

TRACEABILITY ANALYSIS: MODELING FUNCTIONAL REQUIREMENTS SPECIFICATIONS

Nihal Kececi* and Alain Abran**

**Center for Technology Risk Studies, Reliability Engineering
University of Maryland, College Park,
MD 20742, USA
nkececi@eng.umd.edu*

*** Department of Software Engineering
École de Technologie Supérieure – ETS – Université du Québec
Montréal, H3C 1K3 Canada
aabran@ele.etsmtl.ca*

Abstract: Traceability analysis is recognized as a concern in an increasing number of standards and guidelines for requirements engineering. However, there are many challenges in applying current traceability approaches to complex and dynamic software development processes. In this paper, functional traceability is introduced, and difficulties related to implementing traditional traceability methods are discussed. To address these challenges, we propose a model aimed at building functionality into a logic-based graphical framework in order to define the interrelationships between software-based system components, functions, and sub-functions, as well as the interrelationships between software life cycle phases. The proposed model provides functional traceability for a large-scale software development process. The architecture of the Functional Traceability Model (FTM) captures both system and software specifications and design attributes into a multi-level hierarchy. Implementation of the model to assess functional requirements traceability is illustrated as a case study.
Copyright © 2004 IFAC

Keywords: Traceability analysis, functional correctness, requirements assessment, verification, large-scale complex system


1. INTRODUCTION

Traceability analysis is a technique that provides a path to the validation and verification of stakeholder requirements to ensure that their needs are met by the systems delivered. Within any system development life cycle, there must exist a mean, so that system requirements, including the functional requirements, can be traced both forward and backward to ensure that the system is being designed and produced correctly. While implementing adequate and full traceability in large-scale complex system development is challenging, it is even more so when the context includes software engineering and development.

Traceability assessment methods establish the relationships between the requirements specifications and the design, matching elements of one to those of the other. Once matching has been completed, all that remains is either a set of unmapped requirements elements or unfulfilled requirements, or a set of unmotivated additional design elements or unintended design functions.. The first clear signal is design inadequacies, and the second raises strong concerns that the non-specified additional functions might lead to unexpected errors.

There are many different views of traceability, all of them changing with the stakeholder's view of the

system. For example, to the *customer*, traceability could mean being able to ascertain that the system requirements are satisfied. The primary concern of the *maintenance engineer* with traceability may be how a change in a requirement will affect a system, what modules are directly affected, and which other modules will experience residual effects. In addition, traceability is critical to the *operation* and *maintenance phase*, in particular when significant stakeholder changes may be made and when required impact analyses must be performed. Although tracing such changes is difficult, the process must nevertheless be carried through to ascertain the full extent of the impact of additions, deletions, or modifications to the system.

In this paper, we review the current traceability  discuss the challenges in implementing traditional traceability analysis to perform the functional traceability process in section 3. In section 4, we introduce the set of traceability definitions. To address the problems associated with the traditional traceability approach, Functional Traceability Model (FTM) is proposed in section 5. The implementation of the FTM is illustrated with a case study in section 6, and the lessons learned in section 7.

2. CURRENT TRACEABILITY DEFINITIONS, METHODS, AND TOOLS

2.1 Traceability definitions

The definitions most commonly cited in the literature are the following:


- (1) *The degree to which a relationship can be established between two or more products of the development process; and (2) The degree to which a requirement and the design of a given software component match* [IEEE Std.610.12].
- "A Software Requirements Specification (SRS) is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation." Forward and backward traceability are also recommended [IEEE Std. 830].
- (1) *A given term, acronym, or abbreviation means the same thing in all documents; (2) A given item or concept is referred to by the same name or description in the documents; (3) All material in the successor document has its basis in the predecessor document, that is, no untraceable material has been introduced; and (4) The two documents do not contradict one another* [DoD-Std-2167A].

In summary, DoD-Std-2167A associates requirements completeness, necessity, and consistency with traceability analysis. IEEE Std.610.12 focuses on the relationships between products of the development process, while IEEE Std. 830, in contrast, establishes a link between

traceability analysis and requirements correctness. Since there is no standard available for defining complete views of traceability, the traceability process is often misunderstood and misapplied, and is seldom performed correctly.

