of FTA, fault tree theory, methods and computer codes have improved significantly. Fault tree (FT) computation has moved from mainframe computers to desktop computers and fault tree plotting has transitioned from specialized mainframe plotters to common desktop laser and inkjet printers. Now, the Fault Tree Analyzer and Builder (FTAB) computer program has been developed that utilizes new programming language features to store a fault tree in computer memory as a virtual dynamic abstract tree structure. This approach provides the advantage of being able to traverse the tree with fast and efficient algorithms and dynamic memory allocation to build a tree of any size.

FTAB is a single package program with many built-in capabilities for both beginning and advanced FTA. It is an interactive FT editor, a computation engine for generating Cut Sets, probabilities and sensitivity calculations and a FT graphics and report generator that produces quality plots and specialized reports. FTAB is being used by several different projects at Boeing, including the B-2 program, the 767 AWACS program and the Minuteman missile guidance system. This paper discusses the design and operation features of FTAB, along with its capabilities and benefits.

FTA History

FTA was developed by H. Watson of Bell Labs circa 1961 for application on the Minuteman Launch Control System [ref. 1]. It was then used by the Boeing Company for complete quantitative safety analysis of the Minuteman weapon system. The technique was enhanced by Boeing and is currently used on many Boeing products. The first FT computer codes were developed by Boeing. The nuclear power industry then discovered the benefits of FTA and

have enhanced and refined the FT technique even more. It appears that the robotics industry is now beginning to become highly involved with FTA.

FTA Purpose

FTA is a failure oriented, top-down, analytical technique in which a single Undesired Event (UE) of a system is specified (the top event) and the system is analyzed to find all of the credible ways in which this top UE can occur. The primary purpose of FTA is to identify failure paths leading to an undesired event, and to quantify the probability of the undesired event (if desired). It is a deductive analysis tool or technique for rigorously analyzing a problem, establishing all the causal factors and their relationships and documenting the process. FTA is not a hazard analysis, it is a root cause analysis technique for evaluating the causes and probability of an identified hazard (ie, the top UE).

FTA is a combination of logical design, symbolic logic, Boolean algebra, reliability analysis and probability theory [ref. 2]. A FT provides a concise and orderly description of the various combinations of possible occurrences within a system which can result in a predefined "undesired event" [ref. 3]. It also provides a measure of the relative level of safety inherent in any particular configuration. A FT is a technique for diagramming the flow of events through a system as they relate to a particular event being studied [ref. 4]. FTA is the translation of a physical system into a structured logic diagram of cause-effect relationships leading to an UE.

The main attributes of FTA can be summarized as:

- 1) A graphical display of failure paths and cause/effect relationships.
- 2) A formally correct logic model for quantifying the probability of an event.

- 3) A means for demonstrating compliance with requirements.
- 4) A tool for evaluating, verifying and justifying design changes.
- 5) An approach for determining common mode failures.
- 6) A methodology for finding diverse multiple failures causing an UE.
- 7) A comprehensive root cause analysis tool.

The primary purpose of FTA is to assist system safety and reliability engineering in preventing system problems before they occur, by identifying problem causes and predicting their probability of occurrence. FTA helps manage complexity by breaking a complex system into simpler subsystems and then into components. It also diagrams how failures in different subsystems can combine in unexpected ways to cause an UE. Once constructed, the FT can be evaluated either qualitatively or quantitatively, depending on the intent of the analyst.

Why FTAB ?

Why develop a new code when many good codes are available already? We found at Boeing that the existing codes did not meet all of our specific needs, not even existing Boeing codes. Therefore, a new code was developed around the specific desires of experienced Boeing FTA analysts. In addition, new computer science techniques were applied to improve memory utilization and cut set computation.

FTA Definitions

This paper is not intended as a FTA tutorial, but a brief set of definitions and mechanics are provided since precise terminology is needed for describing the FTAB code. See references 5 - 11 for more detailed information. The following standard definitions include:

- a) Tree - A FT containing all of the necessary and sufficient logic and events to cause the top UE.
- b) Tree Top - The single top UE under investigation. A FT can only have one top.
- c) Branch - A portion or segment of tree containing logic gates and events.
- d) Module - A subtree or branch whose terminal event (subtop) does not occur elsewhere in the tree. An independent subtree or tree branch.

