System Architecture Entropy

Dr. Robert Cloutier
School of Systems and Enterprises
Stevens Institute of Technology
Castle Point on the Hudson
Hoboken, NJ 07030, USA

Mary Bone
School of Systems and Enterprises
Stevens Institute of Technology
Castle Point on the Hudson
Hoboken, NJ 07030, USA

Dr. Dinesh Verma
School of Systems and Enterprises
Stevens Institute of Technology
Castle Point on the Hudson
Hoboken, NJ 07030, USA

Kim Sommer
Naval Surface Warfare Center
Crane, IN 47522, USA

Abstract

This paper will introduce the concept of Architecture Entropy in the context of systems engineering and complex systems. Entropy has been explored and discussed in other domains, drawing an analogy to its origins in physics. In its original context, entropy describes the degree of randomness, or level of chaos, that occurs over time. The purpose of this research is to explore application of the concept of entropy for understanding the vitality of legacy system architectures. The discussion is supported by examples of current architecture challenges that may be attributed to architectural entropy. Specific examples are presented to better understand the applicability of architectural entropy.

Introduction

There exists a considerable body of knowledge related to the topic of systems architecture. Searching the IEEE Xplore database, more than 6500 articles matching "system architecture" can be found. The majority of the literature regarding system architecture focuses on three topics 1) the general definition of system architecture and different types of system architectures, 2) applying system architecture and, 3) how to produce better system architectures through solutions to observed system architecture issues. This paper will begin to move beyond the current literature, delve into the factors involved in the system architecture evolution issues. Isaac [1994] recognized the need for this focus and wrote "emphasis from controlling requirements to controlling the system architecture" must be made to support evolving systems. This research begins to uncover some of the factors relating to complex system architecture which need to be controlled to provide the desired output (i.e. sustainable life cycle). This paper will set forth the proposal that these factors are actually variables in the entropy of the system architecture. The goal of this paper is to introduce the idea of system architecture entropy as a key variable in understanding system architectures as they evolve over time, and identify this as an area requiring further research.

Current Uses of Entropy

Classical View of Thermodynamic Entropy. Entropy, as defined from the classical thermodynamic perspective is a measure of energy that is unavailable to do useful work. This perspective

provides a macroscopic view and can also be described as a measure of disorder [Fishbane et al 1996]. A key aspect of entropy in the classical meaning is that entropy increases as a system undergoes an irreversible process such as the transition from ice to water. It is also of interest to note that changes in entropy are path independent. This independence means that a system that goes from state X to state Y will have the same change of entropy, independent of how it gets from state X to state Y. As defined by Clausius in 1865, entropy is a function of the state of the system is:

$$S = Q / T$$

Where S is entropy, Q is heat content and T is Temperature.

Entropy in Information Theory. Claude Shannon's 1948 seminal paper defined entropy in terms of a communications system. Shannon's Entropy demonstrated how much information in a message is useful based on the probability of receiving a message. The amount of information gain between states is inversely proportional to the logarithm of the probability of a state occurrence. Shannon's communication entropy is represented as:

$$H = -K \sum_{i=1}^{n} p_i \log p_i$$

where K is a positive constant, p_i is the probability of an event occurring, and log p_i is the uncertainty related to that event.

Traffic Management and Entropy. In the field of traffic management, entropy has been "adopted to describe the uncertainty of a system" [Hsu et al 2007]. The entropy used for traffic detection is based on the adaptation of Shannon's Entropy [1948] by Pal and Pal [1991]. Pal and Pal developed an exponential form for their entropy form. The overall effect is similar to Shannon's entropy except the inverse relation of information follows an exponential relation rather than a logarithmic curve. Hsu's vision-based detection method was developed to track traffic at night using the measured variation of exponential entropy. Exponential entropy is expressed as:

$$H = \sum_{i=1}^{n} p_i \exp(1 - p_i)$$

When applied to the traffic detection problem by Hsu [2007] the higher the entropy values will show higher variation and when traffic is light the entropy will be lower. This allows for traffic flow at night to be represented by entropy changes. This non-classical example of entropy shows that entropy has a useful place outside of classical thermodynamics.

