Systems and Software Architecting: A Unified Approach

Howard Eisner
The George Washington University
heisner@gwu.edu

Abstract. Formulating a sound architecture has become one of the most important elements in systems engineering, software engineering and systems integration. Significant work has been done in the architecting of systems and, mostly as a separate matter, the architecting of software. This paper focuses upon the issue of a unified approach to both systems and software architecting. A specific procedure for a unified method is suggested and illustrated.

Introduction

There has been a great deal of attention paid to the architecting of systems, especially over the past 15+ years or so. Architecting has been accepted, in the main, as the first of a two-level design process for systems. The second level, following architecting, can be described as subsystem design [1]. In principle, both levels of design need to be addressed before significant parts of the system are built. Two analogies come to mind in this regard. The first has to do with A & E (Architect and Engineer) firms that have existed for years, whereby the architects do the first level of design, followed by the engineers at a more detailed level. Perhaps another analogy is the original formulation of the steps necessary to carry out software development [2, 3]. These steps include preliminary design (as per architecting), followed by detailed design (as per subsystem design as the second level).

Notwithstanding the above two-step perspective, it can also be argued that for some domains and very large systems, a recursive approach involving many levels, or steps, is more appropriate. This notion is accepted here in certain situations, but they will not be further explored in this paper.

A particularly important milestone in the matter of system architecting was the publishing of Rechtin's "Systems Architecting" book [4]. This landmark treatise established a framework for thinking about all aspects of architecting systems. Special attention was paid to the systems architect, boundaries and interfaces, multi-dimensional challenges, and a list of extremely useful heuristics. A few years later (1997), Rechtin teamed with Maier to look more deeply into the complex issues surrounding the architecting of systems [5]. These two books did a lot to encourage researchers, as well as practitioners, to try to figure out how to construct system architectures that were sound and practical. A third activity, one sponsored by the Department of Defense (DoD),

attempted to define the way in which architecting should be accomplished. This was called an "Architectural Framework", and has now been designated as the DoDAF [6]. With this brief background, we now provide a short overview of system architecting approaches.

System Architecting Approaches

The DoDAF approach to architecting, cited above [6], is based upon the notion that three "views" are critical, namely, the:

- *operational view* – a description of the tasks and activities, operational elements, and information flows required to accomplish or support a military operation
- *systems view* – a description, including graphics, of systems and interconnections providing for, or supporting, warfighting functions
- *technical view* – the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements

These basic views were expanded into very specific sets of Essential and Supporting Views in these three categories, while adding a category called "All Views".

The original framework showed a six-step process for constructing an architecture. These steps are reiterated here in Table 1:

1 – Articulate the intended use of the architecture
2 – Establish the scope, context, environment (and any other assumptions) of the architecture
3 – Determine which characteristics the architecture needs to capture
4 – Establish which architecture views and supporting products should be built
5 – Build the needed products
6 – Use the architecture for its intended purpose

Table 1 – Steps Suggested by DoDAF for Building an Architecture

The three basic view approach articulated in the DODAF has been carried forth since approximately 1996, with enhancements, explanations and improvements forthcoming with successive versions. Staying within the view context, the Executive Summary in the 2007 documentation of the DoDAF approach [6] defined an **architecture** as:

- the structure of components, their relationships, and the principles and guidelines governing their design and evolution over time

This approach has been extremely important as a point of departure for the DoD as well as other executive agencies of the government and selected parts of industry. To facilitate a comparison, the following has been documented by this author [1]:

- An **architecture** is an “organized top-down selection and description of design choices for all the important system functions and sub-functions, placed in a context to assure interoperability and the satisfaction of system requirements”

In addition to the well-known DoDAF approach, there are other notions that have been set forth, all of which pertain directly to constructing a system architecture. These include:

- MoDAF (Ministry of Defence architecting framework)
- SOA – the service-oriented architecture
- EA – the enterprise architecture

The MoDAF comes from the UK and expands the DoDAF approach by, for example, adding a strategic view, an acquisition view and a connection to enterprise architecting. The SOA uses internet available services to carry out tasking from remote locations. The Enterprise Architecture provides “a strategic planning framework that relates and aligns information technology (IT) with the business functions that it supports”. [1] Yet another approach, the Zachman method as applied to enterprise architecting, maps the classical six questions (who, what, when, where, why, how) against ways to transform “an abstract idea into an instantiation” [7].