2.2 Techniques

Many techniques have been proposed for providing traceability, including: cross referencing schemes [Evans89]; key-phrase dependencies [Jackson91]; Requirement Traceability (RT) matrices [Davis 90]; matrix sequences [Brown91]; hypertext; integration documents; and more. Some general-purpose requirements verification and validation techniques have also been proposed for tracing, most of which have been developed to directly support textual

 requirements traceability. These differ in the quantity and diversity of information they can trace between, in the number of interconnections they can control between information, and in the extent to which they can maintain requirements traceability when faced with on going changes to requirements. The quality of the resulting requirements traceability, however, depends on the rigid adherence to pre-specified procedures and notations for development. A number of commercially available requirements tools have been developed to support traceability. One of the features common to all tools and techniques is the establishment of links between words/phrases across development life cycle activities to ensure traceability. It has been noted by [Palmer90] that currently available tools are poorly integrated and lack of flexibility.

3. CHALLENGES TO IMPLEMENTING TRACEABILITY ANALYSIS

Many standards for systems and software development recommend the practice of requirements traceability. However, very few guidelines are available on how to establish this traceability or on what should be traced. Some of the challenges to applying traditional traceability approaches are the following:

3.1 Lack of a common understanding of what must be traced

Individual practitioners' understanding of what must be traced leads to diverse applications of requirements traceability. Some examples found in the literature are listed below [Orlena 02]:

- *Purpose-driven* (defined in terms of what the process should do): "the ability to adhere to the business position, project scope and key requirements that have been signed off"
- *Solution-driven* (defined in terms of how the process should be performed): "the ability of tracing from one entity to another based on a given semantic relation";

- *Information-driven* (emphasising traceability information);
- *Direction-driven* traceability (emphasising traceability direction).

3.2 Complexity

In a typical large-scale project, there will be a large number of requirements derived from different sources and expressed at various levels of system detail. This makes it difficult to carry out the mapping when the need for effective mapping is even greater: in such projects, there are more opportunities to generate unmapped or unfulfilled requirements, as well as unmotivated or unintended design functions. Mapping system functional requirements to software functional requirements is a first critical step in the development of software, and, of course, an important potential source of risks when not achieved with the degree of rigor required for safety-critical systems. It is crucial to record and maintain this derivation in order to make it possible for the impact of any subsequent changes to the requirements to be assessed.

In safety-critical systems, such as in a nuclear power plant, software performs two major types of functions. The first type is related to performing the plant safety functions themselves. These safety functions are identified at the system level, and flow down through the software requirements, design, and code. The second type is associated with maintaining the integrity of the system and of the software elements that perform the primary safety functions, such as error checking and fault tolerance. These integrity functions are partially identified at the system level, but are significantly expanded within the software itself. Successful safety system development depends on the ability to trace safety functions at system level and to reflect these in the delivered system. Well-defined traceability analysis methods and tools must play a major role in ensuring that the delivered system is safe and that the delivered system functions properly and as intended.

3.3 Software requirements in natural language

If the requirements are expressed textually in a requirements specification, then a number of validation processes can take place, such as syntactic and semantic checking. Language semantics are needed to ensure that the trace is related to the meaning or context of the requirement or set of requirements, while syntax is necessary to trace to a specific word or phrase, without regard to meaning or context. However, experience has shown that lack of a common terminology and the use of natural language lead to more errors in the requirements phase. Moreover, it was noted in the [Hennell 87] study (contrary to the views expressed in many books [DeMarco 1979, Jackson 1983]) that "*functions cannot be deduced necessarily or exclusively from*

the use of verbs." Further, it is noted that "*there is no known way in which details of functionality can be extracted from natural language text with any degree of certainty.*" The achievement of requirements traceability, more specifically functional requirements traceability, requires verifying functional correctness and completeness. Therefore, the use of natural language to define functionality is the first issue to address in performing traceability correctly.

4. DEFINITIONS FOR FUNCTIONAL REQUIREMENTS TRACEABILITY

System requirements constitute a complex combination of both functional and non-functional requirements. At the present time, no model and no standards are available to support and define functional requirements traceability. To design a framework which allows the fundamental cause of requirement traceability problems to be located and addressed, we first document the definitions to be used in this framework.