Management Entropy. Another current use of entropy in a non-classical approach can be found when "measuring managerial complexity based on entropy theory" [Song et al 2004]. The theory of management entropy defined management entropy as "a measure of the contribution made by the elements of a system to that system considered as whole or total system" [Song et al 2004]. Song's form of entropy is represented as:

$$H = -\sum_{r=1}^{n} P_r \log P_r$$

This is functionally the same as the Shannon form when K=1. Song uses three aspects of management: information transformation, organizational function, and management structure with the entropy theory to demonstrate his theorem for decreasing management complexity. Song was able to demonstrate that by decreasing management entropy the managerial efficiency was increased and the complexity was decreased [Song et al 2004]. The significance of this is that downsizing, regrouping, and reformation of enterprises reduces complexity and increases efficiency. One significant contribution of Song's work is that the management entropy theorem puts forth a qualitative method for demonstrating the efficiency of management within an organization based on the organization's characteristics (management structure, information transformation, and etc). This is particular important to this research because system architecture characteristics will be used to eventually put forth a qualitative method for demonstrating the efficiency of a system architecture.

System Architecture Entropy

Parallels between management entropy discussed above, and the concept of architecture entropy can be drawn. Architecture entropy is implicitly stated in many papers that address the complications of system architecture evolution. The literature approaches solutions to the issue of a system architecture becoming more complicated as the system architecture evolves. This serves to only consider the symptoms, and does not address the systemic problem of architecture entropy. What is missing from the literature is the underlying problem of inherited entropy in an architecture that drives the complexity of architecture as the architecture matures. Many authors [Martin 2002, Isaac and McConaughy 1994, Wilson 1996, Lacerte 2000] discuss increased complexity of architectures but do not explore the set of driving forces behind that complexity. Instead, they go directly to offering a solution to the one driving force in which they have focused their research. These solutions vary and most can be traced to incomplete problem identification. Others begin to recognize the characteristics of architecture entropy, and then identify it by another name. For instance, Percivall [1994] called it "self-organized criticality" and identified it as a point when the complexity of a system will increase until it reaches a critical state. Percivall [1994] also noted that the complexity of systems increase and stated "Complexity tends to increase as functions and modifications are added to a system to break through limitations, handle exceptional circumstances or adapt to a world itself more complex." Another author Henderson [1990] discusses system architecture and places the inability of organizations to shed its preconceived ideas of its architecture in order for the system architecture to successfully evolve. From these references it can be seen that system architecture is important, complex, and solutions have been identified.

Architectures of complex systems are not only influenced by the problem they are solving but can also be influenced by the organizational structure in which they are created. Lacerte [2000] asserts that "Architectures affect organizations as much as organizations affect architectures." This means that complex system architecture is not developed without organizational influence. As the system succeeds, and is improved upon due to its success, the organization changes over time, and this organizational change is reflected in the system architecture's functional and/or physical organization. In fact, this organizational complexity may affect the ability for the sys-

tem architecture to evolve because the organization itself cannot cope with the necessary architectural change. As the architecture attempts to evolve within an organization that may have already undergone many structural changes since the inception of the system architecture, considerable tension may develop between the architecture and organizational structure.

This increased complexity is consistent with the classical definition of entropy in that when the original system architecture is conceived it has, by definition, some entropy. As the architecture evolves, or changes state, the entropy must also change. Understanding the effects of organizational influence and the documented record that architectures increase in complexity as they evolve, architecture entropy must increase with each architectural evolution. The literature does not contain a single paper that demonstrates or discusses architecture which becomes less complex as it evolves. Therefore it is concluded that an evolving architecture will only become more complex. These complexities increase the level of uncertainty, and this system architecture characteristic is architecture entropy.

Architecture Entropy Variables. New research into system architecture entropy has begun to explore and identify the critical variables to enable architecture evolution and organizational growth to occur in a complementary manner. Since companies invest much in their system architectures and many architectures are kept for numerous years this would be a large cost savings and advancement in the world of system architecture development.

Given the existing literature and the author's experience, architecture entropy appears to be a function of legacy interfaces, legacy internal components, organizational forces, customer resistance to change, supply chain management, and original system architecture. The legacy interfaces may impact and limit new capabilities provided by upgrades to the system. One example of how these interfaces can become a limiting factor to upgrades due to constraints can be seen in the increases of data transfer bandwidth. In some cases, hard wire interfaces may need to be upgraded to optical interfaces. Other examples are physical handling systems, and changes in process – such as is found in production assembly lines. This compounding change at the interface can have more impact on a system then is initially realized. Even changing or updating internal aspects of a system can impact the system complexity and affect other parts of the system that were believed to be isolated from those changes.

The legacy internal components of a system can impact architecture entropy when a change is made that impacts a component(s) in some manor. As Henderson [1990] stated "Architectural innovation is often triggered by a change in a component--perhaps size or some other subsidiary parameter of its design--that creates new interactions and new linkages with other components in the established product." These new linkages change the complexity of the system and therefore affect the architecture entropy of the whole system.

