Systems Architecting and Software Architecting - On Separate or Convergent Paths?

Dr. Howard Eisner, George Washington University

Dr. Eisner, since 1989, has served as Distinguished Research Professor at The George Washington University. Prior to that time, he served in industry as a research engineer, manager and executive. He has written 5 books on systems engineering and various aspects of management. He is a Life Fellow of IEEE and a Fellow of INCOSE and the New York Academy of Sciences. He is also a member of Tau Beta Pi, Eta Kappa Nu, Sigma Xi and Omega Rho.

SYSTEMS AND SOFTWARE ARCHITECTING – ON SEPARATE OR CONVERGENT PATHS?

Howard Eisner

The George Washington University

Abstract

Systems Engineering is an overall method and discipline for building large-scale systems. It has been described as consisting of some twenty-five subordinate processes or an equivalent number of elements. Whatever the overall description, it is agreed that software engineering is a critical activity within the overall field of systems engineering. The success of our efforts at building large systems often depends upon how well we execute the software tasks within a systems engineering framework. The software development issue is high on our list of areas that require continuous improvement if we are to be successful at systems engineering. Both systems and software engineering have methods for architecting. In the former case, there are several approaches, all of which are rather definitive, and have well-known proponents and supporters. These include DoDAF, MoDAF, Enterprise Architecting, and others. In the case of software engineering, the approaches seem to be deep, varied and interesting, and not necessarily agreed upon. In this paper, the author provides an overview of both the fields of systems and software architecting, looking especially for areas of possible commonality. Conclusions are set forth including the possibility of a unified approach. Recommendations follow, suggesting future actions as well as research areas. These results have direct relevance to the structure and content of curricula, current and future, for systems and software engineering and architecting.

Introduction

This paper examines both systems and software architecting methods in order to determine if they are on separate or convergent paths. There are several approaches to both systems and software architecting. After exploring these approaches, and looking for areas of possible commonality, the author sets forth a potentially unified approach. This is followed by a set of recommendations regarding future actions and research areas. These results can be applied directly to current and future curricula in the fields of systems and software engineering and architecting.

Systems Engineering Architecting

An extraordinary technical leader who set the stage for architecting large-scale systems was E. Rechtin. His book on architecting established the fundamentals, which included his articulation of the four basic approaches, namely[1]:

1. The normative (pronouncement) methodology
2. The rational (procedural) method
3. The argumentative approach, and
4. The heuristic approach

A few years after Dr. Rechtin's book appeared, the Department of Defense came out with its C4ISR Architectural Framework, later re-named as DoDAF [2]. That important activity focused upon the notion of *architecture views*, establishing the following views as definitive:

- Operational view
- Systems view
- Technical view

The precise way in which these views "become" the architecture was not directly addressed, except that a six-step process for building an architecture was suggested as follows:

1. Articulate the intended use of the architecture
2. Establish the scope, context, environment 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

Whatever the shortcomings of the above "method", the above three views notion has persisted, and even expanded to include an "all view" concept. Perhaps a way to look at this set of activities is to say - if one has constructed an architecture, then the DoD strongly supports the notion that the operational, systems and technical views are critically important views of that architecture. This somewhat leaves open the question – what, exactly, is an architecture? Is it, for example, the "sum" of the views of that architecture? If we return to the DoDAF[2] documentation, an answer to that question appears to be at least connected to a well defined set of products that are strongly related to the "view" concept.

Moving beyond the DoDAF we see the Ministry of Defence Architectural Framework (MODAF), from the U.K. This framework[3] is basically an enterprise architecture where the enterprise is the Ministry of Defence (MOD). It emphasizes information that is or should be present in the enterprise and is represented largely in *seven views*, as:

1. Strategic Views
2. Operational Views
3. Service Oriented Views
4. Systems Views
5. Acquisition Views
6. Technical Views
7. All Views

The orientation toward information is recognized through a Meta Model (also called the M3) that shows how all the information elements are related to one another. On the matter of a recommended architecting procedure within the Framework, none is suggested.

We note that the MODAF is directly related to the notion of an Enterprise Architecture. That is significant in that in this discussion, we are more interested in how to architect a general system in distinction to architecting an enterprise. Indeed, it would appear to be necessary to maintain that distinction rather than blur the lines by trying to think about these two notions as being the same. The perspective is that we are not trying to architect an enterprise; we are trying to architect a system.

Another Enterprise Architecture approach is that of TOGAF, The Open Group Architecture Forum[4]. This is a rather elaborate construction, with many artifacts and apparently based originally upon the Technical Architecture Framework for Information Management (TAFIM) from the Department of Defense.