4.1 Function

The definition of a function used in this study is taken from IEEE Std. 610.12:

"(1) A function; is a module that performs a specific action, is invoked by the appearance of its name in an expression, may receive an input value, and returns a single value; (2) A module is a logically separable part of a program."

4.2 Functional characteristics

The definition of functional characteristics is adapted from the definition of a function, as mentioned above. The characteristics recognized for functional traceability are: inputs, outputs, process, and interfaces (boundary).

4.3 Functional traceability

The definition of functional traceability is modified from IEEE Std. 610.12, and is defined by the authors as follows: (1) the degree to which a relationship can be established between two or more functions and functional characteristics of the system (user) requirements; and (2) the degree to which a requirement and design of a given software module match.

4.4 Functional analysis

Functions are discrete actions that the system must perform. Functional analysis begins by identifying top-level system requirements and decomposing these functions into a hierarchy of sub-functions to form a functional architecture. The objective is to break the system down into simple tasks which can be performed by people, hardware, and software, and

which, when combined with other sub-functions, will achieve the performance of the top-level system functions [Chapman 92]. Although there are many tools and methods used to analyze system requirements, functional analysis ensures that each system function requirement is traced to a software requirement and then to a software design element.

5. FUNCTIONAL TRACEABILITY MODEL

To perform functional traceability, an FTM (that is, extended-functional modelling) is proposed. FTM is a logic-based analysis method providing a graphical framework to establish traceability for multilevel system components of large-scale complex systems. It is aimed at visualizing functional specifications within the characteristics of functionality such as inputs-processes-outputs using a logic-based graphical framework. Figure 1 illustrates the FTM modelling technique of a function (as defined above), as it transforms textual functionality into graphical form using the following graphical language notation.

1. **External output:** A rectangle on the top represents an output set. An output can be an input to another module, a signal to any hardware action, a functional user¹ requirement, or a monitor requirement.
2. **External inputs:** A set of inputs is represented as a rectangle on the left. User inputs, sensor outputs, system variables, and outputs of another component are examples of an external input set.
3. **Internal inputs/outputs:** The numbered circles represent the output of a sub-process which will be an input to another sub-process within a time sequence, as shown in Figure 1.
4. **Sub-functions:** These are illustrated with decision logic symbols in Figure 1. Mathematical operators and control algorithms are examples for sub-functions.
5. **A module/function:** A rectangle in the centre represents a module or a function in a multilevel hierarchy.
6. **Data movement:** Circular connectors are used to map input data to processing steps, and processing steps to outputs.
7. **Boundary:** This is identified by the entry and exit points of a software module.

A general strategy for functional decomposition is to define the required functional processes as a mapping from inputs to outputs. Ideally, the traceability analysis proceeds in top-down fashion, first identifying the functions associated with the system as a whole. Each level of the hierarchy adds detail about the processing steps necessary to accomplish the more abstract function above, a function which controls the processing of its sub-functions. In a complete decomposition of a functional requirement,

the functional hierarchy specifies the algorithms, the internal inputs needed to achieve these algorithms, and the internal outputs resulting from these algorithms, as well as their interrelationships. Any output of any module can be an input to another module or vice versa.

A summary of these steps, together with the graphical notations used, is presented in Table 2.