As presented earlier in this paper, organizational forces also affect entropy. The organizational structure at the time of architecture inception may have been smaller and leaner. As a system succeeds in the marketplace, the size of the organization necessary to support product line changes causes the organization size to grow, and the organizational structure to morph to support the growing product line. These organizational changes can impact the ability of the architecture to change or be upgraded within in that organizational structure. This was summed up by

Grady [1994] who stated “The intensity of interface problems in a developing system is directly proportional to the percentage of interfaces that possess development organization responsibility differences...” An additional organizational force is the communication structure within the organization. That is, how information is gathered, captured, stored, and transferred within and organization can impact the complexity of evolving system architecture. Customers and stakeholders may also impact the architecture entropy if they are only receptive to incremental changes – and resist revolutionary changes. The American auto industry was resistant to the revolutionary adoption of electric vehicles, but embraced E85 technology which is an incremental change to the gas powered automobile. As will be discussed later in this document this resistant to revolutionary change as greatly impacted the American auto industry today. The need to satisfy stakeholders sometimes causes engineering decisions, or system compromises, to be made. These changes and compromises will provide a local optimization which may not be an optimal change for the architecture and its increasing complexity.

Supply chain management can also force changes to a system that increases architecture entropy. The supply chain management is responsible for keeping track and monitoring parts so that issues such as obsolesce can be handled. If a part becomes obsolete and the system must be maintained then changes to the system can be forced that are not ideal and therefore increase architecture entropy.

The final identified component of architecture entropy is the team understanding of the original system architecture itself. Once the system architecture is instantiated, every detail must be understood to successfully upgrade or change it. Many times this idea of successfully understanding the whole architecture at a later time is not achieved and therefore increases the architecture entropy. This means that architecture entropy is not only a function of the architecture complexity but also the knowledge required to manage the evolving architecture.

Industry Examples of System Architecture Entropy

Auto Industry. The auto industry is a mature and well documented industry with over one hundred years of architecture evolution to study. As demonstrated by Gorbea [2008] automobile system architecture evolvability can be expressed using an Architecture Performance Index Versus the Performance. This index tracks the evolution of the automotive architecture from 1885-2008. The architectural performance was a function of power to weight ratio, maximum velocity, fuel efficiency, and manufacturers suggested retail price (MSRP). Though this performance index does not take into account every performance parameter it is meant to be a representative set that can be compared throughout the architectures life. Figure 1 shows that from the introduction of the internal combustion engine (ICE) architecture around 1920 until around 1948 the ICE architecture performance increased quickly but from 1948 to 1998 the performance of the ICE architecture remained virtually unchanged. This is an indication that there must be some force causing the architecture performance to slow. That force or part of that slowing force is due to architecture entropy.

