

### ***What is the Smallest Model of a System?***

Bill Schindel

bill.schindel@incose.org

How we represent systems is fundamental to the history of mathematics, science, and engineering. Recently, the minimality of that representation is of interest: Scientifically, because the size of its “minimal representation” is used to define a system’s complexity, and practically because the size and redundancy of engineering specifications challenge the effectiveness of real systems engineering processes. What is the smallest representation of a system?

#### **Size matters!**

This article summarizes a (potentially least) upper bound on the size of effective representation *for systems engineering purposes*. It draws more directly on scientific traditions for representing behaviour as physical interaction than does traditional systems engineering. When used for families (product lines, ensembles), this representation is subject to significant compression by the use of patterns. We also suggest that these patterns are emergent when the interaction rules of the development process are properly arranged.

#### *Practical representation challenges of traditional systems engineering*

Traditional documentation of system requirements, design specifications, failure mode and effects analyses, test plans, operations and maintenance procedures, and other task-specific system representations over the life cycle of a system can exceed thousands of pages. The effectiveness of these representations may be questioned in light of the following typical experiences: A requirements document, read by three systems engineers, produces three interpretations of its meaning--quite a different experience from three electrical engineers interpreting a properly constructed electrical schematic diagram. Whereas the discovery of an ambiguity in a schematic “blueprint” is considered exceptional, ambiguities in system requirements documents are commonplace and frequently tolerated as the state of the art. Determining completeness and consistency of a specification document is usually a highly subjective assignment, requiring very experienced human reviewers. The systems engineering process is frequently described [ANSI/EIA-632-1998, ISO/IEC 15288 2002, Haskins 2006], but the system representations it produces and consumes (our subject here) remain a challenge.

#### *Complexity science*

Complexity, a seemingly intuitive idea, has become the subject of formalization and study, including both the natural and human-engineered world. Initial efforts sought a theoretical basis for expressing complexity measures or otherwise understanding it [Li and Vitany 1997, Chaiten 2005, Kauffmann 2000]. They have more recently turned to the practical implications of emergent complexity science for engineering processes [Bar-Yam 2003b, 2005, Braha et al 2006, Kuras and White 2005, Schindel 1996]. Some efforts have studied minimal information required to describe a system, as a measure of its complexity. Others have introduced “complex systems engineering” (CSE) terminology in connection with understanding engineering problems or classifying systems, in situations such as highly interconnected systems (networks), adaptive systems, systems embedding humans, issues of scale and scope, ideas about types of emergence, or engineering project failures [Braha et al 2006, Bar-Yam 2003b]. Some studies have focused from the outset on the problems of human engineering or other organic intentional processes in connection with complex engineered systems [Ashby 1957, INCOSE HSI]. INCOSE formed the System Science Enabling Group [INCOSE SSEG] in recognition of the connection between systems science and systems engineering.

#### *System patterns*

Ideas of “patterns” have a number of connected roots in science and engineering. Pattern recognition and classification have a mathematical theory and engineering practices [Duda 2001]. Patterns in engineered systems were recognized in building architecture, later inspiring software engineers, and more recently systems engineers. [Alexander 1977, Gamma et al 1995, Haskins 2005,

Cloutier 2007, Schindel 2005b]. Initially expressed using traditional engineering structures (e.g., prose templates), patterns were later combined with model-based systems engineering (MBSE) to lead to pattern-based systems engineering (PBSE) [Schindel and Smith, 2002, Schindel 2005b].

### Constructing an efficient representation

Model-based representations have a traditional place in verifying that designs will satisfy requirements, or otherwise predicting system behaviour [Karayanakis 1993]. More recently, model-based representations have been used to represent system requirements [Mellor 2002, INCOSE MBSE, Schindel 2005a, SysML Partners]. In the design-oriented case, “model” frequently refers to mathematical descriptions of system physical make-up, often modelling from first principles to create mathematical models that can be analyzed or simulated. In the requirements case, “model” extends this idea to describe desired functional behaviour. In both cases, the term “model” means a *formal* (according to agreed upon rules), *explicit* (core content not implicitly depending on other assumed knowledge), and *unambiguous* (not subject to multiple interpretations) description. As shown in Figure 1, there are three components in a model-based engineering setting: The model, the system modelled, and the model interpreter(s). We want the model to be interpreted with desired process outcomes (e.g., easy, consistent, and unambiguous interpretation, optimality of design, etc.). Global efforts [ISO 10303 AP233; Mellor 2002] are working toward the exchange and interpretation of model data by machines and people, for purposes of simulation, procurement, fabrication, etc.

