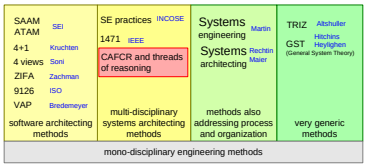
Positioning the CAFCR Method in the World

-



Gerrit Muller
University of Southeast Norway-NISE
Hasbergsvei 36 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

This chapter positions the CAFCR architecting methods realtive to other methods. The other methods originate in software architecting, system architecting and system engineering, and more general systems science. Some background is given of the IEEE 1471 standard that has proven to be to be a useful fundament for the CAFCR method.

version: 1.5 status: finished June 5, 2018

1 Introduction

This chapter positions the "architectural reasoning" *architecting method* relative to other engineering and architecting methods.

Section 2 describes work that is related to the research of architecting methods. Section 3 articulates explicitly the specific contribution of this thesis. The IEEE 1471 is explained further in section 4, because its contents is highly relevant in this context.

2 Related Work

Conventional disciplines, such as mechanical engineering, electronic engineering, et cetera have a clear set of methods and tools. Students can learn the discipline by attending universities and following their curriculums.

This is not the case for systems architecting. Only a few universities teach systems architecting. There are multiple reasons for the fact that teaching systems architecting methods at universities is difficult. First of all, sufficient depth of engineering know-how is needed to be able to work in the architecting area. In other words, a conventional discipline is a prerequisite to become an architect.

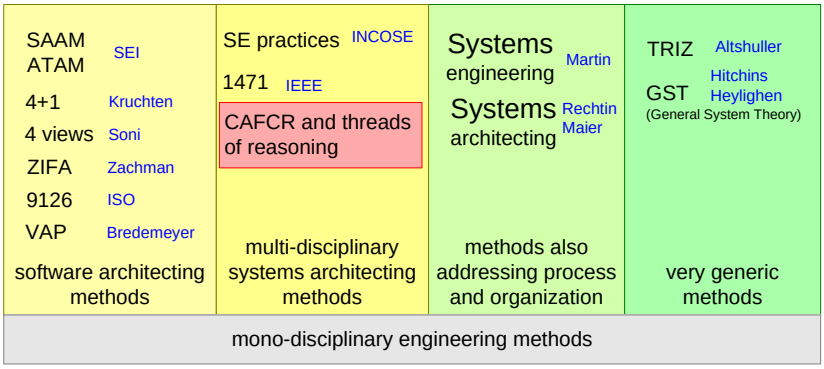


Figure 1: Classification of architecting methods

Secondly, architecting is done for problems with a wider scope than conventional engineering problems. The larger the scope, the more ill-defined a problem becomes. The methods range from *flexible* for ill-defined problems to *rigid* for well-defined problems[1].

[1] Of course this is an oversimplification. Sometimes agile methods are highly effective in well-defined problems. Sometimes rigid methods can perform wonders in an ill-defined problem. In general, mature methods are available for well-defined problems, while the uncertainty in ill-defined methods requires more flexibility.

Figure 1 shows a classification of architecting methods, with the scope of the method as differentiating factor. The software architecting methods have the smallest scope. System architecting methods widen the scope to system level. This thesis addresses the multi-disciplinary systems architecting methods. The scope can be further increased to include processes and organizational issues. The widest scope pertains to very generic methods, which claim to be domain agnostic and to create value by cross-fertilization across domains. At the bottom of the classification we find the mono-disciplinary methods, which are the fundamentals on which all methods build.

2.1 Software Architecting Methods

A whole class of methods originate in the Information Technology (IT) world and address software architecting. The software architecting methods do not address the system level problems, such as hardware/software trade-offs.

The Software Engineering Institute at Carnegie Mellon University, [18] and [19], increases the problem scope and puts a lot of emphasis on processes, and restricts itself to software architecture. Examples of methods developed here are Software Architecture Analysis Method (SAAM) [13] and Architecture Trade Off Analysis Method (ATAM) [12].

Zachman provides a framework for enterprise architectures, see [20]. This framework defines two dimensions with six aspects each, creating a space with 36 different views. Bredemeyer describes a nice visual method "The Visual Architecting Process" [4]. The Bredemeyer method provides context views and a path from context views to design views. Both Zachman and Bredemeyer are software oriented.

Well known multi-view software architecting methods are Soni [8], and the 4+1 method from Kruchten [14]. These two methods use multiple views. The scope of Soni methods, however, is completely limited to the technical solution domain. Kruchten is also focused on the technical solution domain, but he makes a small step into the problem domain by use cases in the fifth view.