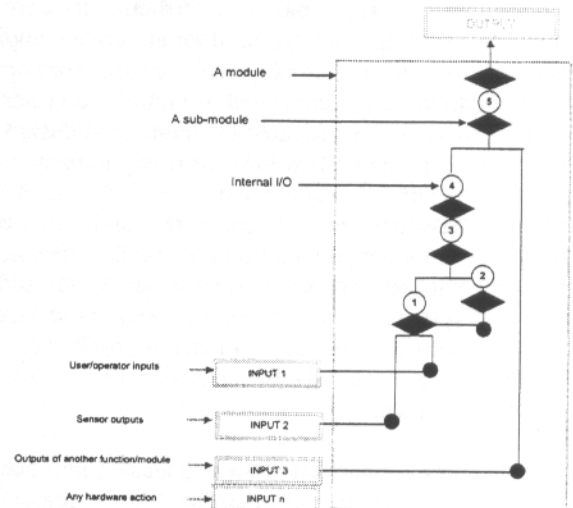


Figure 1. Modelling functionality within the FTM framework

Table 1. Graphical Description of a Functional Requirement

Description	FTP Language	Graphical Language
Function: "A software module that performs a specific action is invoked by the appearance of its name in an expression, may receive input value, and return a single value." Module: "A logically separable part of a program"	Function /Goal/ Process	
Input: To receive data from an external source Output: To transmit data to an external destination	Value of Function External I/O	
Algorithms: Any sequence of operations for performing a specific task	Logic / Algorithms	
Sub-function When a function is decomposed, sub-functions can be identified	Sub- functions/ sub-module/sub-processes	
Internal Inputs and Outputs	Value of sub-functions Internal I/O	
Functional boundary is identifies entry and exit points of a software module Entry: A point in a software module at which execution of the module can be begun Exit: A point in a software module at which execution of the module can be terminate	Functional Boundary	
Data A representation of facts, concept or instruction for communication, interpretation, or by humans or by automatic means	I/O data variable Write/Read	

6. A CASE STUDY

An application of the FTM framework on a Vessel Water Level Controller, which is one of the functional user requirements (FURs) of TRAC-M code is provided in this section.. To build the FTM, the functional characteristics (input/output/process)

¹ User can be either a system or a human

of the Vessel Water Level Controller are searched through the various documents produced during the software development process. The description of the Vessel Water Level Controller function is taken from the System Requirements specification, and is as follows: "The level controller provides the dynamic level position inside a given component along with the current feed water line and steam line mass flow rates. The output consists of the mass flow rate of the FILL component, which provides the inlet flow of the feed-water system."

LAYER 1: Identifying the external I/O

Control block number 202 is identified in the User Manual as the Vessel Water Level Controller function. The control block mathematical operation is described as follows:

Control Block no.: 202
 Control Block name: Vessel water level controller
 Control Block Mathematical Operation:
 $X(out) = f(X1, X2, X3, c1, c2)$
 Block Inputs: X1, X2, X3
 Block Constant: c1, c2

To achieve the Vessel Water Level Controller function, five external inputs are identified in the User Guide, as follows:

1. X1(meter) should be provided with the vessel down (comer?) water level, which could be provided by type 106 signal variable?.
2. X2 (kg/sec) should be provided with the current time step feed water line mass flow rate.
3. X3(kg/sec) should be the current time step steam line mass flow rate.
4. c1 (meter) is the user-desired vessel collapsed water level position,
5. c2 (kg/sec) is the nominal steady state feed water line mass flow rate (kg/sec)

The External I/O of the Vessel Water Level Controller is illustrated in the FTM framework with rectangles outside the functional boundary in Figure2.

LAYER 2: Identifying Sub-functions – Internal I/O

The control block function operators belonging to the Vessel Water Level Controller were identified from the Software Design Specification (SDS). These are traced forward and backward to the User Manual. To achieve the process goal of each control block function operator (sub-functions) as listed in Table 2; their internal inputs/outputs were searched using a functional decomposition technique. The mapping from sub-functions to internal I/O identified that SUBT 54 has an incorrect control block number. This has been replaced by ADD3 in Figure 2.

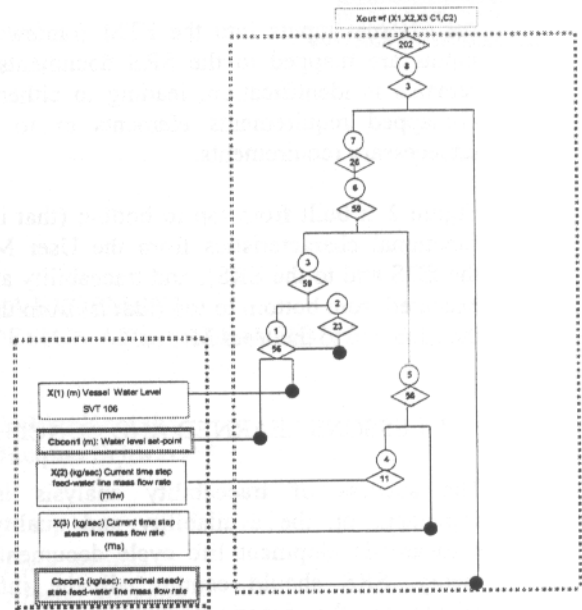


Figure 2. Analyzing the TRAC-M Vessel Water Level Control Function into the FTM Framework

Tracing the functional characteristics of each control block function helped us to identify incorrect and incomplete sub-functional processes, as well as to assess design accuracy. Table 2 summarizes the results of the FTM analysis for design accuracy.