- e) Event - Originally this term meant a basic component failure event. Over time the term has been corrupted to now refer to any event on the tree -- failure, gate, condition.
- f) Gate - A logical Boolean operator that combines input events in a specific way.
- g) MOE - A Multiple Occurring Event or failure mode that occurs more than one place in the FT. Also known as a redundant event, that causes branch dependencies.
- h) Cut Set (CS) - A Set of events that together cause the undesired event to occur (failure path).
- i) Min CS (MCS) - A CS that has been reduced to the minimum number of events and still causes the UE; it cannot be further reduced and still guarantee occurrence of the top event.
- j) Super CS (SCS) - A set that contains a MCS plus additional events to cause the top UE.
- k) Coherent tree - A tree that does not contain NOT or XOR gates.
- l) Noncoherent tree - A tree that contains either a NOT gate or XOR gate. Significant in that more mathematical difficulties are caused by noncoherence.
- m) Prime Implicants - A CS for a noncoherent tree.
- n) Critical Path - The highest probability CS that drives the top UE probability. (The most dramatic system improvement is made by eliminating this CS or reducing its probability.)

Some FT terms are loosely defined, interchangeably used and out of date. This causes some difficulty in developing FT codes, therefore the following more precise definitions are added:

- a) Node - A general term primarily used for tree theory in computer programming. Refers to any event, box, gate, input or condition on the tree.
- b) Basic Event (BE) - A node that is a basic component failure (primary or secondary) or a normal event. The BE's are where the failure rates and probabilities enter the FT. These are circles, diamonds, houses and triangles.
- c) Gate Event (GE) - A node that is a gate operator combining input nodes. These are

- AND, OR, Inhibit, Priority AND and Exclusive OR gates.
- d) Condition Event (CE) - A node that is a conditional event to a GE.
- e) Intermediate Event (IE) - Any node on the tree that is a GE. IE's are the cause-effect relationships that must be developed down the tree, until the BE's are reached.

Figure 1 is an example FT that attempts to show how these definitions relate to an actual FT. Each box on the tree is a node by definition. The top node is known as the root node, the "Top" UE and a GE. All GE's down the tree are also IE's, as well as nodes. At the various tree end points are the BE's, which are also nodes. The idea behind this diagram is to show that all nodes are either a GE, BE or CE, except for the top node. And, all GE's are also IE's. The BE named E2 is a MOE because it occurs more than once in the tree. MOE's cause branch dependencies, which in turn make CS resolution more difficult.

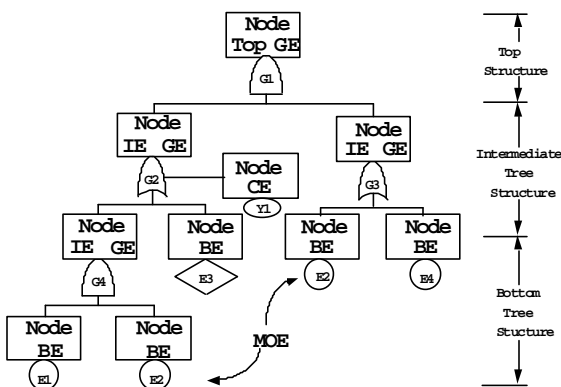


Figure 1 - FT Definitions

FTA Construction Mechanics

Construction of a FT begins with definition of the "top" Undesired Event (UE). This is the system hazard or failure of major interest, generally obtained from a hazard analysis. The causes of the top event are determined and connected to the top event by a logic gate, which thereby creates the second level of the FT. The procedure is then repeated for each of the logic gates on the second level, and is iteratively repeated until all events have been fully developed. The events are considered fully developed when all cause-effect questions have

been resolved. All paths must end at the component failure mode or BE level.

FTA is an iterative process where each IE is the effective top of a branch, which is itself then further developed. This process is repeated until all factors and events contributing to the item of interest are accounted for. The tree grows as each event is expanded by its causal factors. Only failure related events need be considered; the analysis need not consider events or failures which have no effect, or operator intervention to prevent the UE.

