White Paper

Application of Patterns to Systems Architecting

Robert J. Cloutier

Lockheed Martin Corporation, Stevens Institute of Technology

Version 1

07 December 2005

World Headquarters: Telelogic AB (publ.), PO Box 4128, SE-203 12 Malmö, Sweden, Phone: +46 40 650 00 00, Fax: +46 40 650 65 55
American Headquarters: Telelogic North America Inc., 9401 Jeronimo Road, Irvine, CA 92618, USA, Phone: +1 949 830 8022, Fax: +1 949 830 8023
Web: www.telelogic.com, E-mail: info@telelogic.com

Table of Contents

Abstract

A pattern is a model or facsimile of an actual thing or action, which provides a degree of representation (an abstraction) enabling the repeated recreation of that entity. The existence of patterns is almost universal. The human mind seems to perceive patterns without conscious thought - we notice an individual's personal habits because they form patterns. Patterns are also used in a number of engineering disciplines – software engineering, requirements engineering, and mechanical engineering to name a few. This paper discusses the motivations for using patterns in architecting complex systems.

Introduction

Music incorporates repeating patterns to make it easier to learn the tune. Three childhood songs have the same tune – Twinkle, Twinkle Little Star; Baa, Baa Black Sheep and the Alphabet Song. All are from the same Mozart tune. Mathematics is full of patterns – methods to solve similar problems: sum of squares, quadratic equations, and standard integration of common functions, etc. [Solingaros 1999]. The use of cul-de-sacs by civil engineers/architects in a housing development is another example of a commonly found pattern. Other examples of patterns are geometric forms – a large circle and a small circle differ only by the radius.

The architect Christopher Alexander is widely attributed as being the first to understand the value of applying patterns to systems. In Alexander's case, patterns were applied to the construction of homes, buildings, and communities. Throughout the 1960's and 1970's, Alexander strived to improve on the art of urban design by creating patterns that other architects could use. Alexander stated that, "Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution…" [Alexander 1979]. He believed patterns could always be improved. "We may then gradually improve these patterns which we share, by testing them against experience…" [Alexander 1979].

Alexander created what has become known as the Alexandrian form of documenting patterns. It is straightforward in its contents – 1) Pattern Name, 2) Context 3) Problem, and 4) Solution. [Alexander 1977]. The name of the pattern must be descriptive and represent the solution the pattern is addressing. This is critical for pattern reuse; if the pattern name is cryptic or has meaning only to the pattern author, there is no value in documenting the pattern. To demonstrate this concept, Alexander outlines the pattern language for a farmhouse in the Bernese Oberland [Alexander 1977].

- North South Axis
- West Facing Entrance Down the Slope
- Bedrooms in Front
- Garden to the South
- Balcony Toward the Garden
- Two Floors
- Hay Loft at the Back
- Pitched Roof
- Half-Hipped End
- Carved Ornaments

Alexander was attempting to lower the cognitive load of design by exploring large design spaces on behalf of the architect [Coplien 1997] [Alexander 1964]. Alexander found that patterns helped him to express design in terms of the relationships between the parts of a house and the rules to transform those relationships [Coplien 1997]. If we take the previous sentence and replace the word "house" with "system," it easily applies to the notion of systems architecture patterns.

Patterns as a way to capture implicit knowledge

Though one can attain advanced degrees in Systems Engineering, there are many systems engineers that do not have a systems engineering degree. Instead, they are recognized as systems engineers through their experience of working on some number of systems, and by being able to think about how parts of the system interact, both positively and negatively, with one another.

The engineer captures some of his experience in project engineering notebooks or in lessons learned data. Regardless of the information that is overtly captured regarding specific design solutions, the systems engineer carries that knowledge forward to the next project. This undocumented knowledge is sometimes referred to as corporate knowledge or history, implicit knowledge, project history, etc. This implicit knowledge is useful only to those with whom it is shared. According to Hole (2005), the holder of that implicit knowledge may become a bottleneck in applying that systems experience on current or future projects.

Patterns are documented to capture implicit knowledge that has been implemented in other projects to successfully solve similar problems.