It is clear that there is considerable interest in systems architecting and many approaches that have merit and need to be further pursued.

Software Architecting Concepts

Examinations of software architecting are incomplete without at least some reference to Grady Booch, who has made many significant contributions to this field, and also writes the “on architecture” column for the IEEE Computer Society. One of his articles is called “the Irrelevance of Architecture” [8]. Despite this provocative title, Booch concludes with “In retrospect, I think I’ve titled this column incorrectly: architecture is quite relevant”.

The software engineering field has been explored by a large number of researchers and practitioners. A smaller group has focused upon the matter of software architectures. A quite recent book [9] has much to say about the design of architectures along with such topics as connectors, modeling, analysis, visualization, and many others. Relevant definitions in this treatise include:

- “A software system’s architecture is the set of principal design decisions made about the system”, and
- “a Reference Architecture is the set of principal design decisions that are simultaneously applicable to multiple related systems, typically within an application domain, with explicitly defined points of variation” [9]

Other significant investigators into the field of software architecting include:D. Garlan, M. Shaw, L. Bass, P. Clements, R. Kazman, D. Perry, A. Wolf, B. Hayes-Roth and P. Kruchten. Key notions regarding software architectures that appear in the works of these and other researchers (e.g., the Software Engineering Institute at Carnegie-Mellon University) include:

- processing elements, data elements, connecting elements, functional components, abstract system specifications, issues beyond the algorithm and data structures, overall structure, composition and selection of design alternatives, functional partitioning, allocation of domain function, properties of components, relationships between and among components

In addition, we have the following important observations:

(1) "the most difficult design task…is the decomposition of the whole into a module hierarchy" [Wirth, 10]
(2) "from this process, one identifies modules of solutions or of data whose further refinement can proceed independently of other work [Brooks,11]
(3) "the effectiveness of a 'modularization' is dependent upon the criteria used in dividing the system into modules" [Parnas, 12]

We see patterns in these descriptions suggesting that software needs to be decomposed into a module hierarchy, that these modules need to be as independent as possible from one another, that criteria for developing these modules need to be addressed and defined, and that these modules may be interpreted as software components. We also see the notions of functional components, functional partitioning and the allocation of domain function.

Finally, with respect to software architecting, we look very briefly here at a suggestion for how to reconcile system and software architectures, as set forth by M. Maier [13]. The author points out that standard systems engineering approaches are "often not well suited to support complex software developments". The systems side favors functional definition and decomposition whereas the software side tends to favor a data-based approach. Moreover, the author says that "the most successful very large system architectures typically take a strongly-layered approach". A definitive solution to this apparent divergence between systems and software architecting is not completely clear, but directions are suggested.

A Unified Approach

The unified approach to architecting, as described in this paper, is the same whether dealing with hardware, software, or some combination thereof. Indeed, that is the strength of the approach, as it follows the seven essential steps cited in Table 2.

Step 1 – Functional Decomposition
Step 2 - Requirements Definition and Allocation

Step 3 – Synthesis of Alternatives
Step 4 – Analysis of Alternatives
Step 5 – Cost Effectiveness View of Alternatives
Step 6 - Creation of Additional Views
Step 7 – Selection of Preferred Architecture

Table 2 – Steps for Implementing the Unified Approach to Architecting

Each of these steps is discussed in some detail below.