The tree top-down path shows causal relationships, while the bottom-up path shows effect relationships.

FT construction uses the following operators:

- a) OR Gate - A GE where the output occurs if and only if at least one of the inputs occur.
- b) AND Gate - A GE where the output occurs if and only if all of the inputs occur.
- c) Inhibit Gate - A GE where the output occurs if the input event occurs and a condition is satisfied.
- d) Exclusive OR Gate - A GE where the output occurs if only one of the inputs occur.
- e) Priority AND Gate - A GE where the output occurs if all of the inputs occur in a specific sequence.
- f) Condition - A CE where a conditional restriction or probability is placed on a sequence of events.
- g) Primary Failure - A BE that is a basic component failure mode (e.g. resistor failed open).
- h) Secondary Failure - A BE that is a failure event that could be further developed if desired, but left at a higher level for convenience (e.g. transmitter fails).
- i) External Event - A BE that is normally expected to occur, and is necessary in the failure path (e.g. power is applied to circuit).

For these FT operators, the following are standard FT symbols:

- a) Circle - Represents a primary failure (BE).
- b) Diamond - Represents a secondary failure (BE).
- c) House - Represents an external event (BE).
- d) Rectangle - Originally in FTA the rectangle signified an intermediate GE. At that time

node words were placed inside each symbol. With the advent of computer plotting, node words were placed only in rectangles, and the symbol was made smaller and placed under the rectangle. Therefore, the rectangle today really has no meaning, other than symbolizing a tree node, and as a place-holder for node words.

e) Oval - Represents a condition (CE).

The Need For Computer Codes

Historically, computer codes have greatly assisted in FTA. As FTA has grown in usage, so has the size and complexity of the applications. Very small FT's (less than 50 events) can be easily drawn and computed by hand. But, as trees grow in both size and complexity (MOE's, multi-phasing, etc.) all aspects of FT development become significantly more difficult. With computers FT's can be created, edited and modified with little effort, when compared to hand drafting each tree every time a change is made.

Large scale and complex FT's containing MOE's are difficult and time consuming to solve (determine CS's and probability) by hand, and often require computers just for a solution. Multi-phase FT's are practically impossible to solve except by computer. Without FT computer codes, FTA would not be a viable tool for large complex systems.

What FT Computer Codes Should Include

Many codes exist today for FTA. Each particular code has its own unique set of attributes, both positive and negative. A complete FT computer code must perform four basic tasks in order to be useful:

- 1) Editing - creating, editing and modifying FT structure and data.
- 2) Logical Evaluation - compute CS's.
- 3) Probabilistic Evaluation - compute probabilities and importance measures.
- 4) Documentation - preparing quality plots and reports.

In addition, there are several other basic properties that a FT computer code should possess, such as:

- 1) Speed for resolving large FT's.

- 2) Memory for handling large and complex FT's.
- 3) Visual aesthetics for the user.
- 4) Simple to use for non-computer people.

FT evaluation can be done either qualitatively or quantitatively, depending on the intent of the user. There are essentially three methods for FT evaluation, with only the first two providing CS's:

- 1) Simulation: direct, Monte Carlo [refs. 4, 9].
- 2) Deterministic (exact solution): Boolean reduction, tree reduction [ref. 11].
- 3) Approximation: upper/lower bounds, upward flow of gate probabilities [ref. 11].

What The FTAB Code Does

From the start, the FTAB program was designed with the primary goal of being an independent FT workstation that provides all of the necessary tools for the FT analyst in a single package. To do this it must contain the necessary computer code properties and qualities described above. Figure 2 shows the basic FTAB functions and their relative relationships.

FTAB contains the following major features:

- a) Simple to perform FT construction, editing and modifying.
- b) CS Computation.
- c) Probability, importance and sensitivity computation.
- d) Quality FT plots for documentation.
- e) FT reports (node data, CS's, probability, importance measures).

In addition, FTAB includes the following attributes:

- a) Easy for non-experts to use.
- b) Virtual Dynamic Tree (VDT) for speed and controlled memory usage.
- c) Handles MOE's.
- d) Windows 3.1 and 95 compatible.
- e) Single executable program with no additional modules.