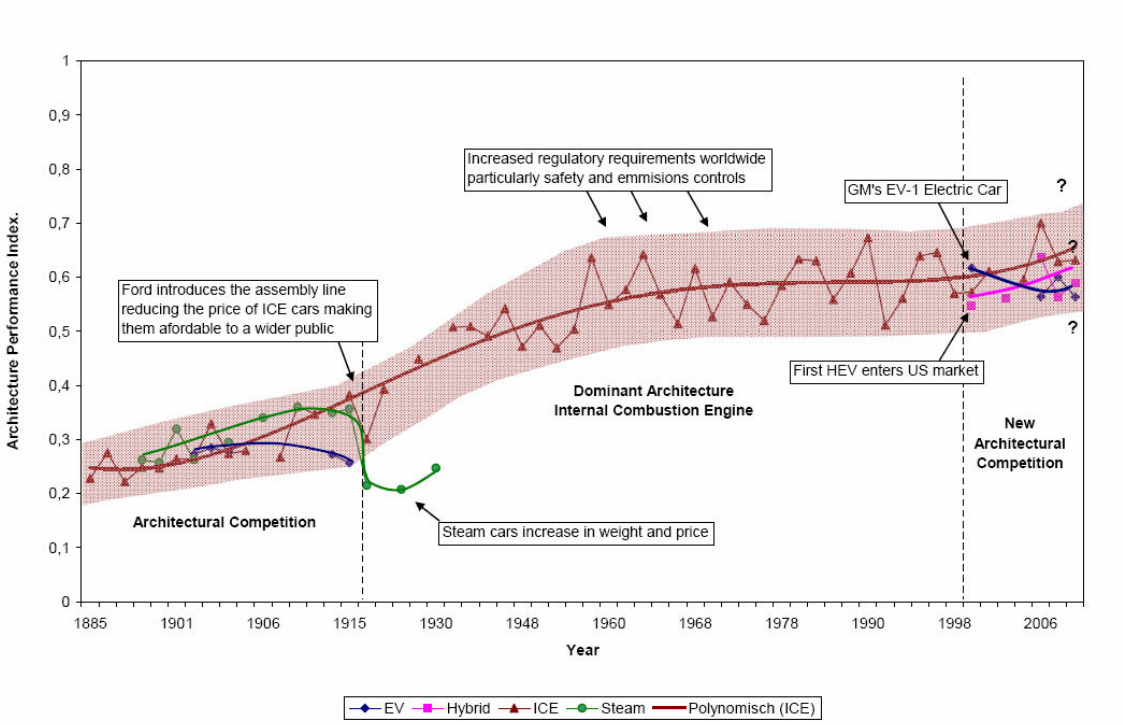

Figure 1 - Performance of various automotive architectures from 1885-2008 [Gorbea et al 2008]

The ICE architecture became the dominant architecture for the automotive industry around 1920. Henderson [1990] addressed dominant design and makes the point that once a dominant design is chosen, engineers do not reevaluate the decisions that have been made for that design. Henderson provides a supporting example from the automotive industry where "…engineers did not reevaluate the decision to use a gasoline engine each time they developed a new design." She goes on to point out that a dominant design allows for industries to focus on the components of that particular design and not focus on ideas outside that scope. For example once the ICE architecture became the dominant design the auto industry paid little to no attention to the steam or electric architectures. This could be seen as technology complacency where instead of looking for new technology outside the dominant architecture the industry starts to focus and elaborate only on the components in its dominant architecture. The reason this becomes important to architecture entropy is that with these dominant designs, certain behaviors begin to develop within an industry. Henderson refers to one of these behaviors is as information filters. Information filters are created to encompass the "knowledge of the key relationships between the components of the technology. These filters are created informally over time but they, along with formally captured information become the set architecture knowledge. This knowledge is how decisions are made regarding the evolution of the system architecture. Once these information filters are developed they make it harder for new information about emerging technology since that type of information is filtered out by the information filter. In respect to an established architecture, these filters can hinder any vision for new technology applied to the established architecture. These filters are implicit knowledge and result in information which is not formally captured not being passed from one generation to the next [Henderson 1990]. This limited sight (not thinking outside the box) and lack of formally capturing information from the filters can

then lead to an increased complexity of the system. The automobile industry has not been immune from this information filter affect. The automobile industry has stuck with the ICE architecture even while its architectural performance as decreased and while factors outside of the architecture (i.e. increased energy prices) have begged for change. This drives the automotive industry into sticking with what it knows and continues to create more and more complexity within the ICE architecture. One example of this is that instead of switching architectures, to electric or hybrid, altogether the American auto industry has decided to focus on automobiles which consume E85 (15% gas and 85% Ethanol). This resistance to change has only increased the complexity of the ICE Architecture and is another factor into the architecture entropy.

Naval Shipbuilding. The U.S. Navy provides opportunities to support the concept of architecture entropy. The Navy has been producing large scale complex systems for over 100 years. The large amount of data on Navy architectures makes it an appropriate industry to investigate.

The new DDG-1000 guided missile destroyer is an extreme case of architecture entropy. Initial guidelines for the ship platform were conceived in the strategic planning documents "…From the Sea" [O'Keefe et al 1992] and "Forward from the Sea" [Dalton et al 1994]. These documents defined the goals and objectives towards littoral warfare superiority in a post-Cold War world and the platforms that would support those plans. DDG-1000 has effectively been under development since "…From the Sea" was published in 1992 and an actual ship has yet to be launched. Currently some of the embedded technologies are not sufficiently mature for product implementation and the architecture is now not able to adapt to meet the evolving requirements. The DDG-1000 architecture stands in stark contrast when compared to the development of the LST (Landing Ship Tank) transports of World War II. The LST architecture was conceived by Winston Churchill after Dunkirk (1940). The United States took over redesign and engineering functions in November 1941 and the first mass-produced LST was launched in December 1942. Over one thousand (1051) LSTs were eventually produced before the war ended in August 1945 [Encyclopedia Britannica 2008]. This meant in only five years the LST architecture was conceived, redesigned, and mass produced compared to the stagnate DDG-1000 which after more then 16 years of architecture work has yet to even have one single unit produced.