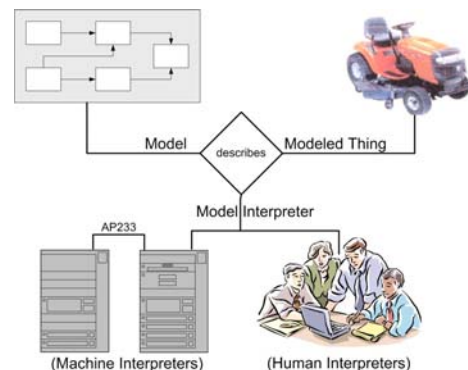


Figure 1: The setting for model-based systems engineering

#### *A metamodel*

A metamodel is a model of other models. We utilize the S\*-metamodel (summarized by Figure 2), a relational / object information model used in the Systematica™ methodology to describe requirements, designs, and other (verification, failure analysis, etc.) information in S\*-models. These may be represented in SysML™, database tables, or other languages. We have applied these to systems engineering in mil/aero, transportation, communication, medical and health care, consumer products, construction, manufacturing, and other environments [Schindel and Smith 2002, Schindel 2002, 2005b].

S\*-models represent interactions as explicit objects. This goes to the heart of what we mean by systems in the engineering and scientific world, and expresses ideas of emergence: By “system”, we mean anything that has interacting components. By “interact”, we mean that one component impacts the state of another component. By “state”, we mean a property of a component that impacts its current or future behaviour. By “behaviour”, we mean a component’s interactions with other components. This is the intentionally circular, relational perspective of the scientist, engineer, or mathematician that has helped describe the natural world since Newton. In this perspective, an interaction is holistic, with two or more components playing logical roles. The “emergent” properties of the interaction are associated with the whole, not any single component. Behaviours of individual components are described by requirements statements as input-output characteristics of their roles [Schindel 2005a].

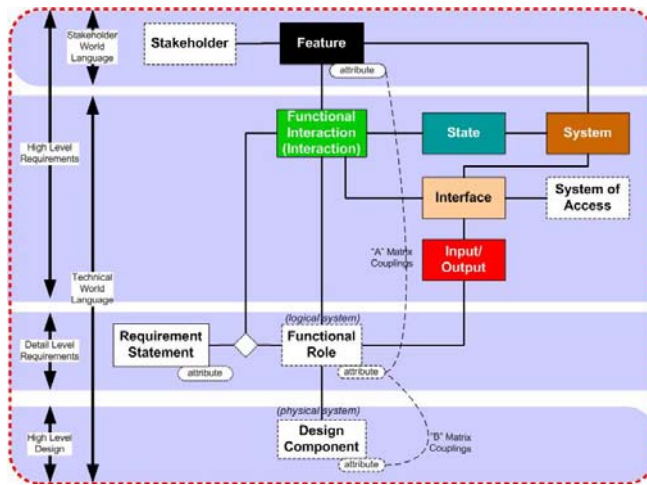


Figure 2: Summary view of the S\* metamodel

### Minimality of representation

The S\* metamodel arose from the research question, “What is the smallest amount of information required to describe system requirements and design?” We won’t reproduce here the formal argument for minimality of S\* models--interested readers may contact the author. However, a summary of that argument is:

1. The sufficiency of S\* models of requirements and designs is first argued, with respect to intended use of the information. Here the *uses of systems engineering information* enter the picture.
2. The minimality of S\* models is established by showing that no metaclass (see Figure 2) of information in an S\* model is redundant with information in another metaclass, and showing that omission of any component results in loss of sufficiency—including classes versus instances.

This argument is “constructive”: Rather than arguing that a minimal model exists, we actually construct it--not the case in most algorithmic information theory. However, this argument does not assert uniqueness: There may be other models no larger that also represent the same system.

