Functional Analysis for Existing Products: a Detailed Procedure

V. A. Lentz, Bruce Lerner
Otis Elevator, a UTC Company
5 Farm Springs
Farmington, Connecticut, USA
860.676.5287 / 860.676.6149
Virginia.Lentz@otis.com
Bruce.Lerner@otis.com

Abstract. Functional Analysis (FA) for regenerative systems is often executed as reverse engineering or design recovery. The initial product developer completed the Functional Analysis, and minimal updates were required or took place as the product evolved. The working definition of a function has often become ambiguous. When the goal is to reify what the product / system does, Functional Analysis is one method to be used in conjunction with the identification of the scenarios to support a system decomposition to discover underlying functions. The scenarios identify how the system / product is used by external systems (a human or other man-made system) and FA helps to expose the system behaviors that are required to support that use, and the functions that are required to support that behavior. The value of functional analysis is the yield of the main and derived functions of the system / product, which are the solution independent functions. The separation of the domain functions from the subsequently developed implementation functions (those required to provide a design dependent capability) facilitates the insertion of technology and management of change.

The transition of the modular product structure Adifon [ADI2001] from the Phase 1 team to the on-going implementation required the clarification and documentation of a procedure for Functional Analysis. The generic part of that procedure is described here with templates and detailed 'user' instructions.

THE FUNCTIONAL ANALYST

In many (new product) systems or product development teams, functional analysis is performed at all levels of the organization and by many different kinds of engineers. In the regenerative product organization, the system architect or the technologist performs functional analysis. In both cases the analyst is interested in the main, derived and implementation functions. Most of the functional analysis, for regenerative products is done in support of product line modernization and product line engineering rather than specific product engineering. The driver for the performance of functional analysis for the main and derived function is often technology innovation and the need to 'remember' the reason for the particular existing implementation functions. There is a moderate amount of functional analysis performed with each product generation in support of the articulation of new implementation functions.

THE PROCEDURE

Procedure overview. An overview of the procedure is presented in Figure 1.

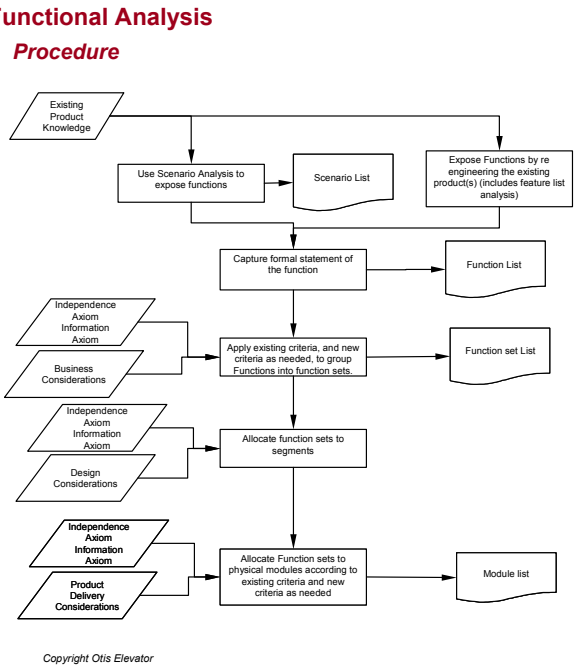

Figure 1 Functional Analysis Task Flow

The trapezoids are inputs, the rectangles are activities and the boxes represent deliverables. Decisions are not shown for diagrammatic convenience.

CAUTION: Please note that while the process is shown in a serial fashion for the sake of simplicity, it is NOT SERIAL. We will iterate, even for a single function.

An engineer's knowledge of the existing product often predisposes the use of re-engineering rather than scenarios. Both techniques should be brought into the overall process by the variety of engineers working on the product. Without scenario-based analysis, it can be difficult to separate domain functions from implementation functions.

Scenario Analysis. Use existing product knowledge to develop scenarios that document the sequence of activities to use the product / system. The user might be the 'consumer' or the maintainer of the system / product. This activity will identify the majority of the main and derived functions. The authors define *a main function as a major function that directly contributes to an Operational Scenario. These functions are unique in that they are not duplicated, shared or reused in the functional hierarchy A derived function is a function that supports the functions at the next level up in the decomposition. They are directly derived from higher level functions and do not depend upon implementation decisions.*

Some thoughts to consider for scenario analysis include:

- For the initial activity, stay with the primary scenarios. Following the path of alternative scenarios can make it difficult to develop a working picture of the system. After the paths of desired outcomes are developed, add variations to expand the description of product capabilities.
- For Scenario Analysis, the completeness of the system behavior column is less of a concern than `the completeness of the function column.
- Permit nested scenarios: In some cases the user visible behavior or the system behavior might be to 'initiate the _____ scenario'.
- A function at one level can break down into another set of scenarios → and another level of scenarios / Many times established 'functions' actually describe system capabilities, a set of functions that yield a behavior or implement a scenario.