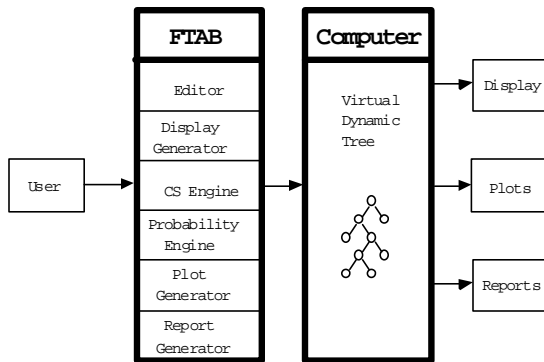


Figure 2 - FTAB Functions

FTAB Design

The overall design of FTAB is based on requirements generated in five basic areas. These particular requirements are the basis for FTAB attributes. Each of the following areas is further discussed below:

- 1) Computer language.
- 2) Aesthetics.
- 3) Plotting.
- 4) User.
- 5) FT computation.

Design Requirements -- Computer Language

FTAB was originally written in Pascal, and then converted to a Windows environment using Delphi. Special capabilities provided by the Pascal language were the real basis for FTAB's unique design. The language unique features provide the following capabilities:

- 1) Tree theory creating a Virtual Dynamic Tree (VDT) structure.
- 2) Actual abstract tree structure in memory.
- 3) Can add/delete nodes dynamically.
- 4) Recursion provides actual tree traversal, quickly and efficiently.
- 5) Dynamic memory/no fixed arrays, only uses memory needed.
- 6) Uses pointers, linked lists and records.
- 7) Each node carries its own data.

The way a computer "sees" or interprets a FT is totally different than the way a user does. In the past, about the only way to store a FT structure in a computer was by using arrays, either fixed size or variable size, depending on the programming language used. Fixed size meant

the maximum size of the FT was fixed from the start by the programmer and computer limitations. Variable size meant the arrays could be increased in size as the user wanted a larger FT, but it was a difficult process and also limited by computer memory.

FTAB structure is based on tree theory, which computer scientists have developed, and is now provided for by some of the newer computer languages, such as C, Pascal and Ada. This approach allows a FT to be stored in memory by special links (pointers), such that the FT can actually be abstractly visualized as a multiway tree structure. This is accomplished by the use of pointers, linked lists and record structures. Tree theory provides the capability for a FT to be visualized and operated as a Virtual Dynamic Tree (VDT) structure. This methodology means the tree size is not limited by the programmer at design time, but only at run time by the amount of memory available. A tree can be increased or decreased in size by the user, and a corresponding amount of memory is utilized. Tree nodes can be easily inserted or removed from the tree, and the memory and tree structure is adjusted automatically. Special algorithms using recursion make tree traversal quick and efficient.

Besides size limitations, another problem with the arrays approach is trying to abstractly visualize a tree structure. There is no natural visual representation for trees using arrays, and programming becomes difficult and unnatural. Once a tree structure is placed in array cells it becomes difficult to insert new nodes and remove existing nodes from a programming standpoint. Arrays wind up with gaps and loss of sequence in nodes. Tree theory provides a natural representation of a tree structure in the computer, thereby simplifying programming and use. The array method also has difficulty handling MOE's, but this is a relatively simple task with tree theory.

In tree theory, each node is a "record" structure, where a record is like a miniature database containing all of the information associated with the node, such as name, type probability, etc. This means that a separate database is not necessary. Whenever information is needed on a node, the program goes down the tree to that node, not a separate database. My own

experience has shown that a common database for tree components is really seldom ever needed or even used. Component databases for many FT's is nice in theory, but often clumsy and difficult in practice. Each FT is a separate project. If common components are used in different projects, the data is just copied from one project to the other. Common databases can create a lot of confusion and extra computer overhead.

Figure 3 shows an example FT in standard FT format. The same tree is also depicted in Figure 4 as it is visualized in tree theory by linked pointers and records. The tree is traversed by traveling up and down the node pointers. Different recursive tree algorithms exist for visiting the tree nodes in different sequences.