ISO 9126 [11] is a standard that consolidates a quality framework. The framework addresses the same type of qualities that are discussed in chapter **??**. Unfortunately ISO 9126 limits itself to software only.

2.2 Multi-disciplinary System Architecting Methods

A further increase in scope can be found in the *Systems Engineering Community*, with INCOSE[9] (International Council on Systems Engineering) as representative organization. All stakeholders are taken into account and the full life-cycle is emphasized. Examples of this approach can be found at the INCOSE web site [5].

Some standardization work has been done in the scope of systems, stakeholders

and the full life cycle. An example is IEEE 1471, which is a framework that fits into this scope, see section 4.

This thesis about architectural reasoning, based on the "CAFCR" method, also addresses the scope of systems, their stakeholders, and the full life-cycle. Boundary conditions to the methods in this thesis are structure and characteristics of the business, the organizations, and the processes.

2.3 Methods also Addressing Process and Organization

The architect is often confronted with many more needs, worries, and complications, originating from human and business aspects. This broad working environment is full of uncertainties. Rechtin and Maier [17] address this wider scope from the architecting point of view. Martin [15] comes from the systems engineering community. He provides a method that deals with all the complexity, but that has less emphasis on the human aspects.

2.4 Very Generic Methods

Many system architecting and design methods are universally applicable. General Systems Theory (GST), for example, addresses any kind of system, ranging from economical, or ecological, to social, see for instance [6] and [7]. GST suffers from being extremely abstract and difficult to apply, due to a broad scope and the generic nature of the theory.

TRIZ [1] is a methodology for innovation that originates in Russia. A set of innovation patterns is derived from studying large collections of inventions. These patterns are transformed into innovation methods that can be applied to a very broad range of applications. One of the starting points of TRIZ is that the way of innovating in one domain provides inspiration for innovation in other domains. TRIZ provides a number of useful insights.

The subtitle of this thesis, *balancing genericity and specificity*, indicates one of the continuous struggles of the architect: the power and the beauty of generic solutions versus the uniqueness of effective, individual solutions. Or in other words, do we get carried away in generic thinking, or do we drown in the details? In this thesis the scope will be limited to systems with embedded processors and software. This still pertains to a very broad range of products: from wafersteppers to televisions, and to systems on a chip).

3 What is the Unique Contribution of this Work?

This section discusses the unique contributions of the CAFCR method. Although every single element mentioned here is present in one of the discussed methods,

the uniqueness of CAFCR is the combined application of all these elements simultaneously.

Integral and Multi-disciplinary This work focuses on architecting methods on the *system* level for embedded systems. As described in Subsection 2.1, many methods focus only on a part of the multi-disciplinary system problem, for instance only on the software architecture. A lot of architecting methods provide more or less closed and complete solutions. The available methods are partial methods from a systems viewpoint. The method described in this thesis addresses the integration of results obtained with these more partial methods. Also a number of multi-disciplinary system design submethods are described in this thesis. The basis for this integration is the combined use of CAFCR views, qualities, and threads of reasoning.

Goal-Oriented This method stresses the importance of being externally oriented. Architecting must be goal-oriented or objective-driven. Many existing methods do not take the goals and objectives into account.

Practical, based on Industrial Experience The method, which is based on a broad industrial experience, addresses the real problems[2] in system design. The usability aspect can be seen in the light-weight use of formulas, and in the association of many statements with common sense. Some of the published methods are more academic, well thought through, but not really addressing the problems in system design, and difficult to implement in the industrial practice.

Flexible The wide application range of the creation of *software and technology intensive products*, requires a flexible and adaptive method. The method must provide guidance, and should not constrain the architect by forcing a rigid harness on him. In principal the *architecting method* must be able to integrate the results of *any* partial method.

Builds on standards The method builds on top of standards, such as ISO 9126 for qualities and IEEE 1471. In fact the method can be viewed as an instantiation of an IEEE 1471 method, see Section 4.

Support for short innovation cycles System engineering methods originate from the aerospace domain, with very different reliability and safety requirements. Such methods tend to be more rigid, resulting in very long development cycles. This distinction of "slow but safe" domains versus "fast but less reliable" domains disappears quickly. Cross-fertilization of these domains can be very useful. In contrast to the aerospace domain the *CAFCR* method is intended for domains with short innovation cycles.