The sequence of function execution is not represented by the function list but is partially captured in the scenario. The slavish pursuit of the sequence of function execution is not critical to the activity of exposing or grouping functions.

***Advice to the novice*:** the most difficult part is getting folks to think functions (what the system has to do to effect the desired behavior) and not think the 'current solution' (how we physically implement the function today). The word 'function' has been severely abused across the engineering community. Determining how the various participants are using the word and coming to agreement on the use of the word for the duration of the architectural activity is a difficult task. Initially the participants will not understand the need for or appreciate the value of gaining a single understanding of this pivotal word. Take the time, try to keep everybody in the room during this activity, or you will have to repeat it. For our purposes, the authors define *'function' as a characteristic task, action or activity that, when performed, contributes to achieve a defined outcome. One or more of the following may implement a function: equipment (hardware), software, firmware, personnel or procedural data.*

Reengineering. It is frequently easier for legacy engineers to work backwards from the current implementation or a series of implementations to the function(s). Let them! Exploit the knowledge of these engineers and facilitate the sessions where they use existing product knowledge to analyze the product components to identify the functions performed by the component. Reverse engineering is particularly useful to identify the implementation functions. We define implementation functions as those functions in the functional hierarchy that support Derived Functions but only exist due to implementation decisions. They are not directly derived from higher-level functions.

Some thoughts to consider for reengineering include:

- Look for different terms that have grown up over the years that really mean the same thing (the same capability, same functionality), and if necessary what are the subtle differences between the evolved use of the terms. Are these several different ways of describing the same things or do subtle differences exist? This activity is particularly important for global companies that have grown through mergers & acquisitions and are consolidating the architectural description of a variety of legacy products.
- Perhaps 60% -70% of the functions a team needs to look at will be implementation functions. Tracing the implementation functions to the product decisions that required the implementation function enables better technology insertion decisions in the future.

Capture the statement of the function. A function is a characteristic task, action or activity that, when performed, contributes to achieve a defined outcome. The statement is typically a verb then noun ….

The trick is to get the team to articulate the domain functions, the generic statement of the customer driven functions – independent of the implementation for any particular product line

Some thoughts to consider for stating the function include:

- Use a specific example then generalize to the generic function. Use another specific example from a different product to test the generic statement. By about the third specific example, the result is generic.
- Make a note of the different classes of products within a product line,- go through the initial exercise with one class in mind, - then map to the other classes of products. In general, for this level of analysis (at the main / derived functional level), the differences should disappear. The differences in requirements for the other classes might provide the rationale for the differing implementation function. They will provide the rationale for the differences in the physical instantiation of the implementation function.

How does the team know they are done exposing functions? When they have completed the architectural initiative and all the enterprise participants are using the result. However, this is an iterative process. Do enough to get started, work through the rest of the steps and then come back, as you need to. The 1st pass is complete when a rich set of functions is available for allocation. As the team works through the remaining activities of functional analysis and continues applying the rest of the systems engineering methods to support delivery of the product, additional functions may be required.

Group Functions to Function Sets. The authors define function sets as sets of functions (or function sets) grouped using defined design principles and trade-offs to support system implementation goals. Affinity Grouping based on the team's knowledge of the functions and the product was the initial step. Validation of the groupings applies criteria based on the information and independence axioms as postulated by Nam Suh [SUH1990]. We found that both grouping and separation criteria were needed to overcome the tendency to group to a monolithic set.

Function to Function Set grouping criteria:

- Independence Axiom
  - Minimize external functional dependencies
  - Maximize testing independence (greater coupling within than without)
  - Maximize opportunity to innovate/improve within the Function Set without impacting other Function Sets
- Information Axiom
  - Maximize functions with similar data needs

Function to Function Set separation criteria:

- Independence Axiom
  - Minimize differing customer sets
  - Minimize differing source of requirement
- Information Axiom
  - Minimize the information scope

Group Function sets to Segments. There is a larger grouping of the function sets that facilitates product development. For the purpose of not confusing this grouping for product development with the extant subsystem terminology for product delivery, we chose the word 'segment'. This provides for the optimization of development entities around main functions. The 7 plus or minus 2 rule applies to the results of the grouping. There might be a tendency with the first architectural initiative to group things according to the engineering disciplines of mechanical, electrical and software. RESIST. The result of applying this particular set of constraints will be sub optimization and a conflict of the axiomatic design principles. Using the information and independence axioms as postulated by Nam Suh, group the function sets into segments.