Weapons systems are procured to have a "qualitative superiority over an enemy's weapon system or neutralize the enemy's superiority – not only today but in the future." [Charette 2008] The DDG-1000 program has evolved with the assumption that all competitors (i.e. expected enemies), their strategies, tactics, and development were understood. In late 2007 and early 2008 evidence came in that the DDG-1000's planned capabilities could not address an emergent threat scenario from ballistic cruise missiles [McCullough and Stiller 2008]. Unfortunately the architecture has been defined to the point that it cannot be changed or easily evolved to meet requirements outside its design envelope. The Navy is now dealing with a disruptive counter to the DDG-1000 architecture. We can establish a parallel by looking at this situation via the lens of Christensen's The Innovator's Dilemma [1997] and view the issue of ballistic cruise missiles as a disruptive innovation competing with an existing architecture (DDG-1000). A disruptive technology has the hallmarks of being initially less capable, cheaper, smaller, and easier to use [Christensen, 1997]. While ease of use may be arguable in the case of ballistic cruise missiles, the cost proposition of a single missile against a ship and trained crew is certain. In this case, the market is control of the littoral area.

The basic Christensen model is shown in Figure 2. In this figure, the horizontal axis represents time and the vertical axis represents a particular characteristic.

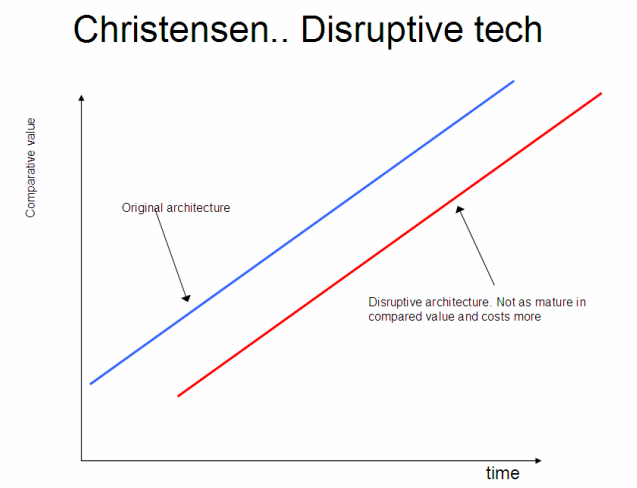

Figure 2 - Disruption in Marketplace [Christensen 1997]

In Figure 3 we see the effect a disruption causes to a system architecture. For simplicity of representation we will show disruptions as instantaneous state changes. In the real world this disruption would more likely match a logistics curve. The vertical axis now represents a scale of one characteristic relative to another. This is effectively adding another dimension for comparing the architectures. Adapting Song's [2004] multidimensional entropy model from management relations to architectures might be an appropriate exercise for future research.

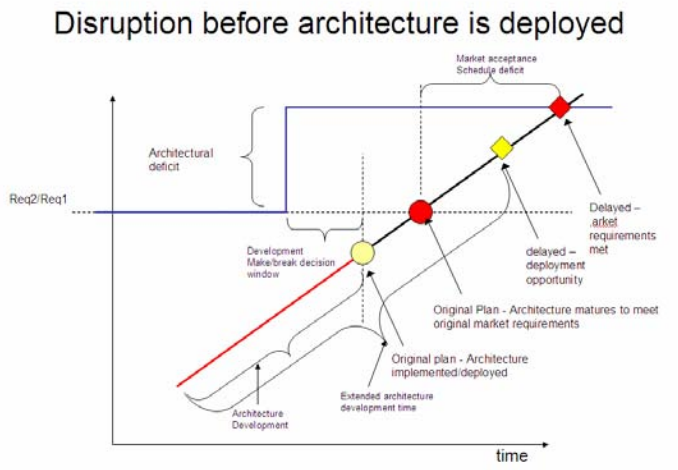

Figure 3 - Market timing effects from disruptions

We attempt to show the three-dimensional aspect of this model in Figures 4 & 5. These figures show how a later innovation can compete with established technology. The established technology is more capable but only along one dimension. It has little importance or value on the other

scale. This illustrates in concept the circumstances affecting the DDG-1000 program and its inability to overcome the entropy of it architecture.