### Measures of model complexity

Models communicate information, as quantified in communication theory [Shannon 1963]. More recently, complexity of objects has been quantified in algorithmic information theory (AIT or Kolmogorov complexity) using the “smallest program” capable of constructing the object or its behaviour [Li and Vitany 1997, Chaiten 2005]. The minimality of S\* models (measured in bits) also has several practical sides:

1. Clarifying “too small” versus “big enough” models: The S\* metamodel reminds us of types of systems engineering information that, if omitted, will leave us with an incomplete description of a subject system’s requirements, design, or connecting relationships. A practical example is the use of states in a requirements model, reminding us that for any requirement statement, “when does this requirement apply?” is a fair (and often not explicitly answered) question. We may omit this information for pragmatic reasons, but are reminded of what we have not communicated.
2. Reducing redundancy and associated inconsistency: Although documents or other task-oriented views generated from an S\* model may be redundant, the information in an S\* model is not. The consistency of a large number of redundant derived documents and views is easier to maintain or check against a single minimal model.
3. So, how big? How does an S\* model-based compare in size to a traditional systems engineering prose-based description? A practical discovery is that a typical S\* model of technical requirements is more complete than a corresponding traditional technical requirements document. More complete, it is bigger, not smaller! Figure 3 illustrates some typical sizes.

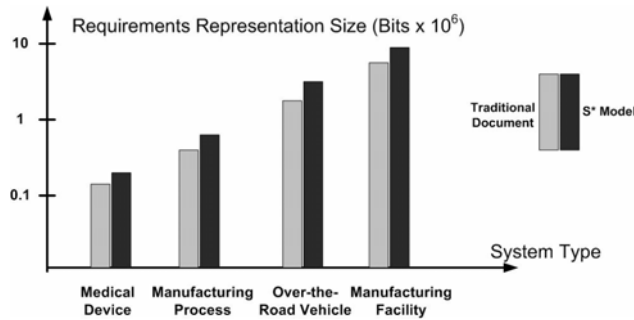


Figure 3: Typical sizes for models and traditional systems engineering documents

### Using patterns to compress models

Descriptions of systems engineering processes often focus on engineering a single system, beginning with stakeholder needs [Buede 2000]. However, real projects are often concerned with engineering in product line enterprises that specialize in products or systems with a high degree of similarity across projects or generations. How should engineering processes take maximum advantage formally of a large base of enterprise domain expertise about systems that are (partially) similar across evolutionary versions? What are formal engineering methods to deal directly with “variable sameness”? Something different is needed than simply repeating “from scratch” SE process for each new system variation. [ANSI/EIA 632 1998] was originally titled “Processes for Engineering A System” (emphasis added), consistent with historical single system SE focus. A question is how engineered systems at the base of Figure 4 should depend on descriptions of the general system at its apex.

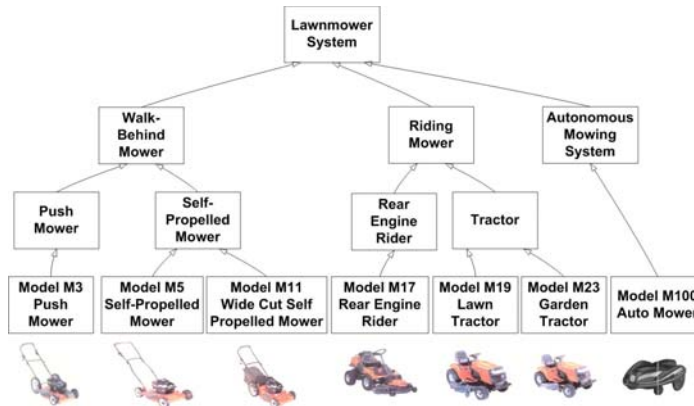


Figure 4: Class hierarchy of systems

### Model-based patterns

In this paper, patterns are re-usable, configurable S\* models of families (ensembles, sets, classes) of systems [Schindel and Smith 2002], ready to be configured or specialized for use in individual project situations fit to the pattern.

The term “pattern” in engineering has a number of historical interpretations. For example, less formal patterns than these model-based structures may be expressed in prose template form as a re-usable solution to a problem in a context [Alexander 1977, Haskins 2005, Cloutier 2007]. The relationship between such prose-based patterns and the model-based patterns described here resembles the relationship between prose-based engineering and model-based engineering. The “patterns” of statistical pattern recognition and pattern classification have a longer history in engineering, science, and mathematics [Duda 2001]. As in those cases, we are interested in model-based representations that “soak up” as much of the description of real systems as needed for practical purposes, while parameterizing configurable variations across them.