If a pattern exists only as implicit knowledge, then others can only understand and use it through a form of repeated storytelling.

Though a pattern may be reusable by the person who recognizes the pattern first, it is of more value if the pattern is transferable to others so they can apply the pattern also. The value of patterns in engineering disciplines today is to facilitate sharing of implicit knowledge. Though a pattern can be transferred through verbal communications, such as storytelling, it is more accurately and reliably transferred through the more formal approach of documentation. If engineers do not fully document patterns, then it is less likely that they will implement the complete pattern the next time they use it, as they may forget or inadvertently leave something out. Documenting the pattern greatly increases the possibility of a correct implementation. To this end, Alexander created what has become known as the Alexandrian form, which is straightforward in its contents:

1. Pattern Name
2. Context
3. Problem
4. Solution

Pattern Hierarchies

There are patterns that are applied at different levels of a system based on the appropriateness of the pattern and the detail of the system at the point in which the pattern is being applied. For instance, within the software community, there are system patterns (sometimes referred to as architecture patterns), design patterns and idioms. Figure 1 represents a pattern hierarchy proposed by van Zyl and Walker [2001] for software systems.

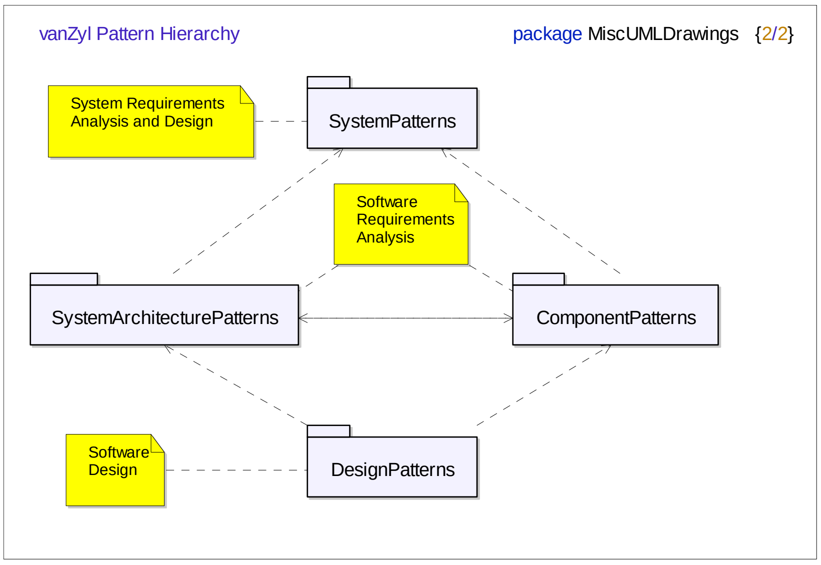
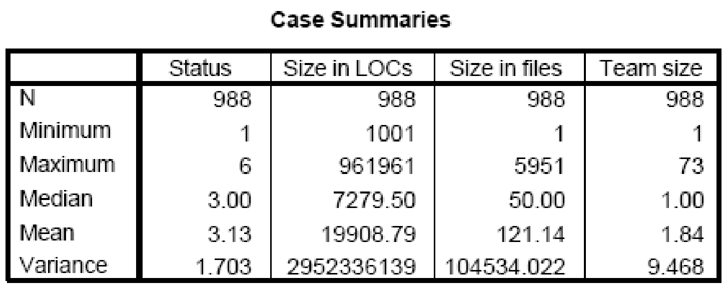

vanZyl Pattern Hierarchy
package MiscUMLDrawings {2/2}
System Requirements Analysis and Design
SystemPatterns
Software Requirements Analysis
SystemArchitecturePatterns
ComponentPatterns
Software Design
DesignPatterns

Figure 1 – van Zyl Pattern Hierarchy

It is important to recognize that in their work, when they refer to the system, they are really referring to the software system. Their work may be extendable to the broader system, or system of systems architecture with further research. Systems level patterns may be applicable when representing the highest levels of a system to represent an entire system or a part of a system. They may also include structure and system boundaries. Based on knowledge gained by the use of patterns in other disciplines, use of patterns in Systems Engineering may also provide the foundation for common design and development as well as a common means of communicating about the system.