Step 1 – Functional Decomposition

Whatever the system, and whatever its size, the first essential step is to decompose it into its major functions and sub-functions. This decomposition is essentially hierarchical, and there should be no question as to whether or not a function is within or outside the scope of the system. The decomposition is carried out so as to minimize the interfaces between parallel decomposed elements. The decomposition is not based upon the physical system; only on the functions that must be performed. The architecting team must be careful to not decompose to a level of detail where one is trying to "look at every tree in the forest". To try to operate at the level of the "tree" is to jeopardize the entire architecting process.

An example of the step of functional decomposition is shown in Table 3, taken from a real-world system developed by NASA [14].

FUNCTION 1 – Flight Operations
Sub-Function 1.1 – Mission Control
Sub-Function 1.2 – Mission Planning and Scheduling
Sub-Function 1,3 – Instrument Command Support
Sub-Function 1.4 – Mission Operations
FUNCTION 2 – Science Data Processing
Sub-Function 2.1 – Data Processing
Sub-Function 2.2 – Data Archiving
Sub-Function 2.3 – Data Distribution
Sub-function 2.4 – Data Information Management
Sub-Function 2.5 – User Support for Data Information
Sub-Function 2.6 – User Support for Data Requests
Sub-Function 2.7 – User Support for Data Acquisition and Processing Requests
FUNCTION 3 – Communications and System Management
Sub-Function 3.1 – Distribution of EOS Data and Information to EOSDIS Nodes
Sub-Function 3.2 – Distribution of Data Among Active Archives
Sub-Function 3.3 – Interface with External Networks
Sub-Function 3.4 – Network/Communications Management and Services
Sub-Function 3.5 – System Configuration Management
Sub-Function 3.6 – System/Site/Elements Processing Assignment and Scheduling
Sub-Function 3.7 – System Performance, Fault, and Security Management

Sub-Function 3.8 – Accounting and Billing

Table 3 – Illustrative Decomposition of a NASA System

Approximately the same approach is followed for the functional decomposition of a software system. Although there is considerable commentary on the matter of software decomposition, software modules, and related notions, we offer here in Table 4 an example of the functional decomposition of project management software, drawn from the menu of familiar software (Microsoft Project) of this type.

- File (New, Open, Close, Save, Find File…)
- Edit (Cut, Copy, Paste, Clear, Delete Task…)
- View (Gantt, PERT, Resource Graph, Zoom…)
- Insert (Insert Task, Insert Column, Notes…)
- Format (Font, Timescale, Gantt Wizard…)
- Tools (Spelling, Change Time, Leveling…)
- Window (New, Hide, Split, Arrange All…)
- Help (Contents, Search, Preview, Cues…)

Table 4 – Illustrative Decomposition of a Software System

This table illustrates eight functions with the sub-functions shown parenthetically. The functional decomposition does not include a definition of the way in which these functions are to be instantiated. These functions and sub-functions are not the same as the needed software modules, although at a later time they may (or may not) become the "names" of the modules.

It should be noted here that functional decomposition for software does not keep the software engineer from instantiating the software with a data-based or even object-oriented approach. Yet another way to think about this process is to recognize that every software module, however constructed, has a purpose, and that purpose is its "function". The bottom line is that this unified approach, constructed in the manner described, does not overly constrain the software architect to take an approach that he or she considers less desirable. Separating the function from its method of instantiation is designed to provide sufficient freedom in terms of software design.

Step 2 – Requirements Definition and Allocation

Given the decomposition, we now wish to define and allocate requirements to all the decomposed functions and sub-functions. We typically refer to the "requirements" document that has been provided by the system sponsor in order to carry out this step. It is clearly desirable to have at least one key requirement for each and every one of the decomposed functions and sub-functions. Where this is not the case, it becomes a task of the architecting team to "derive" a set of requirements from the information at hand. Where this is not possible, a "TBD" status can be assigned, and updated at a later time. The architecting team uses the best information it has at the time it is necessary to carry

out the architecting process. As usual, an incomplete set of requirements increases the risk attendant to designing and building the system at hand.