Figure 5 shows the three recursive traversal algorithms: prefix, infix and postfix, and the sequence for visiting the tree nodes. These methods are called "walking the tree". Recursion, is only allowed in some computer languages, is where a procedure or subroutine keeps calling itself. This saves in design complexity and is generally faster in execution time and efficiency.

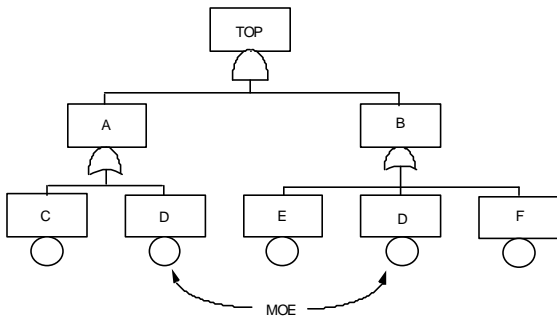


Figure 3 - Example FT Structures

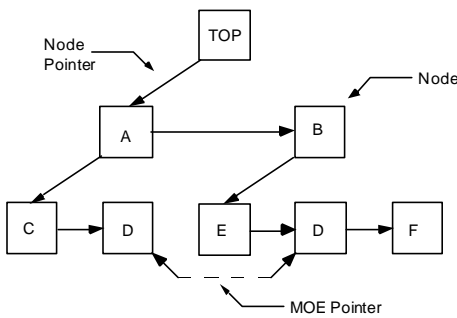


Figure 4 - VDT Structures

TOP	C	D
A	D	C
C	A	F
D	E	D
B	D	E
E	F	B
D	B	A
F	TOP	TOP
Prefix Walk	Infix Walk	Postfix Walk

Figure 5 - Recursion Methods

The MOE pointer in Figure 4 shows how MOE's are linked together. This link helps in speeding the CS substitution and reduction process.

Design Requirements - Aesthetics

Much of FTAB design was driven by aesthetic requirements for standardization and consistency in the plot layout. This means the user could expect a specific layout with no surprises. All tree layout is designed by the user, not arbitrarily made or changed by the computer. The following requirements were responsible for many design decisions:

- 1) 11x17 inch standard page size.
- 2) 10 nodes across and 8 levels down per page.
- 3) Consistent tree layout: fixed symbol sizes, fixed spacing.
- 4) Five (5) character symbol names.
- 5) Wording in a rectangle consists of 4 lines at 20 characters per line.
- 6) Component layout determined by designer, not computer.
- 7) Entire tree can be scaled up or down when plotted.
- 8) Gate node must be above one of its input nodes when displayed.

The five character symbol name limitation is a significant design decision. However, I have found from many years of FT experience that short symbolic names are actually easier to use than long names representing subsystem names. This is especially true with very large FT's from 1,000 to 10,000 events in size. It is much easier to code subsystems using alpha-numeric sequences, such as defining that X1 through X100 represent BE's and G1 through G300 represent GE's on the flight control system, rather than names like flt_cntl_res_sA_153. On Minuteman FT's we found it very easy to name 10,000 nodes with only 5 characters. The short

character name also helps to reduce computer memory needs and actually make it easier to recognize, visualize, sort and identify nodes. It is important that the naming convention be established prior to the actual FTA construction effort.

Design Requirements - Plotting

One of the major objectives of FTAB was to provide the capability for high quality FT plots for documentation. To meet this goal the following plotting requirements were established:

- 1) Plot structure exactly as shown on screen with no computer surprises.
- 2) Uses standard FT symbols.
- 3) Plot tree in pages.
- 4) Use HPGL graphics language for plotters, lasers, ink jets.
- 5) Provide 3 headers and 2 footers on page if desired.
- 6) Titleblock - six different styles.
- 7) Notes - 10 notes per page.
- 8) Two styles for tree layout - balanced and condensed.
- 9) Plot in landscape mode only.
- 10) Use Windows graphics for all Windows printer plots.