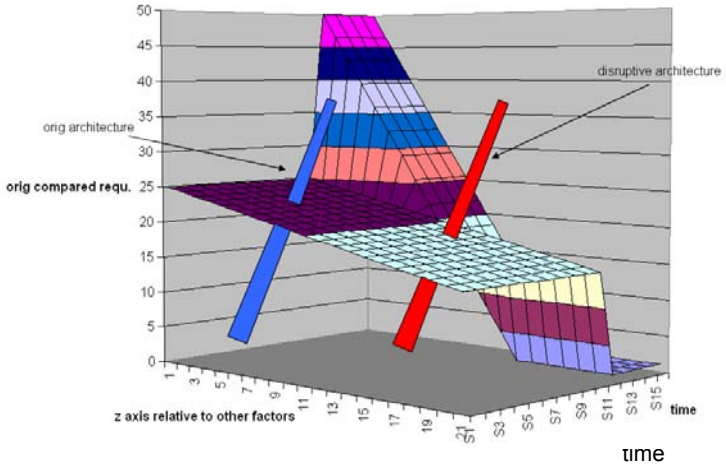

Figure 4 - Disruption in aspect view

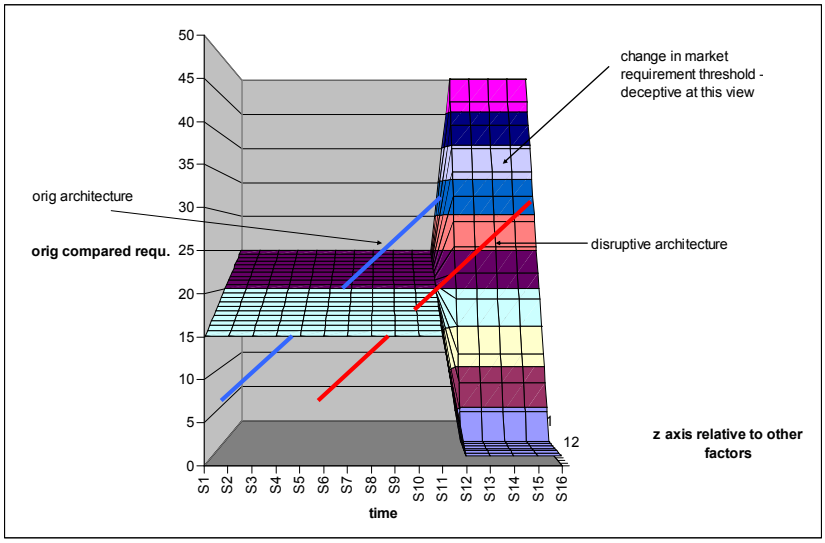

Figure 5 - Disruption in forward view

According to Wilson [1996] many Navy systems were designed with "architectural constraints which may no longer apply." This idea of architectural constraints that were designed into the system and at some point becomes non-applicable making the architecture harder to reengineer or upgrade. Wilson goes on to state that the "longer term goal is the ability to effectively transform complex legacy systems into systems that can evolve gracefully over time." The issue of system architecture evolveability is now being addressed by the Navy's implementation of a new system architecture process.

Traditionally, a ship's hull, mechanical, and electrical systems (HM&E) have been designed separately from the combat systems.[Wells 2006] The new Navy architecture process, called To-

tal Ship System Engineering (TSSE), intends to take the old methods of Navy ship building and merge them with current system standards and processes for integration. The Navy established a TSSE curriculum at its postgraduate school in 1992 [Naval Postgraduate School 2008]. The intent was to emphasize integration across technologies and disciplines using systems engineering practices. The TSSE discipline was used for the DDG-100 and had been used in the predecessor SC-21 program. [Clayton, 2002] Despite the integrated effort to meet the design requirements, we argue that the inability to meet updated strategic requirements can be described as architecture entropy

Microsoft Windows™. We could focus on computer operating systems in general. However the Microsoft Windows story is sufficiently detailed to provide a case study of architecture entropy.

Microsoft Windows has been a Microsoft product for over twenty years. The original Windows 1.0 (1985) was a graphical program loader that ran as an application on top of the DOS (Disk Operating System) environment. This fundamental architecture layering was used for Windows 2.0 (1987), 3.0 (1990), Windows 95 (1995), 98 (1998), and the Windows ME (2000) systems. The underlying core operating system of these environments all ran on some form of DOS with all the limitations that involved. In 1993 with the introduction of Windows NT, Microsoft had a parallel product line that leveraged off the original Windows user environment but was integrated in a way that made the GUI fully part of an operating system. This NT architecture ran parallel to the DOS architecture until Windows XP was introduced in 2001.

Microsoft's problem with architecture entropy started in its DOS days. As other companies started to use its architecture, these companies discovered undocumented functions that could enhance their product's performance [Schulman and Maxey 1990][Schulman 1994]. Microsoft had two main reasons to hold some functions as undocumented. First, undocumented functions could provide Microsoft a competitive advantage in applications development. Second, undocumented functions could be used in an experimental sense in that they could be changed or deprecated without having effect on the official system Application Programming Interface (API). Third, some undocumented functions were developed to apply to certain microprocessor generations and would be able to be removed with improved processors.