Function Set to Segment grouping criteria:

- Independence Axiom
  - Minimize external functional dependencies
  - Maximize testing independence (greater coupling within than without)
  - Maximize opportunity to innovate / improve within the Segment without impacting other Segments
- Information Axiom
  - Maximize functions with similar data needs

Function Set to Segment separation criteria:

- Independence Axiom
  - Minimize differing customer sets
  - Minimize differing source of requirement
- Information Axiom
  - Minimize the information scope

In addition to Nam Suh's axioms, additional criteria were applied to address business goals for the system. To further combine and separate the functions in a function set the authors added:

Function Set to Segment grouping criteria:

- Management to meet business goals
  - Maximize speed of development and introduction
  - Maximize control of cost and quality

Function Set to Segment separation criteria:

- Focus to meet business goals

- Maximize focus for development of a corporate capability in the functional area
- Maximize defined responsibility
- Maximize ability to manage scope of responsibility

These business criteria reflect company priorities and may need to evolve to address current technologies, external regulations etc.

Sanderson, in a 1998 Management Roundtable [SAN1998] provided guidance on mapping patterns of model evolution. The criteria included model variety and the rate of technological change. This technique enables the analysis to support the business criteria for innovation and different customer sets. The map is a simple grid as shown in Figure 2.

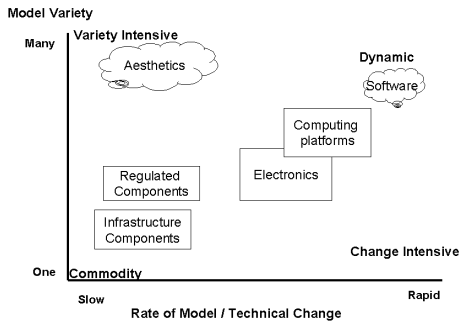

Figure 2 Mapping patterns of model evolution

For the elevator community, the infrastructure components are things the passenger doesn't see. They include rails and car frames (think heavy metal).

The regulated components tend to have more models, since the regulations have developed independently in different parts of the world. Regional variations are required. External safety driven product regulation becomes a business criteria due to the generally extensive (and expensive) testing and product qualification for these elements. The business need is to either manage the change to minimize the re-qualification required or to make major upgrades on a cyclic basis as a group of changes. Another option is to reduce the number of models supported. Understanding the function of the module / component facilitates making these decisions.

The electronics and computing platforms are changing at a rate greater than the vertical transportation industry chooses to incorporate. The design decision that moved from 'relays' to 'computer-assisted controls' identified these items as implementation functions. Software is pervasive in regenerative products and is dynamic. It is frequently still easiest to change software to fix a problem, or increase the capability of the product. The model variety may be a production factor more than an engineering factor, but software is frequently parametrically adjusted to an individual installation rather than being identical across multiple installations.

The customer visible parts of regenerative products are also very dynamic. While General Motors might reuse transmissions across the product lines, the bodylines and interiors are unique to the specific company and perhaps even within the models. For elevators, the elements having many models and finishes are those seen by the passenger; the cab interiors and fixtures. The architectural trick is to group the things together that are customer visible, and minimize the impact of changes in that area to the underlying elements. So yes we do have a couple modules that group 'things that are important to the passenger'. We call it aesthetics. The model variety in this area is extensive. The Building architects frequently want a particular look. The perception of beauty varies significantly around the globe. Global industries need an architecture that will accommodate this variety.

The other set of needs to be addressed for grouping function sets to segments are Design constraints that were represented as gray arrows on Figure 2 in Adifon [ADI2001]. The methods for shipping and installation of a product and their associated constraints (size, manufacturing techniques and locations) that are needed to ship and install a product will affect the allocation of functions to modules

Allocate function sets to physical modules. Again, applying the information and independence axioms as postulated by Nam Suh, along with a few additional guidelines, allocate the one or more function sets to top-level physical entities (modules). Module Guidelines:

- Elements in a Segment (product development) that are in different Subsystems (product delivery) must be in different Modules
- Elements in different Segments must be in different Modules
- Maximize the flexibility to 'scale' implemented Modules to cover the product range.

Apply product delivery considerations e.g. manufacturing sources, installation processes to further refine the allocation of one or more function sets to physical entities. At this level the mapping is probably going to be constrained by manufacturing processes. For elevators we have another step in that final assembly of the product is in the building elevator shaft. Thus we also need to consider the shipping packages

and installation units when mapping to the physical architecture.