Value of Patterns

Hahsler [2004] has produced one of the very few sources of quantitative work done on the value of patterns. He studied the use of design patterns in open-source software written in Java™. By studying the log files of configuration-controlled, open-source software, Hahsler was able to isolate software that utilized design patterns if the patterns were documented in the code comments. The team only used the 23 patterns introduced by Gamma, et al [1995]. The dataset included 988 projects representing over 19.5 million lines of code, with 1,487 different software developers working on that code. The code represented the entire lifecycle of software development, from planning to mature software. Descriptive statistics for the sample are shown in Table 1.

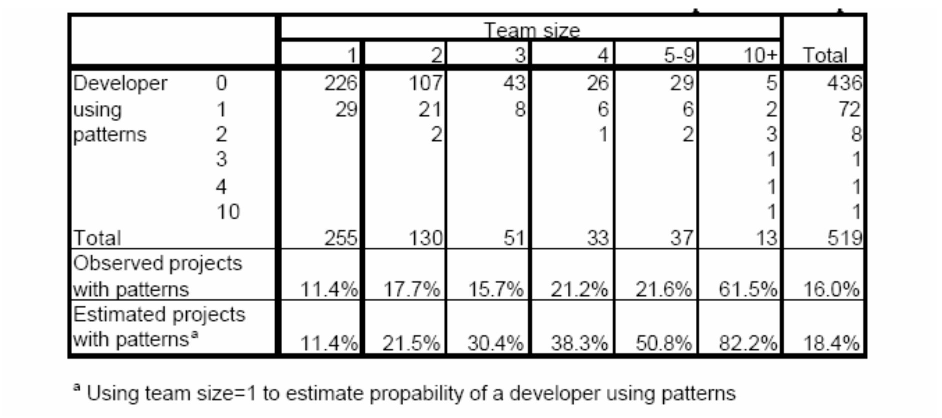


Case Summaries

	Status	Size in LOCs	Size in files	Team size
N	988	988	988	988
Minimum	1	1001	1	1
Maximum	6	961961	5951	73
Median	3.00	7279.50	50.00	1.00
Mean	3.13	19908.79	121.14	1.84
Variance	1.703	2952336139	104534.022	9.468

Table 1 – Pattern Usage in Open Source Software Projects Summary Statistics

By removing the projects that were in the planning phase, and removing projects that had less than 1,000 lines of code, 519 projects remained for analysis. Several key facts were noted in the data. Looking at team size, the larger the team size the higher the estimated projects with patterns as seen in Table 2. In Hahsler's conclusions, he states that the results show that "design patterns are adopted for documenting changes and thus for communication in practice by many of the most active open source developers."



		Team size						
		1	2	3	4	5-9	10+	Total
Developer using patterns	0	226	107	43	26	29	5	436
	1	29	21	8	6	6	2	72
	2		2		1	2	3	8
	3						1	1
	4						1	1
	10						1	1
Total		255	130	51	33	37	13	519
Observed projects with patterns		11.4%	17.7%	15.7%	21.2%	21.6%	61.5%	16.0%
Estimated projects with patterns[a]		11.4%	21.5%	30.4%	38.3%	50.8%	82.2%	18.4%

[a] Using team size=1 to estimate propability of a developer using patterns

Table 2 – Estimated Pattern Usage in Open Source Software Projects

Patterns are models or abstractions of reality and not silver bullets. However, in other engineering domains, they help solve difficult problems by leveraging a predecessor's knowledge.

Today's systems have become extremely complex. It is difficult, if not impossible, for a systems engineer to mentally juggle all of the smallest details of a system. As already demonstrated, patterns can exist at multiple levels. During the course of system architecture, design, and implementation, a project team may use architecture patterns, design patterns, process patterns, implementation patterns (for software code), machine patterns (to cut metal for cabinets), test patterns, and validation patterns.   At each level of the architecture, the pattern will contain the correct level of detail for the stage in which it is applied.