[2]Many problems in system design are caused by unforeseen interactions between independent designed functions or qualities. See for instance Chapter **??** for examples of system design problems in the Medical Imaging case.

4 IEEE 1471

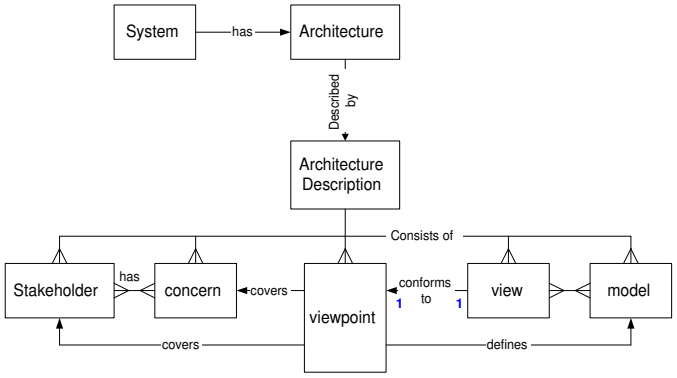


Figure 2: The IEEE 1471 model for stakeholders, viewpoints and architecture descriptions

Figure 2 shows a somewhat simplified IEEE 1471 model. IEEE 1471 [2] is a standard that describes a framework for architecting. The framework introduces a number of important concepts:

Stakeholders People or organizations that have an interest in the system under consideration.

Concerns The articulation of the needs and worries of the stakeholders.

Viewpoints The points of view used to describe part of the problem or solution. IEEE 1471 makes a subtle difference between *view* and *viewpoint*. We ignore this difference here.

Models Frequently used method to make problem and solution descriptions.

Architecture description The combination of stakeholders, concerns, viewpoints and models to describe the architecture of a system.

The main contribution of IEEE 1471 is to provide a framework that covers all of these aspects. The individual concepts have been in use by many architects for a long time.

On top of providing the framework, IEEE 1471 also recognizes the fact that complete consistency in the entire architectural description is an illusion. The real world of designing complex systems is full of stakeholders with fuzzy needs, often contradictory in itself and conflicting with needs of other stakeholders. The insights of individual designers are also full of different and changing insights. This notion of incomplete consistency is not an excuse for sloppy design; quite

the opposite: recognizing the existence of inconsistencies is a much better starting point for dealing with them. In the end, no important inconsistencies may be left in the architecture description.

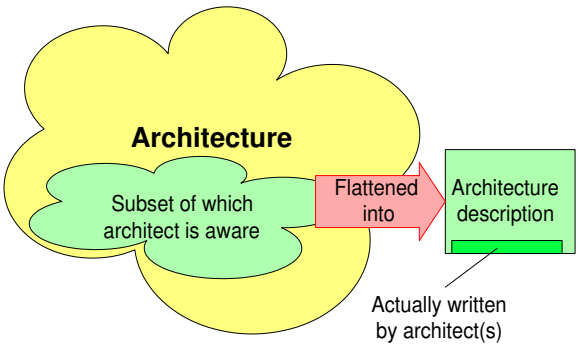


Figure 3: The architecture description is by definition a flattened and poor representation of an actual architecture.

IEEE 1471 makes another interesting step: it discusses the architecture *description* not the *architecture* itself. The *architecture* is used here for the way the system is experienced and perceived by the stakeholders[3].

This separation of *architecture* and *architecture description* provides an interesting insight. The *architecture* is infinite, rich and intangible, denoted by a cloud in figure 3. The *architecture description*, on the other hand, is the projection, and the extraction of this rich *architecture* into a flattened, poor, but tangible description. Such a description is highly useful to communicate, discuss, decide, verify, et cetera. We should, however, always keep in mind that the description is only a poor approximation of the *architecture* itself.

5 Acknowledgements Positioning Architectural reasoning in the world

Eugene Ivanov introduced TRIZ to me. He translated some of the articles and summarized the essentials. He also showed the similarity between TRIZ ideas and the Gaudí articles.

[3]Long philosophical discussions can be held about the definition of **the** architecture. These discussions tend to be more entertaining than effective. Many definitions and discussions about the definition can be found, for instance in [7], [3], or [10]

References

[1] Genrich Altshuller. *The Innovation Algorithm; TRIZ, systematic innovation and technical creativity*. Technical Innovation Center, Worcester, MA, 2000. Translated, edited and annotated by Lev Shulyak and Steven Rodman.

[2] Architecture Working Group (AWG). *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. The Institute of Electrical and Electronics Engineers, Inc., 2000.