Unfortunately, for Microsoft, the information about undocumented functions was discovered outside of the company through information leaks, debugging, and reverse engineering. Third party applications such as Lotus 1-2-3 that exploited these functions had become highly successful. This created a customer requirement for Microsoft to retain those functions that allowed those third-party programs to run. Microsoft had maintained backward compatibility with previous versions of DOS. Thus newer versions of DOS had to retain undocumented but commonly used functions and, also replicate certain bugs that were commonly exploited.[1] Backward compatibility also limits the ability to incorporate improved features since they would break old functions. The overall effect increased the entropy of the DOS architecture by retaining functions that had little value for future utilization.

[1] This backward compatibility of functions and bugs is also practiced in the microprocessor world. Today Intel Pentium processors are designed to retain in their microcode functions and bugs that originated in the 8086 16-bit processors.

Other operating environments such as the Macintosh and Amiga with easy to use graphical interfaces were being developed that did not have the entropic weight that DOS had gained. Microsoft in turn developed the Microsoft Windows operating system to provide a way to break users from requiring DOS. The architectural strength and liability of Windows was in building a system with a common Application Programming Interface (API) that could use common libraries even though the underlying layer was vastly different (DOS, NT). This type of library system is referred to as "Dynamic Link Libraries" or DLLs. Using DLLs provides flexibility to a point. However, DLL's have versioning problems and a newer version of a DLL might break a program that had been running perfectly. Microsoft created a whole new entropic problem with this DLL nightmare.

The entropic load did not end with the DLLs. The API suffers with age as well. Microsoft has continued to implement undocumented functions in the Windows architecture [Schulman and Maxey 1992]. Certain functions and operations have become deprecated or changed with time. Microsoft tries to manage this via online downloads of major Service Packs to deal with fixing bugs and security issues as well as updating and changing the API. With the Service Packs clients are uniformly told that some programs will cease to work. These updates then require other software vendors to have ongoing programs to fix their products to run on Microsoft's updated platform. The result is the product architect unable to evolve the architecture because the Microsoft Windows enterprise and third-party applications will suffer.

To date, Microsoft's approach is not an architectural fix but merely a postponement of the inevitable. Part of the issue with architecture entropy is that exogenous forces provide competition. This requires the vendor to match features offered by competitors or create new innovative uses for their product. The latest incarnation of Microsoft's operating system – Vista, is intended to meet user requirements for new capabilities, ensuring Microsoft would maintain its market presence. This is a case of a technology push [Walsh et al 2002]. However, the architectural weight of the added capabilities built on top of the past Windows architecture has provided no new incentive to the market, and in large part, has been viewed by many as a disincentive. Vista requires markedly more storage space (hard drives) and much more memory to operate as smoothly for users as a less powerful computer running Windows XP. Anecdotally the perceived benefit Vista provides to customers over XP is marginal. This may be a hallmark of high entropy architectures. - a system architecture becomes so fragile that the overhead required for new capability comes at too high a price in performance. The system architecture has too much entropy to perform the intended mission effectively.

Conclusion

Entropy traditionally is defined as a measure of energy unavailable to perform work. In an organization, it is untapped capability, or organizational potential. We can extend this thought process to system architecture entropy, and define it as a measure of disorder in the system architecture that grows more disordered over time as the architecture evolves to satisfy new requirements. When this happens, the level of entropy increases. Examples of this include increases in the number of interfaces within an architecture - the entropy – or potential for unused capability increases

This paper has formally identified and put forth the concept of system architecture entropy. Since one goal of legacy systems architecture is to evolve existing systems in a graceful manner, the initial system architecture must be designed with the full understanding of system architecture entropy. The factors that go into architecture entropy were identified and discussed.

System engineers performing with architecting responsibilities can immediately begin to apply the ideas and concepts set forth in this paper to their everyday practice. While many have experienced the organizational influence on system architecture, beginning to address this phenomena in the context of entropy and the factors that play into that entropy is a leap forward in the thought process that should go into creating an architecture. Through the examples provided in this paper current practitioners should be able to apply those lessoned learned to their own development of architectures to decrease the impact of entropy.

The authors intend on continuing research on this top to expand our understanding of the impact and cost of system architecture entropy. Other research questions to be investigated include: 1) what role does architecture management play in minimizing the impact of architecture entropy, and 2) once the impact and cost of system architecture is better understood, how can systems engineers cope with architecture entropy.

References

Charette, Robert. 2008. What's Wrong with Weapons Acquisitions?, IEEE SPECTRUM 45.11: 32-39.

Christensen, Clayton M. 1997. The Innovator's Dilemma : When New Technologies Cause Great Firms to Fail. Boston, MA, Harvard Business School Press.

Clayton, D. H., G. M. Jebsen, et al., 2002. The All Electric Warship From Vision to Total Ship Integration. Naval Surface Warfare Center Dahlgren Division.