Applications of Patterns to Systems Engineering

Patterns are in use in other technical disciplines, including object-oriented software engineering, which embraced patterns in the mid 1990's. Requirements engineers have been applying patterns to their field for a number of years [Gross 2000]. They are exploring the use of documentation patterns to facilitate common functionality and specifications across systems.

In 1998, IEEE conducted a half-day colloquium on "Understanding Patterns and Their Application to Systems Engineering." The systems engineering community now appears to be more interested in learning about patterns and their application. At the annual symposium for INCOSE in 2004 and 2005, Haskins [2004, 2005a, 2005b] and Harrison conducted a tutorial entitled *Introduction to Patterns through Writing Systems Engineering Patterns*. Haskins has also done work on systems engineering process patterns.

One of the strongest arguments in favor of using patterns in Systems Engineering is improved communications. I believe that improved communications of the architecture and design teams, resulting from the use of patterns, may facilitate the capture of good architectural concepts and implementations and preserve them for future projects. Patterns are needed at the systems architecture level to create a common lexicon between systems architects. The ability to describe parts of a design and implementation in the context of known and understood patterns fosters a common understanding of the architecture as it does in other engineering disciplines. This has been facilitated with the use of architecture frameworks such as DoDAF (Department of Defense Architecture Framework) and implemented as a Telelogic TAU® G2 plug-in. However, systems architecture patterns may enable the implementation of common design features across systems (reuse) to reduce overall systems TOCs (Total Ownership Costs) by reducing the cost to design and produce a new system, as well as reducing the long term maintenance costs due to commonality.

In the communities that have adopted the use of patterns, the patterns tend to become standardized as they are implemented in other programs and as they are presented at professional conferences and in professional journals. When this standardization occurs within a company, it fosters reuse of designs and even code that is generated from the architecture patterns. This reuse has improved efficiency and productivity [Coplien 1997]. Based on Coplien's experience, one could argue that documenting current patterns may reduce the documentation costs and complexity for any company that elects to pursue systems engineering patterns. Finally, as was found at Bell Laboratories, architecture patterns provide expert advice to novice architects. Going hand-in-hand with that is the fact that architectural patterns can help control the complexity of an architecture by standardizing it on a well-known and practiced pattern.

The work on the SysML standard may be a very useful adjunct to the use of patterns in systems engineering. With the integration of SysML and UML, there may be an evolving syntax that will be used by system architects, systems engineers and software engineers to define the problem and describe the solution using the same toolset.

Earlier in this paper, van Zyl's pattern hierarchy was shown. Extending that to a broader application, the following systems pattern hierarchy is proposed as shown in Figure 2.