[3] Dana Bredemeyer. Definitions of software architecture. `http://www.bredemeyer.com/definiti.htm`, 2002. large collection of definitions of software architecture.

[4] Dana Bredemeyer and Ruth Malan. The visual architecting process. `http://www.bredemeyer.com/pdf_files/WhitePapers/VisualArchitectingProcess.PDF`, 2003.

[5] J. C. DeFoe (Editor). An identification of pragmatic principles. `http://www.incose.org/workgrps/practice/pragprin.html`, 1999.

[6] F. Heylighen and C. Joslyn. What is systems theory? `http://pespmc1.vub.ac.be/SYSTHEOR.html`, 1992. *Principia Cybernetica Web* (Principia Cybernetica, Brussels).

[7] Derek K. Hitchins. Putting systems to work. `http://www.hitchins.co.uk/`, 1992. Originally published by John Wiley and Sons, Chichester, UK, in 1992.

[8] Christine Hofmeinster, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Addison-Wesley, 2000.

[9] INCOSE. International council on systems engineering. `http://www.incose.org/toc.html`, 1999. INCOSE publishes many interesting articles about systems engineering.

[10] Carnegie Mellon Software Engineering Institute. How do you define software architecture? `http://www.sei.cmu.edu/architecture/definitions.html`, 2002. large collection of definitions of software architecture.

[11] ISO/IEC. ISO 9126: The standard of reference. `http://www.cse.dcu.ie/essiscope/sm2/9126ref.html`, 1991.

[12] R. Kazman, M. Klein, and P. Clements. ATAM: Method for architecture evaluation. `citeseer.nj.nec.com/kazman00atam.html`, 2000.

[13] Rick Kazman, Leonard J. Bass, Mike Webb, and Gregory D. Abowd. SAAM: A method for analyzing the properties of software architectures. In *International Conference on Software Engineering*, pages 81–90. ICSE, 1994.

[14] Philippe B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, pages 42–50, November 1995.

[15] James N. Martin. *Systems Engineering Guidebook*. CRC Press, Boca Raton, Florida, 1996.

[16] Gerrit Muller. The system architecture homepage. `http://www.gaudisite.nl/index.html`, 1999.

[17] Eberhardt Rechtin and Mark W. Maier. *The Art of Systems Architecting*. CRC Press, Boca Raton, Florida, 1997.

[18] Carnegie Mellon Software Engineering Institute SEI. Software engineering management practices. `http://www.sei.cmu.edu/managing/managing.html`, 2000.

[19] Carnegie Mellon Software Engineering Institute SEI. Engineering practices. `http://www.sei.cmu.edu/engineering/engineering.html`, 2002.

[20] John Zachman. The zachman framework for enterprise architecture. `http://www.zifa.com/`, 1987.

History

Version: 1.5, date: May 24, 2004 changed by: Gerrit Muller

- added abstract
- updated the accompanying presentation

Version: 1.4, date: April 19, 2004 changed by: Gerrit Muller

- added little bit more explanation to other architecting methods
- replaced book by thesis
- added footnote to “real problems”

Version: 1.3, date: April 5, 2004 changed by: Gerrit Muller

- significant update to the Section Unique Contribution
- small text improvements
- changed status into finished

Version: 1.2, date: February 27, 2004 changed by: Gerrit Muller

- changed title to fit with the book title
- small text improvements
- changed status into concept

Version: 1.1, date: January 13, 2004 changed by: Gerrit Muller

- added the contribution of a number of multi-disciplinary system design submethods
- changed status into draft

Version: 1.0, date: November 27, 2003 changed by: Gerrit Muller

- changed ”system level methods” in ”systems architecting methods”
- many small textual updates
- changed status into draft

Version: 0.5, date: September 26, 2003 changed by: Gerrit Muller
- changed the graph in a classification
- structured the text according to the classification

Version: 0.4, date: July 29, 2003 changed by: Gerrit Muller
- the entities in the graph have all been changed to method and source
- added external references, removed references to Gaudí
- added (still empty) section "What is the unique contribution of this work?"

Version: 0.3, date: March 21, 2003 changed by: Gerrit Muller
- added TRIZ to comparison
- changed status to preliminary draft
- added acknowledgements

Version: 0.2, date: November 11, 2002 changed by: Gerrit Muller
- defined logo

Version: 0.1, date: September 2, 2002 changed by: Gerrit Muller
- Created, no changelog yet