In complex systems, a higher degree of inter-component interactions is typically present, and they may be less well understood by traditional decomposition engineering analyses using first principles. For such systems, increased concurrency in pursuit of understanding and multiple designs, as well as more use of rapid incremental refinement in an evolutionary fashion, has been recommended [Bar-Yam 2005]. We have found model-based patterns to be an effective means of representing what is known and not known about complex systems, and found that concurrent pattern improvement and configuration activities are typical across multiple levels of abstraction (Figure 5).

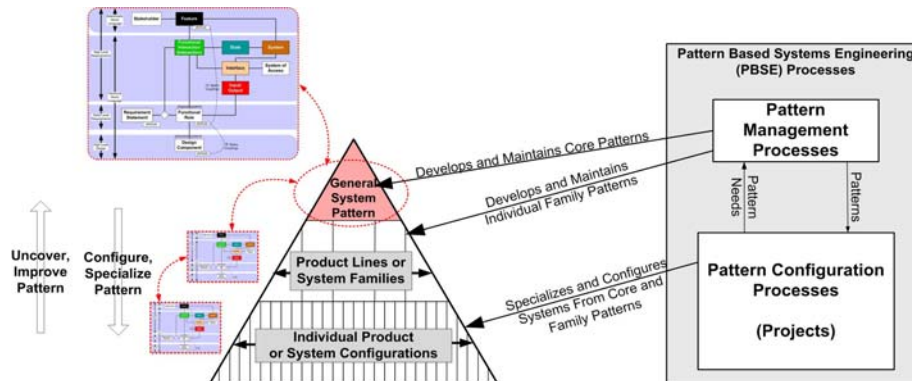


Figure 5: Model-based patterns and related systems engineering processes

Pattern-based systems engineering (PBSE) processes fall into two main categories (Figure 5):

- A. **Pattern Configuration Processes:** Incrementally configure and re-configure requirements and designs for specific projects, from an available Pattern. These configurations describe systems in the “language” established by the Pattern—a configured population of its components, relationships, and attributes, evolving during a project.
- B. **Pattern Management Processes:** Accumulate incremental learning about the underlying pattern, from the activities occurring across different projects.

#### *Compression of models, using patterns*

A pattern describes a family of similar systems. A specific model of one of those systems can be created by configuration or specialization of the pattern. The configurations table of Table 1 illustrates an extract of key variable configuration data for different configurations of a single requirements/design pattern, alternately described by Figure 4. Such tables are familiar in catalogues of products. Configuration, involving only population and attribute values, is simpler than specialization. When specialization is required, the Gestalt rules [Schindel 1996] provide a powerful paradigm for understanding what constitutes valid specialization consistent with a given pattern.

The configuration table illustrates how patterns facilitate compression of the representation of a system. Each column of a configuration table is a compressed representation of a system, with respect to (“modulo”) a common family pattern. Given a known pattern, a configuration represents the requirements and design of a system described by that pattern, in compressed form compared to the pattern. The amount of compression involved is very large—typically orders of magnitude (see Figure 6(a)). This compression also tells us how much of a pattern is not unique (i.e., is a transcendent or immutable aspect of a system family’s pattern—a system family’s “essence”). We have found that domain experts on projects frequently express surprise at how large a fraction of their expertise applies to virtually all the systems of their enterprise domain—i.e., is expressed by the top general system pattern of Figures 4 and 5. Apparently, humans intuitively believe that the complexity of their systems includes a much higher proportion of project-unique content than actually turns out to be the case.