Step 3 – Synthesis of Alternatives

This "synthesis" step is critical under this unified architecting approach. One can visualize this step as a table of rows and columns. The rows are the decomposed functions and sub-functions. The columns are three alternative architectures, defined by:

a. a low cost approach to architecting the system
b. a knee-of-the-curve approach to architecting the system, and
c. a high-effectiveness approach to architecting the system

These 3 approaches establish what might be called the "low cost domain", the "best value domain" and the "high effectiveness" domain.

Each cell of the table is filled out by the architecting team by selecting the way(s) in which each sub-function is to be instantiated. Stated another way, each cell represents the design approach selected by the team so as to satisfy the requirements for each and every sub-function. It is at this point that the architecting team can select a hardware solution, a software solution, or a combination solution, for each and every sub-function. The precise nature of that solution is defined as completely as possible, given the aggregate state of knowledge of the architecting team.

We note again that in this "synthesis" step we have to come to terms with how to approach the design of each function and sub-function. If part of the design is pure software, the architect may select a data-driven, object-oriented, or any other approach representing his or her best solution. It is believed that, as constructed here, the "functional" approach and the "data-driven" approach are not ultimately in conflict.

Step 4 – Analysis of Alternatives

The result of step 3 is to provide (at least) three alternative architectures, corresponding to the low cost, best value, and high effectiveness domains. Unless otherwise constrained, the team now analyzes and evaluates the effectiveness of each of the three alternatives against a set of criteria. This analysis represents an effectiveness evaluation for the overall system. Evaluation criteria are selected by the architecting team based upon the best information provided by the system's sponsor as well as the domain knowledge of the architecting team. A set of twelve evaluation criteria is illustrated in Table 5 below.

Criterion 1 – Reliability
Criterion 2 – Maintainability
Criterion 3 – Producability
Criterion 4 – Electronic Security
Criterion 5 – Physical Security
Criterion 6 – Grade of Service

Criterion 7 – Safety
Criterion 8 – Capacity
Criterion 9 – Sustainability
Criterion 10 – Risk
Criterion 11 – Human Factors
Criterion 12 – Graceful Degradation

Table 5 – Illustrative Criteria for Evaluating the Effectiveness of Alternative Architectures

The results of the evaluation constitute a set of effectiveness measures. They are combined by defining and using weighting factors, if the criteria are considered to have different levels of importance. Standard "weighting and rating" schemes can be used to calculate effectiveness levels [1]. The overall results can, at this point, be called a complete effectiveness evaluation of the alternative architectures, using the best knowledge available to the architecting team. In general, the architectures contain both hardware and software, as well as positions and roles for the human element. This "effectiveness" assessment leads immediately to the next step, and also represents the "analysis" part of the "synthesis/analysis" duality.

Step 5 – Cost-Effectiveness View of Alternatives

The results of step 4 provide measures of the effectiveness of the alternative architectures. It is now necessary for the architecting team to calculate the overall life cycle costs for these alternatives. Putting the effectiveness measures and costs in a graphical display yields the cost-effectiveness view we are seeking. We are now able to "see" our results in a form more conducive to decision-making. Our overall concept is clearly one that can be characterized by a cost-effectiveness analysis of alternatives.

Although it is easy to talk about calculating costs, the realities make it a difficult and often arduous process. Bottoms-up approaches generally require lengthy data collections. Top-down approaches, like parametric cost estimating, tend to have gaps and questions as to the applicability of empirical data. Fortunately, we've been at cost estimating for a long time and basically understand how to do it.

Step 6 - Creation of Additional Views

Unlike some of the other approaches to architecting, we attempt now to create additional views that provide as much quantitative insight as possible as to the features (both positive and negative) of the alternatives that have been set forth. A candidate set of such views is provided below in Table 6.