John Zachman did the entire field a service through his formulation of the Zachman Framework[TM,5]. Much of this approach can be understood by examining his matrix of columns and rows whereby:

- The columns are the “standard” interrogatories of what, how, when, who, where and why
- The rows are identification, definition, representation, specification, configuration and instantiation

As Zachman suggests, “this matrix would necessarily constitute the total set of descriptive representations relevant for describing something…anything: in particular an enterprise”[5]. So Zachman takes a rather generalized approach that applies to any and all enterprises. And Zachman insists that his approach is an ontology for describing an enterprise, and not a specific methodology to describe a particular process. Thus, it does not suggest how to develop a particular system or software architecture.

The author of this paper has spent a considerable amount of time formulating and testing a method for architecting a system[6]. Early concepts were brought by the author from the industrial world to academia, starting in 1989. These concepts were articulated and presented in a first course in systems engineering. As the concepts were refined, they were incorporated into a second course in systems engineering. For approximately 16 years, students used this method in order to fulfill mid-semester examination requirements. Thus, the method was “tested” by years of graduate student applications and class discussions.

The following four steps summarize the overall method, called **CE-AM (**cost-effectiveness architecting method), at a top level:

1. Functional decomposition
2. Synthesis
3. Analysis
4. Cost-effectiveness evaluation

After the first step, it is critical to synthesize a set of alternatives by constructing a matrix where:

- The rows are the system functions and sub-functions, and
- The columns are (at least) three system alternatives representing (a) a low cost approach, (b) a high effectiveness approach, and (c) an attempt to find the knee-of-the-curve solution, sometimes known also as the "best value" solution

This synthesis step is the centerpiece of the architecting process, and puts the architect in a position to move on to a cost-effectiveness assessment of a set of specific alternative architectures. This architecting procedure has been used in some detail in an academic environment for 23 years and applied to the architecting of both systems and software. Outgrowths of the procedure are reported to have been applied in both government and industry. Within the context of this method are the following definitions[6]:

"**Architecture.** 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;

"Architecting. A process with the following simplified steps: (1) functional decomposition of the system, (2) construction of design choices for all important functions and sub-functions (synthesis), (3) evaluation of the resultant interoperable system alternatives (analysis), and (4) display of the results so as to facilitate the selection of a preferred, cost-effective architecture from among the constructed alternatives"

Software Engineering Architecting

A straightforward definition of a software system architecture can be found in the interesting treatise on agility and discipline[7] as:

" a software system architecture defines a collection of software and system components, connectors and constraints; a collection of system stakeholders' need statements; and a rationale which demonstrates that the components, connectors and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements"

This definition clearly emphasizes components, connectors, constraints and stakeholder needs.

Garlan and Shaw set the stage for a better understanding of software architecting as early as 1994[8] with their overview of this important topic. At that time, it was an "emerging" field, and they articulated a number of common architectural "styles". They asserted that a number of heterogeneous styles could be combined into a single design. Examples of styles included:

- The pipe and filter style
- The data abstraction and object-oriented style
- The event-based, implicit invocation style
- The layered system style

The software system architect has these styles at his or her disposal, and the results of the style selections, in effect, constitute an architecture.

A more recent and detailed presentation regarding software architectures[9] substantially agrees with these Garlan and Shaw notions. Emphasis, however, is placed upon two design techniques available to the architect, namely, object-oriented design (OOD) and domain-specific software architecting (DSSA). The following top-level definition is offered:

“A software system’s architecture is the set of principal design decisions made about the system”

The basic building blocks to characterize a software architecture are:

- Processing elements
- Data elements, and
- Connecting elements

Sets of and decisions regarding these elements, therefore, can constitute an architecture.

If we move to yet another source[10], we see a pointer toward software system *structure,* which “includes the organization of a system as a composition of components; global control structures, the protocols for communication, synchronization and data access; the assignment of functionality to design elements; the composition of design elements; physical distribution; scaling and performance; and dimensions of evolution. This is the software architecture level of design”

Perhaps the most relevant paper in terms of the overall topic of this exploration is that produced by Mark Maier[11]. First, he gives us a definition of an architecture as “the fundamental organization of a system, embodied in its components, their relationship to each other and the environment, and the principles governing its design and evolution”. He also cites that an architecture is “the embodiment of the set of design decisions that define essential characteristics of the system”. Moving explicitly to software, he suggests that such an architecture is “the embodiment of the earliest set of design decisions about a (software) system, and these early bindings carry weight far out of proportion to their individual gravity with respect to the system’s remaining development, its service in deployment, and its maintenance life”.