Table 1: Example Configuration Table

Lawnmower Product Line: Configurations Table									
		Units	Walk-Behind Push Mower	Walk-Behind Mower	Walk-Behind Self-Propelled Mower	Riding Rider	Riding Tractor	Riding Mower Tractor	Autonomous Autonomous
			Push Mower	Self-Propelled	Wide Cut	Rider	Lawn	Garden	Auto Mower
	Model Number		M3	M5	M11	M17	M19	M23	M100
	Market Segment		Sm Resident	Med Resident	Med Resident	Lg Resident	Lg Resident	Home Garden	High End Suburban
Power	Engine Manufacturer		B&S	B&S	Tecumseh	Tecumseh	Kohler	Kohler	Elektroset
	Horsepower	HP	5	6.5	13	16	18.5	22	0.5
Production	Cutting Width	Inches	17	19	36	36	42	48	16
	Maximum Mowing Speed	MPH	3	3	4	8	10	12	2.5
	Maximum Mowing Productivity	Acres/Hr			1.6				
	Turning Radius	Inches	0	0	0	0	126	165	0
	Fuel Tank Capacity	Hours	1.5	1.7	2.5	2.8	3.2	3.5	2
	Towing Feature						x	x	
	Electric Starter Feature				x	x	x	x	
	Basic Mowing Feature Group		x	x	x	x	x	x	x
Mower	No. of Anti-Scalping Rollers		0	0	1	2	4	6	0
	Cutting Height Minimum	Inches	1	1.5	1.5	1.5	1	1.5	1.2
	Cutting Height Maximum	Inches	4	5	5	6	8	10	3.8
	Operator Riding Feature					x	x	x	
	Grass Bagging Feature		Optional	Optional	Optional	Optional	Optional	Optional	
	Mulching Feature		Standard	Factory Installed	Dealer Installed				
	Aerator Feature					Optional	Optional	Optional	
	Autonomous Mowing Feature								x
	Dethatching Feature					Optional	Optional	Optional	
Physical	Wheel Base	Inches	18	20	22	40	48	52	16
	Overall Length	Inches	18	20	23	58	56	68	28.3
	Overall Height	Inches	40	42	42	30	32	36	10.3
	Width	Inches	18	20	22	40	48	52	23.6
	Weight	Pounds	120	160	300	680	705	1020	15.6
	Self-Propelled Mowing Feature			x	x	x	x	x	x
	Automatic TransFeature							x	
Financials	Retail Price	Dollars	360	460	1800	3300	6100	9990	1799
	Manufacturer Cost	Dollars	120	140	550	950	1800	3500	310
Maintenance	Warranty	Months	12	12	18	24	24	24	12
	Product Service Life	Hours	500	500	600	1100	1350	1500	300
	Time Between Service	Hours	100	100	150	200	200	250	100
Safety	Spark Arrest Feature		x	x	x	x	x	x	

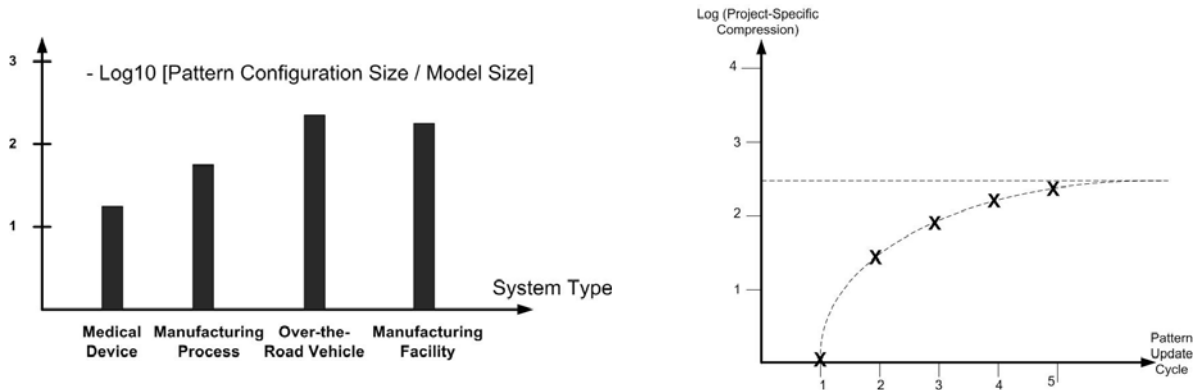


Figure 6 (a) and (b): Compression, using patterns

We use the above approach to generate system models from compressed system configuration data, using a universal generation algorithm parameterized by a domain pattern. This is a practical approach that parallels the more mathematical/scientific but non-constructive approach of Kolmogorov or algorithmic information theory (AIT) [Bar-Yam, 1997]--which relies upon theoretical Turing Machines. A key difference is that the practical methodology includes a means of constructing the models involved. The non-constructive theoretical results of AIT help to inform us of what is possible.

Recent literature concerned with emergence in systems engineering offers a number of ideas on how emergence may be usefully defined and characterized [Bar-Yam 2004a]. Modelling the systems engineering process as an intelligent system creates a framework for viewing emergence over time as discovered pattern content. One way to measure convergence or progress in such time-based emergence is to use a stakeholder feature attribute based metric. Another approach, connected to the compression described above, is to view learning over time as improving a modelled domain requirements pattern so that it better compresses the requirements of various system projects presented to it. Refer to Figure 6(b).