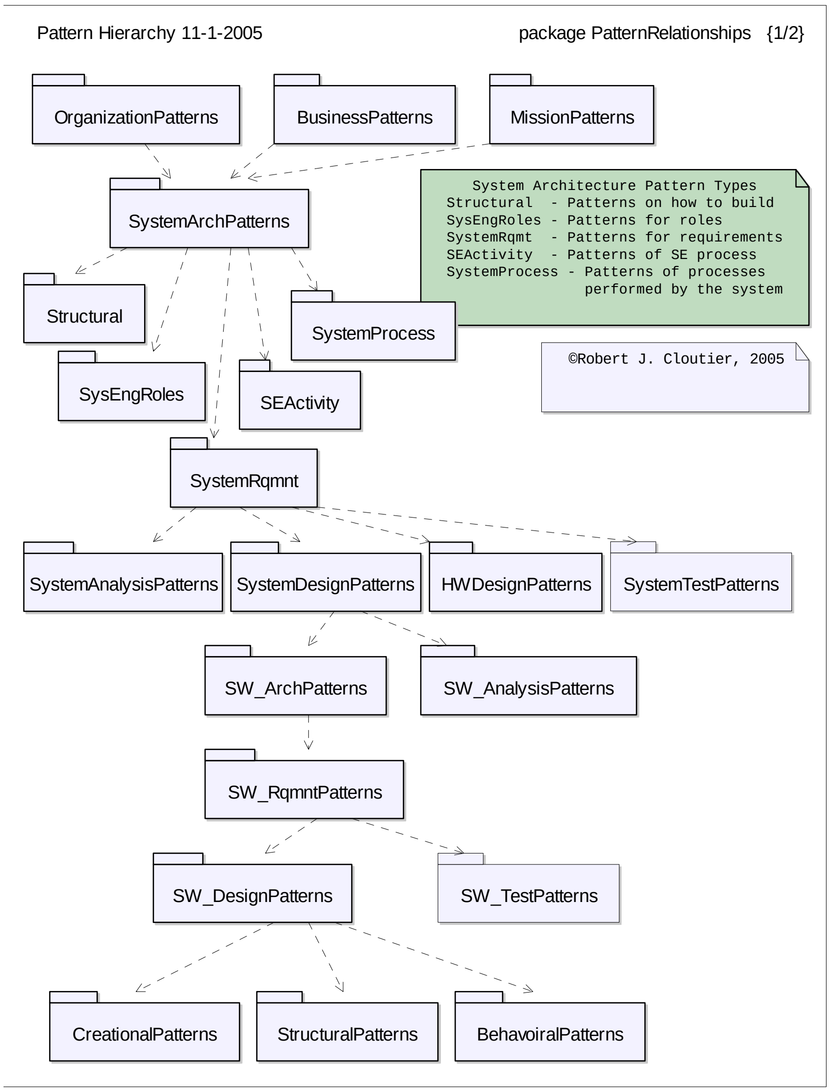


Figure 2 – Proposed System Pattern Hierarchy

In this proposed taxonomy, system architecture patterns are broken into:

1. Structural patterns
2. System Requirements patterns
3. Systems Engineering Activities patterns
4. Systems Engineering Roles patterns
5. System Process patterns

A structural pattern provides a physical pattern to follow when designing a part of the architecture. System requirements patterns prescribe the format of a properly formed requirement, or a collection of requirements that can be reused to describe desired functionality. Activities patterns, also described as systems engineering process patterns, indicate how the process of architecting or systems engineering is performed.

Systems Engineering role patterns help describe how the architecting/engineering role is performed. System Process patterns capture how the system does what it does by using control loops, algorithms, etc.

Carpenter [2002] discusses the systems architect's role and observes that the art-like quality of systems architecting depends on the architect’s ability to recognize complex system requirements patterns and to match those patterns to architecture solutions. He claims that through years of experience the architect is able to recognize relationships and patterns, and then apply the correct solution to the problem at hand.

The Argument against Patterns

The list of arguments against using patterns is shorter, but there are some.

The most obvious argument against using patterns is that they may stifle creativity and innovation when imposed on highly independent or creative individuals. The argument goes on to say that there will be no breakthroughs if organizations mandate that new architectures be assembled from proven patterns.

There appear to be three major reasons to avoid the use of patterns:

1. When addressing new or unique requirements which have not solved before
2. When the requirements require a unique solution, e.g. aesthetics over function
3. When the pace of technological change does not warrant the use of patterns

However, there are system designs that are so proven and effective, the case for defining and using a pattern to represent those system designs is overwhelming. The heuristic, then, is to innovate where it is most beneficial to the value chain, and use patterns for bread-and-butter work. In this manner, more effort can be used in applying creativity to the more challenging and higher-risk aspects of the system.

The second argument against the use of architecture patterns is harder to overcome because it is an organizational issue – patterns are of little use to the experts. The experts probably invented some of the patterns. Therefore, unless the expert is a visionary and understands the importance of training up the next generation of systems engineers, there is little reason for them to perform the documentation and validation of patterns. If the experts do not contribute to the patterns library, the junior systems engineer will find nothing there to help and will quit checking the library.

The third argument is self-evident – if technology is evolving so quickly, it is unlikely a solution will be implemented enough times to emerge as a pattern. Or, if the technology is evolving so rapidly, using older patterns in the solution will not satisfy the forces driving the problem context.