Design Requirements - User

FTAB provides the user with the capability to interactively construct and edit the FT, in a graphic environment. The FT visually grows and changes on the computer screen as the user edits the tree. The following user needs were a major factor in FTAB design:

- 1) Menu driven.
- 2) Intuitive operation.
- 3) Simple and easy to use for non-computer type person.
- 4) Allow user to see as much tree structure as possible on screen view.
- 5) Graphical user interface, with text edit capability.
- 6) Try to prevent input errors (traps and warnings).
- 7) Export tree structure and data for other user needs.
- 8) Never have to leave graphical environment to plot or compute CS's.

- 9) Two styles for tree display -- large node box and small node box.
- 10) Capability to scroll tree horizontally and vertically on the screen.

Providing the capability for the user to see as much tree structure as possible on the screen has its advantages and its disadvantages. The advantage is that the more nodes the user can see at one time the better he is able to relate to the tree. The disadvantage is that to do so requires sacrificing text and symbols on the tree structure.

This obstacle has been overcome by using two different tree views. The user can select Large Node Box mode, which displays a complete FT node box with four lines of text, a gate/event symbol and node name. Or, the user can select Small Node Box mode, which only displays a small box with the node name in it. The node type and text are displayed at the top of the screen when a specific node is selected. This gives the user the option to visualize the tree pretty much as it would be plotted, but with fewer nodes visible at any one time. Or, the user can see more tree nodes at one time, but with less information visible.

In addition to the graphical construction or edit of a FT, FTAB also provides the capability for the user to edit a FT via a text file. An FTAB FT is stored in a text file that adheres to a strictly defined format. By following the format conventions the user can edit the text file, using a text editor, and construct or edit a FT. The file can then be read into FTAB and be graphically displayed.

Design Requirements - FT Computation

One of the main parts of any FT program, and one of the most difficult to program, is the CS evaluation function. In designing the evaluation section, the following considerations were made:

- 1) Utilize VDT structure.
- 2) Use modified MOCUS algorithm for CS generation.
- 3) Avoid intensive reduction techniques (e.g., Boolean reduction).
- 4) Easily resolve MOE's.
- 5) Provide accurate correct CS cutoff by order and/or probability.

6) Include a CS editor.

Fast, efficient and accurate CS generation is difficult for large complex FT's. Many top-down and bottom-up reduction methods are too slow and/or inaccurate for very large trees. Some programs merely compute a top probability by various algorithms (e.g., module reduction), but this does not satisfy the needs of the analyst. The CS's must be generated in order to know where and how to fix the system being analyzed. Sometime the top probability cannot be computed until MOE dependencies are resolved, which requires CS's. CS content (components) define the items needing fixing, while CS probability establishes the priority for fixing items.

FTAB uses a modified MOCUS (Method of Obtaining CUt Sets) algorithm called Continuous Substitution And Reduction (CSAR) that takes special advantage of tree theory and recursion for CS generation. Using the tree pointer links, MOE pointer links and recursion simplifies the substitution and reduction process. The MOCUS algorithm, demonstrated by Figures 6 and 7 [ref. 11], is a top-down methodology. The algorithm starts with the top gate, then each gate is iteratively substituted with its inputs, until all the gates have been eliminated, and only BE's remain. Then, the CS's are reduced such that only MCS's remain, and all SCS's have been eliminated.

In very large FT's the number of CS's can become too large to work with. In this situation the number of CS's can be reasonably reduced by order cutoff and probability cutoff, which allows concentration on the more important CS's.

1	2	3	4	5	6
G1	G2, G3	1, G3	1, 1	1	1
			1, 3	1, 3	
		2, G3	2, 1	1, 2	
			2, 3	2, 3	2, 3
				↑ SCS	↑ MCS

Figure 7 - MOCUS Algorithm

In Figure 7, step 1 lists the top gate to initiate the process. In step 2 the top gate is replaced with its inputs. Since it is an AND gate, all inputs are listed together and separated by a comma. In step 3, the first gate is replaced by its inputs. Since it is an OR gate it increases the number of CS's. In step 4 the second gate is replaced with its inputs. Step 5 shows all of the CS's for this FT. In step 6 the SCS's are reduced, and only the MCS's remain. Notice that OR gates expand the number of CS's and AND gates increase the number of elements within a CS.