In terms of the importance of a coherent architecture, Maier properly asserts that “if a system has not achieved a system architecture, including its rationale, the project should not proceed to full-scale development”. Further, Maier basically does not support the notion that systems and software architecting should be based upon the same or similar methods. His reasons are traceable to his observations as to the differences between system and software developments and especially their structures.

As part of his case, Maier also points to a comment by one of our most capable software engineers, Frederick Brooks. In Brooks’ notable treatise on software engineering[12] he points out the need for conceptual integrity in our software systems. This is typically embodied in the software architecture. A good software architect will thereby assure this integrity, i.e., it is the *job* of the software engineer to make sure the system has the required integrity.

Finally, with respect to the matter of software architectures, we take a brief look at the IEEE recommended practice for architectural descriptions[13]. In this “standard” a formal architectural description (AD) is introduced. Further, at that time, it was acknowledged that there was not “any reliable consensus on a precise definition of a system’s ‘architecture’”. However, it is still possible, and desirable, to record an architecture by its description (AD). A system’s AD can also be directly related to a set of “views” of the architecture of that system. In effect, what they were saying is: we don’t have consensus on what an architecture is, but we can still provide descriptions (and views) that are useful for practitioners of software engineering.

A Unified Approach

The title of this paper is now addressed by the following:

- The current evidence is that systems architecting and software architecting appear to be on divergent paths, unless an approach is suggested that demonstrates the possibility of bringing these two notions together

This part of this paper suggests that systems and software architecting can be “unified”, at the appropriate level of what is meant by an architecture.

The overall procedure is the cost-effectiveness architecting method (referred to here as **CE-AM**), as described earlier. A basic notion is to accept the idea that systems (hardware, software, both, etc.) can and should be broken down into functions and sub-functions. Thus, functional decomposition becomes a critical aspect of the unified architecting procedure. The next step, of primary importance in this approach, is to construct the “synthesis matrix” which defines alternative design approaches for each and every sub-function, for (at least) three system architectures. The design approaches explicitly cover the systems domain and *also* the software domain. The overall notion is depicted in Figure 1 below.

Functions	**Sub-functions**	**Architecture 1**	**Architecture 2**	**Architecture 3**
1	**1.1**	**System DA1.1-1** **Software DA1.1-1**	**System DA1.1-2** **Software DA1.1-2**	**System DA1.1-3** **Software DA1.1-3**
1	**1.2**	**System DA1.2-1** **Software DA1.2-1**	**System DA1.2-2** **Software DA1.2-2**	**System DA1.2-3** **Software DA1.2-3**
1	**1.3**	**System DA1.3-1** **Software DA1.3-1**	**System DA1.3-2** **Software DA1.3-2**	**System DA1.3-3** **Software DA1.3-3**
2	**2.1**			
2	**2.2**			

. **.** **.**	**.** **.** **.**			
N	**N.1**			
N	**N.2**			
N	**N.3**			
N	**N.4**	**System DAN.4-1** **Software DAN.4-1**	**System DAN.4-2** **Software DAN.4-2**	**System DAN.4-3** **Software DAN.4-3**

Figure 1 – The Synthesis Step for Systems and Software Architecting: DA = Design Approach; DAx.y-z = Design Approach for sub-function x.y for architecture z

After the synthesis matrix has been developed, the (three) alternative architectures are evaluated using a ”standard” weighting and rating scheme[6]. This is the “analysis” step which produces measures of the cost and the effectiveness of each of the alternatives.

We note that in the **CE-AM** approach we are explicitly defining and evaluating alternative architectures, with the ultimate goal of finding a cost-effective architecture (solution) for the customer(s) (stakeholders). The steps of the procedure are the same as the critical “views” in that each step is defined by an unambiguous view. Some might call this a “one-to-one” relationship such that when the step is taken, the view is automatically generated.

Above all, the functional decomposition of the system becomes the unifying element of this architecting approach. As such, it brings the system design and the software design together under a common “umbrella”, the sub-function. That also assures that both system and software considerations will be brought to bear for each and every sub-function, and with an understanding of the relationship between the system and the software, at that level of design.