1. Requirements Satisfaction
2. Risk and Requirements
3. Interoperability
4. Cost by Function

5. Cost vs. Requirements
6. Sensitivity to Changes in Criteria Weights
7. Effectiveness vs, Risk
8. Effectiveness vs. Human Factors
9. Effectiveness vs. RMA
10. Effectiveness vs. Residual Performance Factors

Table 6 – Other Possible Views [15] That Support the Architecture Selection

The above list, however, does not at all preclude the use of more qualitative views that help the team with respect to their evaluation of alternatives. This includes any and all of the views suggested by the DoDAF (e.g., operational view, etc.) and other approaches cited earlier.

Step 7 – Selection of Preferred Architecture

An integral part of this unified approach to architecting is that the team creates several (at least three) alternative architectures from which, ultimately, a preferred architecture will be selected. Each of the alternatives is synthesized (designed) by formally creating design alternatives for each and every sub-function of the system. These are then used to create the low cost, best value, and high performance alternatives. Each alternative is then characterized by measures of effectiveness and cost, as well as additional views that create further qualitative and quantitative information. The architecting team, at that point, has sufficient information to select a preferred architecture from among the alternatives. This selection is presented to the customer as the preferred solution. If the customer concurs, then the team enters the next phase, namely, subsystem design. If the customer does not concur, specific non-concurrence issues are defined and considered in detail.

Further Commentary on Views

The notion of “views” appears to be straightforward but actually contains several subtleties as well as unanswered questions. There is great value in addressing and constructing views. But perhaps some additional factors need to be explored.

Reasonable re-statements of the views of DoDAF are:

- an operational view is a view of the system architecture from an operational perspective
- a system view is a view of the system architecture from a system perspective
- a technical view is a view of the system architecture from a technical perspective

These re-statements can be seen to imply that, first, a system architecture is constructed and then three views are developed from that construction. This more-or-less separates the notion of an architecture from the views of that architecture. The views provide interesting and useful information about the architecture, but are distinctly not the

architecture itself. Further, if that is true, then what exactly is the architecture, and how is it to be constructed?

A contrary perspective is simply to accept that the three architectural views are, in the aggregate, the architecture itself. This has some attraction, but also some serious problems. Two brief examples follow that might bring out aspects of these problems:

(1) Suppose we are trying to define the architecture for an advanced radar system. Try constructing an operational view, a systems view and a technical view. Do these views capture the essence of the architecture of the radar?
(2) Suppose we are trying to define the architecture for the human body. Try constructing an operational view, a systems view and a technical view. Do these views capture the essence of the architecture of the human body?

Of course, embedded in the above questions is the possible answer – it depends upon what we mean by an architecture. Go back to the two definitions presented in this paper and see if that helps. In the light of the above, one might say that it is likely that certain views of an architecture do not actually reveal the underlying architecture. Other views might come closer, but still fall short. A snapshot of a house might tell us something about its architecture, but not enough to reveal the essence of its full architecture. Multiple snapshots add information, but might still fall short in important ways with respect to revealing what it is that constitutes the architecture.

Continuing on, yet another conclusion is as follows:

- A view, or set of views, might fall significantly short of revealing the method of architecting

Stated another way, in this as well as many other real-world cases, the products (the views) do not imply the process (the method of architecting). In various ways, perhaps one should not have the expectation that the products of architecting should reveal the process of architecting. But one might also believe that such a feature is highly desirable. This feature is approached if each step in the process clearly produces products that are the *essential* set of “views”.

Finally, we complete this commentary regarding views by briefly examining the notion of Architectural Descriptions (ADs). For this purpose, we note the existence of a standard dealing with ADs of software-intensive systems [16]. This standard specifically distinguishes between an architecture and an AD, as follows:

“An Architecture: The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution”

“An Architectural Description (AD): A collection of products to document an architecture”

One of our leading researchers into matters of architecting reiterates and emphasizes this difference [13]. If this notion is applied to the DODAF construct, a conclusion might be that although we have three "preferred" views of systems, we also have areas that require further elucidation, such as:

- a substantive, repeatable and agreed-upon method for system architecting
- a more complete and agreed-upon definition of a system architecture
- inferring the precise nature of an architecture from the views of that architecture, and
- a unified method for system and software architecting

It is believed that this paper sets forth a clear and definitive method of architecting both systems and software. A companion suggestion involves a recommended definition of a systems architecture and a set of relevant views.