FTAB Operation

FTAB is a fully interactive program with full screen dialog and selection menus, consistent with Microsoft Windows format. It utilizes both the mouse and the keyboard. FTAB has a main menu system for performing master functions which is referred to as the Program Function menu. FTAB has a secondary menu system that is on tabbed notebook pages that provides quick access to the various data edit functions. This menu is referred to as the Data Function menu.

Table 1 lists the Program Functions provided by FTAB, along with the drop-down submenus associated with each function.

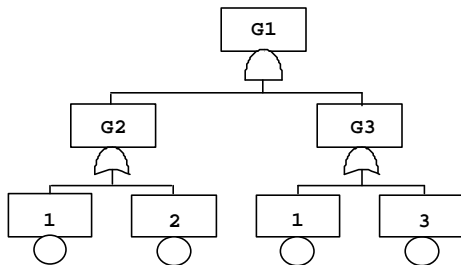


Figure 6 - Example FT

Table 1 - FTAB Program Function Menu

	Function	Drop-Down Menus
1	File	New Tree Open Save Printer Setup Export Data Exit
2	Edit	Copy Node Paste Node Find Node
3	Plot	Plot Tree (Pages) Plot Titleblock Only Plot CP Tree Batch Plot
4	Report	Screen Tree Plot Entire Tree Plot (Pages) Tree Structure Plot BE Data GE Data MOE List Node Data
5	Options	General Plot Report CS
6	Evaluation	Run CS Edit CS Run Prob
7	Help	Help About

FTAB provides FT construction and editing via two different menus. The most used tree edit functions, shown in Table 2, are obtained from a pop-up menu that appears when the right mouse button is pressed. Any of these functions can then be selected and the effect is immediately made to the FT, both on the user's screen and in computer memory. When a node is created by Add Node, it is given a pseudo-name by the program and appropriately placed on the tree. The user then uses the Change Name function to input the desired name. This methodology allows the user to build an entire FT structure before entering node names, or to enter names as each node is created.

Once a node is created the Copy and Paste functions on the Program Functions menu can be used to copy identical data from one node to another. This is only for node data, and not for node name or type. Node data is considered as node description, probability, exposure time, labels, notes, etc.

In order to provide as much tree structure information to the user as possible, only node

boxes, node names and structure lines are shown on the screen (Small Node Box mode). Relevant node data is shown at the top of the screen for a selected node:

- 1) Node name.
- 2) Node type.
- 3) Number of MOE's.
- 4) Node description.

This data is also displayed and ready for edit when the Node tab is selected. The user can switch back and forth between tree edit and node edit screens, or stay on the tree edit screen.

Table 2 - Tree Edit Functions

	Data Functions	Action
1	Add Node	Create and add node to FT.
2	Change Name	Change node name.
3	Move Node	Move node elsewhere in tree.
4	Delete	Node, Tree Branch
5	Insert	Gate Above, Gate Below, Condition, SubTree
6	Reorder Node	Change order of inputs to a gate.
7	Dup Subtree	Duplicate a tree branch.

Various program options can be changed from the Options menu. When Options is selected, it brings up another Tabbed Notebook with pages for General, Plot, Report and Cut Set options.

The Data Function menu is immediately below the Program Function menu. The Data Functions are brought up for edit by selecting the appropriate notebook tab. When a tab is selected the form for that Data function immediately appears and is ready for user input. The user can switch between data edit forms at any time. Table 3 lists the Data Functions, on the notebook tabs, and their purpose.

Table 3 - FTAB Data Function Menu

	Data Functions	Action
1	Tree	Edit FT structure.
2	Node	Edit FT node data.
3	Notes	Edit plot notes.
4	Titleblock	Edit plot titleblock.
5	Header/Footer	Edit plot headers and footers.

The FT node data is brought up by clicking on the notebook *Node* tab, after selecting a tree

node with a single mouse click. Alternatively, the user can do the same thing by double clicking the tree node. This action switches to the Node Edit screen, where the data for each node is created, edited and modified. The buttons on the side are used to save the entered data, clear all items to blank or return the original data (if not previously saved). There is also a Find button to search for a particular node and bring up its data for edit. This saves the user from returning to the tree screen and selecting the desired node.