Future Actions

Suggested future actions depend upon understanding the possible implications of a unified architectural approach that works, from a practical point of view. The top-level features of that approach may be summarized as:

- A common framework that facilitates and incorporates both systems and software architecting
- A key element of that framework is functional decomposition of the system, which typically is instantiated by both hardware and software

- An overarching method that seeks to formulate a provably cost-effective system for the customer and stakeholders
- Synthesis and analysis of alternative architectures, leading to the selection of a preferred architecture
- Tested through the development of hundreds of architectures, containing both systems and software elements
- The steps of the architecting process provide outputs that are themselves the critical views, i.e., a one-to-one correspondence between the steps and the most important views
- The critical views are: (1) functional decomposition, (2) synthesis of alternative architectures, (3) analysis of alternative architectures, (4) graphical representation of the cost and effectiveness of each alternative

In terms of actions, this author suggests:

1. Widespread testing of the **CE-AM** across the board within industry, academia and government
2. Further research with the main focus to verify and validate the method
3. Use of the method for real systems of both hardware and software, that fill a specific need
4. Systematic construction of new views, above and beyond those already considered
5. Formalizing the method with respect to moving from analyses of alternative systems to the selection of a preferred alternative system

Basically, new methods are accepted as more and more people experiment with them, and find successful outcomes.

Research Areas

Back in 2006, Barry Boehm highlighted a trend that he called "the increasing integration of software and systems engineering"[14]. Given the key issue of this paper, we would see the matter of unifying systems architecting and software architecting as a major challenge under this overall trend. More specific areas of research for bringing systems and software architecting together include the following:

1. Constructing test cases that use the unifying approach suggested here
2. Looking at interoperability issues
3. Exploring inter-relationships between the system and software design approaches at the sub-function level
4. Exploring and suggesting the next level of "views" and architectural descriptions (ADs)

5. Looking at cases for which there is significant interaction between sub-functions that are not part of the same functions
6. Defining and formalizing the processes that lead to a preferred architecture from a series of alternative architectures

Education for Systems and Software Architecting

We have a quite interesting summary of the background of systems engineering in the academic community[15]. In particular, Elliott Axelband examines both systems engineering as well as what he calls "the professionalization of systems architecting". In other words, he set the stage for a most serious exploration of systems architecting, starting with curricula in academia.

As noted earlier in this paper, a first course in systems engineering was introduced into the author's graduate education program more than twenty years ago. System architecting and software engineering (not software architecting) were important parts of that curriculum. As the architecting of systems received more attention (e.g., DoDAF[2]), it was brought into that course. Meanwhile, as the specific **CE-AM** procedure matured, it too became part of the curriculum. This procedure was discussed in the first course, and eventually tested in the second course through student projects. Many of these projects dealt with software architecting, a challenging application that suggested a unifying approach might be feasible and practical.

It is fair to say that current curricula exist in various engineering schools, dealing with both systems and software architecting and architectures. They may not be in the same department, but they are present. If we are able to unify systems and software architecting, the natural next step would be to construct one or more courses that define the resultant theory and practice. Bringing these notions into the academic world could be an important step in terms of (a) helping with real-world problems and issues for large-scale systems, (b) extending the theory and practice of architecting, and (c) recognizing the critical importance of architecting as a touchstone for systems and software engineering.

Bibliography

1. Rechtin, E., "Systems Architecting", Prentice-Hall, 1991
2. C4ISR Architecture Framework, version 2.0, December 18, 1997, Department of Defense
3. See www.modaf.com
4. See www.togaf.org
5. See test.zachmaninternational.com
6. Eisner, H., "Essentials of Project and Systems Engineering Management", John Wiley, 3rd Edition, chapter nine
7. Boehm, B. and R. Turner, "Balancing Agility and Discipline", Addison-Wesley, 2004

8. Garlan, D. and M. Shaw, "An Introduction to Software Architecting", CMU Software Engineering Institute Technical Report CMU/SEI-94-TR-21, ESC-TR-94-21, (1994), SEI, Pittsburgh, PA

9. Taylor, R., N. Medvidovic and E. Dashofy, "Software Architecture – Foundations, Theory and Practice", John Wiley, 2010

10. Garlan, David, "Software Architecture", *Encyclopedia of Software Engineering, Second Edition,* John Marciniak (E-I-C), John Wiley, 2002, p. 1318

11. Maier, M., "System and Software Architecture Reconciliation", *Systems Engineering,* Volume 9, Issue 2, Summer 2006

12. Brooks Jr., Frederick, "The Mythical Mon-Month", Addison-Wesley, 1995

13. "Draft Recommended Practice for Architectural Description", Software Engineering Standards Committee for Architecture Working group, IEEE P1471/D5.2, December 1999

14. Boehm, B., "Some Future Trends and Implications for Systems and Software Engineering Processes", *Systems Engineering,* Volume 9, Number 1, Spring 2006

15. Axelband, E., "The Professionalization of Systems Architecting", chapter 11 in *The Art of Systems Architecting,* by E. Rechtin and M. Maier, CRC Press, 1997