Summary

This paper presents a unified approach to systems and software architecting. Several of the key points in the paper are cited below:

1. the architecting approach is based upon a cost-effectiveness assessment of specific and well-defined alternatives
2. the approach leads to the selection of a preferred architecture, from among the alternatives
3. broad domains that help in constructing architectural alternatives include low cost, best value or knee-of-the-curve, and high effectiveness approaches
4. the overall approach leans heavily upon the need to functionally decompose the system, whether the system is largely hardware, software, the human element, or combinations thereof
5. the functional decomposition for software systems is a major part of the process, in that it defines these key functions and sub-functions, but not how to instantiate them. The functional decomposition leaves room for developing a module hierarchy as well as "modules of solutions or of data whose further refinement can proceed independently of other work" [11]
6. in general, there is not a forced one-to-one correspondence between the functional decomposition hierarchy and the ultimately constructed sets of modules that constitute the designed software system
7. this unified approach, whose first step is functional decomposition, does not generally prevent the software architect from selecting data-based, object-oriented, or other preferred methods as ways to instantiate the specified functions
8. the "synthesis" part of the architecting process specifically develops design alternatives (including software modules where necessary) that become the basis for the set of architectural alternatives

9. each step in the architecting process leads to a product, but the dominant products, or views, deal with (a) functional decomposition, (b) synthesis, (c) analysis (of alternatives), and (d) the cost-effectiveness "view"
10. study of the output "views" of the recommended process tends to define the process itself, a quite desirable feature of an architecting approach
11. current approaches to architecting do not appear to embody the above feature
12. an architecture, hardware or software, is typically not fully defined by the views of that architecture. For the recommended architecting process herein defined, it is suggested that the key views (see '9' above) actually define the critical (but not all) aspects of the architecture.

References

1. H. Eisner, "Essentials of Project and Systems Engineering Management", 3rd Edition, 2008, John Wiley, Hoboken, NJ
2. A. Sage and J. Palmer, "Software Systems Engineering", 1990, John Wiley, Hoboken, NJ
3. J. Marciniak and D. Reifer, "Software Acquisition Management", 1990, John Wiley, Hoboken, NJ
4. E. Rechtin, "Systems Architecting", 1991, Prentice-Hall, Englewood Cliffs, NJ
5. E. Rechtin and M. Maier, "The Art of Systems Architecting", 1997, CRC Press, Boca Raton, FL
6. Department of Defense Architectural Framework (DoDAF), version 1.5, Volumes I, II and III, 23 April 2007, U. S. Department of Defense
7. see **www.zachmaninternational.com**
8. G. Booch, "The Irrelevance of Architecture", IEEE Software, IEEE Computer Society, May/June 2007
9. R. Taylor, N. Medvidovic and E. Dashofy, "Software Architecture – Foundations, Theory and Practice", 2010, John Wiley, Hoboken, NJ
10. N. Wirth, (1995), "A Plea for Lean Software", IEEE Computer Magazine (February): 64-68
11. F. Brooks, (1995), "The Mythical Man-Month", Reading, MA: Addison Wesley Longman
12. D. Parnas, "On the Criteria to Be Used in Decomposing Systems Into Modules", Communications of the ACM, 15, No. 12, December 1972, pp. 1053-1058
13. M. Maier, "System and Software Reconciliation", Systems Engineering, Vol. 9, Number 2, Summer 2006
14. Phase C/D Requirements Specification for the Earth Observing System Data and Information System (EOSDIS) (1990) Greenbelt, MD: Goddard Space Flight Center
15. H. Eisner, "New Systems Architecture Views", American Society for Engineering Management (ASEM), 25th Annual Conference, October 20-23, 2004, Alexandria, VA
16. IEEE Standard 1471-2000, "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems", 2000