Summary

FTAB is a new FT code based upon VDT theory. It is a single package FT workstation that provides all the necessary tools for the FT analyst: 1) graphic editing, 2) CS and probability computation, and 3) graphic output. Due to its unique design, FTAB is fast and efficient.

Currently FTAB exists in two forms: the FTAB1 program and the FTAB2 program. FTAB1 is a more limited program, intended primarily for creating, editing and plotting FT's. FTAB1 is used at Boeing to develop FT's which are input into a Boeing multi-phase simulation program called SAFTE. FTAB1 does handle large multi-page FT's.

FTAB2 is a full scale FT program that is both a FT builder and a FT analyzer. It handles large multi-page FT's and performs CS evaluation and probability calculations. Table 4 provides some sample test results from FTAB2 for CS generation for three different FT structural complexities. These tests were performed on a Pentium 120 Mhz IBM PC. Testing to date has not been extensive, and these times are not CPU times, but the measured time from hitting the run button to having a "done" message appear on the screen, which included generating the CS and putting it in the printer queue.

Table 4 - FTAB2 Results Summary

Size	Structure	No. Of CSs	Time
111 GE's 500 BE's	Simple (OR only)	500	4 sec.
111 GE's 500 BE's	Moderate (AND/OR)	460	4 sec.
115 GE's 500 BE's 8 MOE's	Complex (AND/OR/ MOE's)	462	4 sec.

Future FTAB enhancements are planned for:

- 1) Tree plots using PostScript.
- 2) Direct edit of Large Node Box data.

References

- [1] H. A. Watson, Launch Control Safety Study, Section VII Vol 1; Bell Labs; Murray Hill, NJ, 1961.
- [2] A. B. Mearns, Fault Tree Analysis : The Study Of Unlikely Events In Complex Systems, System Safety Symposim (Boeing/UW), 1965.
- [3] D. F. Haasl, Advanced Concepts In Fault Tree Analysis, System Safety Symposium (Boeing/UW), 1965.
- [4] P. M. Nagel, Importance Sampling In System Simulation, Annals Of R & M Symposium, 1966, p330-337.
- [5] C. A. Ericson II, System Safety Analytical Technology -- Fault Tree Analysis, Boeing document D2-113072-2, 1970.
- [6] R. E. Barlow & J. B. Fussell & N. D. Singpurwalla, Reliability And Fault Tree Analysis, Conference On Reliability And Fault Tree Analysis; UC Berkeley; SIAM Pub, 1975.
- [7] J. B. Fussell, Fault Tree Analysis - Concepts And Techniques, Generic Techniques In Systems Reliability Assessment, Noordhoff Publishing, 1976, p133-162.
- [8] N. H. Roberts & W. E. Vesely & D. F. Haasl & F. F. Goldberg, Fault Tree Handbook, NUREG-0492, 1981.
- [9] R. E. Altschul & P. M. Nagel, The Efficient Simulation Of Phased Fault Trees, Annuals Of R & M Symposium, 1987, p292-296.
- [10] D. J. Mahar & J. W. Wilbur, Fault Tree Analysis Application Guide, Reliability Analysis Center, Report F30602-87-C-0228, 1990.
- [11] E. J. Henley & H. Kumamoto, Probabilistic Risk Assessment And Management For Engineers And Scientists, IEEE Press (2nd edition), 1996.

Biography

Clifton A. Ericson II
 The Boeing Company
 18247 150th Ave SE, Renton, WA 98058 USA
 206-657-5245 (W) 206-657-2585 (Fax)
 email: clifton.a.ericson@boeing.com

Mr. Ericson works in system safety on the Boeing 767 AWACS program. He has 32 years experience in system safety and software design with the Boeing Company. He has been involved in all aspects of fault trees since 1965, including analysis, computation, multi-phase simulation, plotting, documentation, training and programming. He has performed Fault Tree Analysis on Minuteman, SRAM, ALCM, Apollo, Morgantown Peplemover, B-1 and 737/757/767 systems. He is the developer of the FTAB fault tree computer program. He helped start the software safety discipline in 1975 and has written two papers on software safety and taught software safety at the University of Washington. Mr. Ericson holds a BSEE from the University of Washington and an MBA in quantitative methods from Seattle University.