Table 2. Functional Traceability for Design Accuracy

SRS/User Manual		FTM
Control Block number	Control Block type & name	Control Block number and type
202	Level Controller	202
23	ING (Integrate)	ING 23,
26	LAG First-order lag	LAG 26
56	SUBTC (Sum constant)	SUBTC56
59	WSUM (Weighted summer)	WSUM 59
11	DEAD (Divide)	DEAD11
54	SUBT (Subtract)	ADD3
59	WSUM (Weighted summer)	WSUM 59
56	SUBTC (Sum constant)	SUBTC56

The control block function operators are identified with the decision symbol, while the I/O values of block functions are represented with numbered circles in Figure 2.

The external inputs identified in layer 1 are mapped to the internal inputs of the lowest level of sub-process obtained using functional decomposition of the system requirement in Layer 2.

As shown in Figure 2, the inputs of the Control Block Function numbers 11, 3, 56, and 23 are mapped to the external inputs. After verifying the correctness of

the external inputs into the FTM framework, these inputs are mapped to the SRS documents; all that remains is identification, leading to either a set of unmapped requirements elements or to a set of unnecessary requirements.

Figure 2 is built from top to bottom (that is, tracing functional characteristics from the User Manual to the SDS and to the SRS), and traceability analysis is executed from bottom to top (that is, from the SRS to the SDS and to the User Manual)

7. LESSONS LEARNED AND CONCLUSION

The success of traceability analysis is highly dependent on the availability and quality of the software development life cycle documentation. In theory, SRS should capture all the information related to the functional requirements, e.g. the inputs/process/outputs paradigm. However, in practice, as happened in the project analyzed in this case study, the data and the information required had to be collected not only from the SRS, but from throughout the life cycle process documents as well. Furthermore, there was always be occasions when the information required was either: not be there; be tailored to a different audience; or not be entirely suited to the purpose at hand. However, even when suitable information is available, the ability to augment this face-to-face communication was found to be desirable, often essential, and even a fundamental working practice. To account for the context of end-use, research is needed to provide flexible functional traceability, where trace can dynamically mature to queries.

The multilayered nature of functional requirement traceability problems are illustrated in this paper. A new graphical model is proposed that captures a complete set of functional characteristics which can be located in the documentation of any phase of the software development life cycle. For FTM, automation would need to build upon research carried out to date on modelling automation.

REFERENCES

- Brown, P.G., (1991). QFD: Echoing the Voice of the Customer, AT&T Technical Journal March/April pp. 21-31.
- Davis, A.M., (1990). "Software Requirements: Analysis and Specifications." Prentice-Hall, Inc.
- DeMarco T., (1979). "Structured Analysis and System Specification". Prentice-Hall.
- DOD-STD-2167 (1998). *Defence System Software Development-DOD-STD-2167-A*. U.S.A Department of Defence Military Standard, Washington D.C., Feb 29 1998.
- Evans, M.W., 1989. "The Software Factor". John Wiley and Sons.

Hennell, M.A., 1987. "Requirement Specification & Testing." Edited by B. Littlewood. Blackwell Scientific Publication.

IEEE 830. "IEEE Guide to Software Requirements Specification," 1998.

IEEE 610.12. "Standard Glossary of Software Engineering Terminology, 1990.

Jackson, J., 1991. "A key-phrase-based Traceability Scheme, Tools and Techniques for Maintaining Traceability during Design," IEEE Colloquium, Computing and Control Division, Digest No: 1991/180.

NUREG-0800: "Guidance on the Software Review Process for Digital Instrumentation and Control Systems." U.S.A. Nuclear Regulatory Commission (NRC). June 24, 1997.

Orlena C.Z., Anthony C.W., Finkelstein A., 2002. "An Analysis of the Requirements Traceability Problem".

<http://www.cs.ucl.ac.uk/staff/A.Finkelstein/papers/rtprob.pdf>

Palmer, J.D., 1990. *Software Systems Engineering*, John Wiley and Sons, New York, N.Y., 1990