## References

- Alexander, Christopher; Sara Ishikawa, Ingrid Fiksdahl-King, Shlomo Angel. 1977. *A pattern language: Towns, buildings, construction*. New York: Oxford U. Press.
- ANSI/EIA-632-1998. Processes for engineering a system.
- Ashby, W. Ross. 1957. *An introduction to cybernetics*. London: Chapman & Hall.
- Bar-Yam, Yaneer. 1997. *Dynamics of Complex Systems*. Reading, MA: Addison-Wesley.
- . 2003b. When systems engineering fails—toward complex systems engineering. *Proceedings of the International Conference on Systems, Man & Cybernetics*, Vol 2, 2021-2028. Piscataway, NJ: IEEE Press.
- . 2004 a. A mathematical theory of strong emergence using multiscale variety. *Complexity*. Vol. 9, No. 6.
- . 2005. About engineering complex systems: multiscale analysis and evolutionary engineering. ESOA 2004, LNCS 3464, pp 16-31, Spinger-Verlag, 2005.
- Braha, D., A. Minai, Yaneer Bar-Yam, eds. 2006. *Complex engineered systems: Science meets technology*, City: Springer.
- Buede, Dennis M. 2000. *The engineering design of systems: Models and methods*. New York: John Wiley & Sons, Inc.
- Chaitin, Gregory. 2005. *Metamath: The quest for omega*, New York: Pantheon, 2005.
- Cloutier, Robert J., Dinesh Verma. 2007. Applying the concepts of patterns to systems architecture. *Systems Engineering*. Wiley. Vol 10, No. 2. pp 138-154.
- Duda, Richard. O., Peter E. Hart, David G. Stork. 2001. *Pattern classification*, (2nd ed.), New York: Wiley.
- Gamma, E., R. Helm, Ralph Johnson, J. Vlissides. 1995. *Design patterns: Elements of reusable object-oriented software*. Reading, MA: Addison-Wesley.
- Haskins, Cecilia. 2005. Application of patterns and pattern languages to systems engineering. Paper presented at the 15th annual international symposium of the international council on systems engineering, Rochester, NY.
- Haskins, Cecilia, ed. 2006. *INCOSE systems engineering handbook, Version 3*. Seattle, WA: International Council on Systems Engineering.
- INCOSE HSIg web site. <http://www.incose.org/practice/techactivities/wg/hsi/>
- INCOSE MBSE web site. <http://www.incose.org/practice/techactivities/modelingtools/mdsdwg.aspx>.
- INCOSE SSEG web site. <http://www.incose.org/practice/techactivities/wg/sseg/>
- ISO 10303 AP233 web site. <http://www.ap233.org/>
- ISO/IEC 15288: 2002. Systems engineering – System life cycle processes. Geneva: International Organization for Standardization.
- Karayanakis, N., *Computer-assisted simulation of dynamic systems with block diagram languages*. CRC Press, 1993.
- Kauffman, Stuart. 2000. *Investigations* New York: Oxford University Press.
- Kuras, M. L., B. E. White. 2005. Engineering enterprises using complex-system engineering. Paper presented at the annual international symposium of the International Council on Systems Engineering, July, Rochester, NY.
- Li, Ming, Paul Vitany. 1997. *An introduction to Kolmogorov complexity and its applications*. Second edition. Springer.
- Mellor, Stephen; Marc J. Balcer. 2002. *Executable UML: A foundation for model-driven architecture*. Boston: Addison-Wesley.
- Schindel, William D. 1996. Systems engineering: An overview of complexity's impact. Tech Paper 962177, SAE International.
- . 2002. Does our SE house have a foundation? Paper presented at the INCOSE Crossroads of America Chapter technical program May 22, Peoria, IL.
- . 2005a. Requirements statements are transfer functions: An insight from model-based systems engineering. Paper presented at the annual international symposium of the International Council on Systems Engineering, July, Rochester, NY.
- . 2005b. Pattern-based systems engineering: An extension of model-based systems engineering. INCOSE TIES tutorial presented at 2005 INCOSE Symposium.
- Schindel, William D., Vern R. Smith. 2002. Results of applying a families-of-systems approach to systems engineering of product line families. Technical Report 2002-01-3086. SAE International.
- Shannon, Claude. 1963. *A mathematical theory of communication*. Champaign, IL: University of Illinois Press.
- SysML Partners web site. <http://www.sysml.org/>