The Generic statement of the function is what enables the team to get to the 'platforms', the scalable, reusable across product lines pieces of the enterprise products

Some thoughts to consider for functional allocation to the physical architecture include:

- Not all of the resulting modules need to be in the same product. Some modules will support the very low end of the product line, and other will support the very high end. Typically 80% of the modules will be in a specific item in the product line.
- The interfaces for factory decisions will vary by product and possibly by factory
- Not all 'replication of functions' based on the scenario analysis will be synthesized. There might be some repetition of monitor health, detect degradation, failure functions in multiple parts of the system. There might be some level of aggregation of functions that store, report and 'manage' the collection of health and failure data for the various segments.
- In the same manner that the function sets were grouped into segments to facilitate product development, the physical modules that are the juncture of the function sets and a physical package, will be grouped into subsystems to facilitate product delivery. This will probably require iteration of the grouping of functions to function sets.
- One rule that might drive a team back to the activity for exposing functions is that a function set shall not be shared among physical entities.
- In doing the functional synthesis, we want a stable set of modules, however, we often end up wanting to group things together based on what is in a scenario. The Caution is that Industry data tells us that solutions that slavishly follow the functional or operational architecture are terribly inefficient. Use of the independence and information axioms can alleviate this inefficiency. Another issue is the decreased ability to 'share' implementation functions.

The Result. The team now has a robust architecture. It provides an effective base for research and development that facilitates the reuse of existing supply chains. The enterprise is now in a position to perform up front planning of generation changes at the module / component level. As the technology matures we can decide to provide additional capability to the customer or returns to the enterprise. The release of a new product can then become the integration of proven components.

Don't underestimate the effort to follow through. There is resistance & roadblocks from legacy engineers, from factories, and other parts of the organization. This is CHANGE – if only evolution.

THE DELIVERABLES

The deliverables of the functional analysis and decomposition activity include a scenario list, a function list, the list of function sets and the list of modules. There are no physical widgets yet.

The Scenario List. For documenting scenarios: the user visible behavior and the system behavior will typically start out representing a solution within the product range. The functions will, at the first pass, also represent a specific implementation.

A scenario description template is provided.

P1: Scenario Title
<<Description of the scenario>>
P1a: Scenario sub-Title

Step	User Visible Behavior	System Behavior	Functions
1			
2			
3			

Figure 3 Scenario Description Table

The Function List. Information to capture about each function:

- Name
- Description
- Type: main, derived, implementation
- Source: scenario, 'parent', module/component
- Part of Function Set: (can be linked in tool)
- Inputs: needs from other functions
- Outputs: provides to other functions

Two templates are provided for documenting functions.

Name: Spin	**Type:** Implementation	**Source:** Scenario 23
Description: Do the hoochy-kootchy		
Part of Function Set: Dance		
Name:	**Input / Output**	**Description**
The Beat	I	
Motion	O	Spin yourself around

For Functions exposed by reverse engineering existing 'components' we add:

<<Existing component>>

Intended Outcome:	Unintended Impact:
Coolness	Dizziness

The second template was suggested by Ed Crawley [CRA2001].

Form	Function	Process	Other Forms
Toaster	Char Bread	Breakfast	Oven
Hose Nozzle	Water Lawn	Beautify Yard	Sprinkler

The Function Set List. Link the function set to the contributing functions. This is a simple list.

The Module List. For each module in the list provide:

- Module Name. Augment with a description of the module.
- Module functionality. This is a table of the function sets allocated to this module
- The Module Interface. This includes a description of each link end that connects this module to other modules. Standard items for a link end description includes:
  - **Headline:** << a unique designator for this link end>>
  - **Type:** Software links / Electrical links (will typically have mechanical characteristics) / Mechanical link
  - **Level:** Controlled - will be controlled by systems engineering / Identified - is only identified by systems engineering
    Connects To: Name of module where the link end ***physically*** 'connects to'.
  - **Description:** Description of link end.
  - **Item ID:** Identification of item that crosses the link end.
  - **Item Description:** Description of item.
  - **From/To Module:** Name of the module connected by the ***functional*** link carrying the item.
  - **Encodes Items:** Y: items are encoded in this link. / N: no encoded items

Type: SW	**Level:** Controlled	**Connects To:** E: Computing Platform	
Description:	Communication to other modules through the Computing Platform.		
Items That Cross the Link End			
Item ID	Item Description	From/To Module	Encodes Items
Software Bus Message	A message that is sent or received from the Computing Platform.	From/To Computing Platform	Y
Specifi-cation:	This link end does not provide anything to the link – it only uses capabilities of the Computing Platform. Therefore, there is no detailed specification here.		