The problem with this argument is evident in the aerospace industry – the experts are aging, or have begun to retire. The younger engineers are re-learning standard designs because the experts with the understanding walked out the door. The design experience that enabled senior engineers to recognize a problem and a pattern solution are no longer available and that knowledge was not documented as a pattern. Therefore, costs are higher for a new system because there is little or no reuse from one system to another. If, as engineers, we don’t learn from history, we are destined to repeat it. What a colossal waste of time and energy.

Adopting Patterns for Systems Architecting

The adoption and standardization of patterns is due to professional peer reviews at conferences and in professional journals. When organizations standardize patterns, they foster reuse of designs and even code generated from architecture patterns. This reuse improves efficiency and productivity [Coplien 1997]. Based on Coplien's experience, it might be argued that applying patterns to complex systems architectures makes system complexity more manageable and affordable.

As was found at Bell Laboratories, [software] architecture patterns provide expert advice to novice architects. Architectural patterns can also help control the complexity of an architecture by standardizing it on a well-known and practiced pattern. In this sense, patterns generate architectures [Sanz 1999] [Douglass 2003] [Rubel 1995].

In a paper published in an IEEE Journal, James Coplien, working for Bell Laboratories in 1996, recalls the reasons for using patterns:

> "… *mining the patterns of classic embedded systems to capture the core competencies of their business… Why? We can trace availability and fault tolerance to patterns, and we have extracted those patterns from the minds of long-standing experts.*" (Coplien 1997).

Alexander began with form and context. Martin Fowler refers to patterns as useful ideas that may translate to another context. Senge describes his archetypes (process patterns) as a method for clarifying and testing mental models of systems. Brandon Goldfedder states that "One of the key goals of patterns is to capture the solutions to recurring problems (and the constraints or context in which they can be used) in a manner which is easily accessible to others." [Goldfedder 2002].

Patterns have been embraced in other engineering and technical disciplines. They may be beneficial to systems engineers in controlling the complexity of systems, providing a common language to discuss similar aspects of systems, and capturing good architecture concepts for reuse. However, research is necessary before systems engineers can use them extensively. There is anecdotal evidence that systems architects on large projects are applying patterns. This is consistent with Hahsler's findings that the larger the team, the more useful patterns become.

Conclusion

This paper has presented an historical perspective to the use of patterns in engineering disciplines. The value of patterns in the improvement of communications between the best open-source software developers was shown. The paper went on to discuss why a systems engineering organization should capture valuable implicit knowledge in the form of patterns when appropriate. After a discussion of the problems with patterns, the author concluded the paper with a discussion of the potential value of implementing patterns within the systems engineering discipline.

References

[Alexander 1964] Alexander, Christopher. Notes on the Synthesis of Form. Cambridge: Harvard University Press, 1964.

[Alexander 1977] Alexander, Christopher. A Pattern Language. New York: Oxford University Press, 1977.

[Alexander 1979] Alexander, Christopher. A Timeless Way of Building. New York: Oxford University Press, 1979.

[Buschmann 1996] Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture: A System Of Patterns. West Sussex, England: John Wiley & Sons Ltd., 1996.

[Carpenter 2002] Carpenter, Robert Glenn. System Architect's Job Characteristics and Approach to the Conceptualization of Complex Systems. Doctoral Dissertation presented to the Faculty of the Graduate School University of Southern California, 2002.

[Coplien 1997] Coplien, James O. "Idioms and Patterns as Architectural Literature". IEEE Software Special Issue on Objects, Patterns, and Architectures. January 1997.

[Douglass 2003] B.P. Douglass, Real-Time Design Patterns: Robust Scalable Architectures for Real-Time Systems. Reading, MA: Addison-Wesley, 2003.

[Gamma 1995] Gamma, E., Helm, R., Johnson, J., Vlissides, J., Design Patterns: Elements of Reusable Object Oriented Software. MA: Addison-Wesley. 1995.

[Goldfedder 2002] Goldfedder, Brandon. The Joy of Patterns. Boston: Addison-Wesley. 2002.

[Gross 2001] Gross, D. Yu, E., From Non-Functional Requirements to Design through Patterns

Requirements Engineering. Springer-Verlag. 6(2001) 1: 18-36.

[Hahsler 2004] Hahsler, Michael. 2004. Free/Open Source Software Development. Pages 103-124. Edited by Koch Stefan. Idea Group Publishing.

[Haskins 2004b] Haskins, Cecilia and Harrison, Neil, "Introduction to Patterns Through Writing SE Patterns Tutorial", Proceedings of the INCOSE 14th Annual International Symposium (Toulouse, France, June 20th – 24th, 2004).

[Haskins 2005a] Haskins, Cecilia, "Application of Patterns and Pattern Languages to Systems Engineering." Proceedings of the INCOSE 15th Annual International Symposium (Rochester, NY, July 10th-13th, 2005).

[Haskins 2005b] Haskins, Cecilia and Harrison, Neil, "Introduction to Patterns Through Writing SE Patterns Tutorial", Proceedings of the INCOSE 15th Annual International Symposium (Rochester, NY, July 10th-13th, 2005).

[Hole 2005] Hole, Eirik. "Architectures as a Framework for Effective and Efficient Product Development in a Dynamic Business Environment". Proceedings of the 2005 Conference on Systems Engineering Research, March 2005.

[Rubel 1995] B. Rubel, "Patterns for generating a layered architecture," in Pattern Languages of Program Design, D.C. Schmidt and J.O. Coplien, Eds. Reading, MA: Addison-Wesley, 1995, pp. 119-128.

[Salingaros 1999] Salingaros, Nikos, PhD. 1999. "Architecture, Patterns, and Mathematics". Nexus Network Journal, Volume 1, pages 75-85. Downloaded 10/25/04. URL: http://www.math.utsa.edu/sphere/salingar/ArchMath.html

[Sanz 1999] R. Sanz, C. Sánchez-Largo, and A. Juez, "Patterns in control systems" (in Spanish), Universidad Politécnica de Madrid, Tech. Rep. ASLAB-R-1999-006, Oct. 1999.

[van Zyl] vanZyl, Jay & Walker, A.J., "A Pattern Architecture: Using Patterns to Define Overall Systems Architecture". Downloaded 10/22/04. URL: http://osprey.unisa.ac.za/saicsit2001/Electronic/paper37.pdf

About the author

Rob works for Lockheed-Martin Corporation in Moorestown, NJ where he is a systems engineer and architect responsible for modeling systems and systems of systems architectures. He is also a Doctoral Candidate and occasional systems engineering guest lecturer at Stevens Institute of Technology in New Jersey, USA.

Rob holds a B.S. from the United States Naval Academy and an M.B.A. from Eastern University where he is an adjunct professor. He has over 20 years experience in systems engineering, software engineering, and project management in both commercial and defense industries.

Robert J. Cloutier, Principle Engineer
Lockheed Martin Corporation, MS2
Mailstop 13000-2
199 Borton Landing Road
Moorestown, NJ 08057
(856) 638-7437
robert.j.cloutier@lmco.com
rob@calimar.com

About Telelogic

Telelogic® is a leading global provider of solutions for automating and supporting best practices across the enterprise – from powerful modeling of business processes and enterprise architectures to requirements-driven development of advanced systems and software. Telelogic's solutions enable organizations to align product, systems, and software development lifecycles with business objectives and customer needs to dramatically improve quality and predictability, while significantly reducing time-to-market and overall costs.

To better enable our customers' drive towards an automated lifecycle process, Telelogic supports an open architecture and the use of standardized languages. As an industry leader and technology visionary, Telelogic is actively involved in shaping the future of enterprise architecture, application lifecycle management and customer needs management by participating in industry organizations such as INCOSE, OMG, The Open Group, Eclipse, ETSI, ITU-T, the TeleManagement Forum, and AUTOSAR.

Headquartered in Malmö, Sweden with U.S. headquarters in Irvine, California, Telelogic has operations in 20 countries worldwide. Customers include Airbus, Alcatel, BAE SYSTEMS, BMW, Boeing, DaimlerChrysler, Deutsche Bank, Ericsson, General Electric, General Motors, Lockheed Martin, Motorola, NEC, Philips, Samsung, Siemens, Sprint, Thales and Vodafone.

For more information, please visit www.telelogic.com