Figure 4 Link End Description

The most effective presentation for the set of modules is a matrix with the modules included in the cell identifying the source segment and the associated subsystem for product delivery.

The Module Matrix has been well received and referenced.

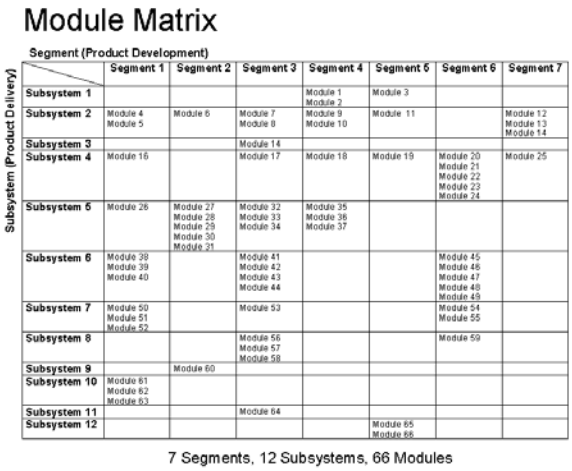

Figure 5 Module Matrix

TOOLS

Sanderson and the Software Productivity Consortium both emphasize the need for an enabling information technology infrastructure to support an architectural initiative of this kind and the subsequent product development. You can use MS Word or MS Excel, it is however cumbersome and time consuming to specialize them to Systems Engineering's needs. Using a tool such as CORE by Vitech greatly facilitates the linking and production of reports. Just having tools is not sufficient unless all of engineering can access these tools in real-time. The communications bandwidth among

engineering teams and centers is equally important.

Microsoft Office tools were initially used to capture data for this process. As the data set grew, information was captured in CORE™©® from Vitech Corporation. This class of tool allows for the graphical capture of scenarios, the text capture of element descriptions and attributes and the programmatic capture of links between elements. They are flexible enough to allow for customization to corporate terminology and schema. Once the data has been entered, general and specialized reports can be extracted from this single, central data store. Data entry errors decrease, as does the time required for report generation. Tool shortcomings fall in two areas: data exchange and visualization. Data exchange is typically a unidirectional, batch process, which requires planning to optimize the appropriate transfer level to minimize churn. The visualization issues include control of color and layout of the existing representations and the relatively limited set of views available to satisfy audiences expecting animated PowerPoint®.

FUTURE PROCESS CLARIFICATION

The authors were driven to clarify the functional analysis process for existing products to enable other engineers in the organization to apply the procedure consistently. Yes, the authors have a list of other procedures that need to be clarified. . Further work is needed in the following areas:

- How do we capture the decisions (the alternate paths) within the scenario table and the functional flow block diagrams (with 'and' and 'or' gates)
- Do we need to capture different things for the implementation functions: intended function / unintended function. This information will later be useful in analyzing emergent properties and for technology insertion

REFERENCES

[ADI2001] Adifon et al, "Validating a Commercial Product Architecture." Proceedings of the 11th Annual International Symposium of INCOSE, Melbourne, Australia, 2001.

[CRA2001] Crawley, Ed MIT 2001 "Introduction to System Architecture" Course ESD.34

[SAN1998] Sanderson, "Managing Product Families: Product Architecture and Modularity, The Management Roundtable, 1998.

[SPC1996] Software Productivity Consortium, 'Product -Line Management and Engineering Course", SPC-96006-MC version 01.00.05 June 1996

[SUH1990] Suh, Nam P., *The Principles of Design*, Oxford University Press, 1990

BIOGRAPHIES

V. A. Lentz is responsible for Program Management for Otis Elevator Company. She has been with United Technologies since 1996. Her favorite topic is the use of Systems Engineering in commercial enterprises. Prior to UTC, she spent 30 years at IBM Federal Systems, LORAL, Lockheed Martin building large, unprecedented computer-based systems such as Global Positioning System Control Segment. She was also responsible for Systems Engineering Technology, Process and Training. Ms. Lentz was President of INCOSE in 1996, a recipient of the Founders Award in 1999, and represents UTC on the INCOSE Corporate Advisory Board.

Bruce Lerner is a Principal Systems Engineer with Otis Elevator, a United Technologies Company. He has been at Otis since 1986, managing Software development for embedded communications and control, leading Software Process Improvement activities and spearheading the deployment of Systems Engineering. Mr. Lerner is a member of INCOSE (Constitution Chapter Treasurer) and the IEEE Computer Society.