Dalton, J. H., J. M. Boorda, et al..1994. Forward ... from the sea, Washington, D.C., Department of the Navy.

Encyclopedia Britannica. “Landing ship, tank”, Encyclopedia Britannica Online. http://www.britannica.com/EBchecked/topic/329366/landing-ship-tank

Fishbane, Paul, Stephen Gasiorowicz and Stephen Thornton. 1996. *Physics for Scientists and Engineers,* Prentice Hall, New Jersey, USA.

Grady, J. O. 1994. System integration, Boca Raton, CRC Press.

Gorbea, Carlos, Ernts Fricke, and Udo Lindermann. 2008. The Design of Future Cars in a New Age of Architectural Competition, Proceedings of the ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. New York, USA.

Henderson, Rebecca and Kim Clark. 1990. Architectural innovation: the reconfiguration of existing product technologies and the failure of established firms - Technology, Organizations, and Innovation. Administrative Science Quarterly.

Hsu, Wei-Lieh, Chang-Lung Tsai and Tsung-Lun Chen. 2007. Traffic Detection at Nighttime

Using Entropy Measurement. Proceedings of the Third International Conference on International Information Hiding and Multimedia Signal Processing.

Isaac, David and Gail McConaughy. 1994. The Role of Architecture and evolutionary Development in Accommodating Change. Proceedings of the Fourth Annual International Symposium of the National Council on Systems Engineering.

Lacerte, Yves. 2000. Legacy System Evolution to an Enterprise-Wide Architecture Framework. Proceedings of the Tenth Annual International Symposium of the International Council on Systems Engineering.

Martin, James. 2002. On the Transition of a Legacy System to a New System (Upgraded or Otherwise). Proceedings of the Twelfth Annual International Symposium of the International Council on Systems Engineering.

McCullough, Vice Admiral Barry, and Allison Stiller. 2008. Navy Force Structure and Shipbuilding, Subcommittee on Seapower and Expeditionary Forces, Committee on Armed Services: U.S. House of Representatives, July 31, 2008 of U.S. House of Representatives.

Microsoft Press Release, Microsoft Launches Windows Vista and Microsoft Office 2007 to Consumers Worldwide, January 29, 2007. http://www.eweek.com/c/a/Windows/Pushing-Forward/

Microsoft Press Release, Windows History: Windows Desktop Products History, March 7, 2006. http://www.microsoft.com/windows/WinHistoryDesktop.mspx

Morison, Elting E. 1966. Gunfire at Sea: A Case Study of Innovation, Men, Machines, and Modern Times. Cambridge, MA, The MIT Press, pps 17-44.

Naval Postgraduate School. 2008. "Total Ship Systems Engineering." Naval Postgraduate School, http://www.nps.edu/Academics/GSEAS/TSSE/index.html.

O'Keefe, S., F. B. Kelso, et al. 1993. From the sea : preparing the naval service for the 21st century, Washington, D.C., Department of the Navy.

Percivall, George. 1994. System Architecture for Evolutionary System Development. Proceedings of the Fourth Annual International Symposium of the National Council on Systems Engineering.

Shannon, C. E. 1948. A Mathematical Theory of Communication, Bell System Technical Journal.

Schulman, Andrew, et al. 1990. DOS Undocumented, Addison-Wesley Longman Publishing Co., Inc.,

Schulman, Andrew, and David Maxey. 1992. Undocumented Windows; a Programmers Guide to Reserved Microsoft Windows API Functions. The Andrew Sc. Addison-Wesley Longman Publishing Co., Inc.

Schulman, Andrew. 1994. Undocumented DOS: A Programmer's Guide to Reserved Ms-Dos Functions and Data Structures. Andrew Schulman Programming Series, 2nd ed. Reading, MA, Addison-Wesley Pub Co.

Song, Hualing, Jinli Wang and Cornelis Reiman. 2004. iThe Multi-dimension Metric for the Assessment of Organizational Complexity – Case Study of Shendong Coal Co. Ltd. 14th

Annual International Symposium Proceedings.

Walsh, Steven T., Bruce A. Kirchhoff and Scott Newbert. 2002. Differentiating Market Strategies for Disruptive Technologies, IEEE Transactions on Engineering Management 49.4, pps 341-51.

Wells, Brian. 2006. Applying Systems Engineering to Naval Shipbuilding. Proceedings of the 16th Annual International Council of Systems Engineering.

Wilson, Mark. 1996. Navy Legacy Systems Reengineering. Proceedings of the Sixth Annual International Symposium of the International Council on Systems Engineering.

Biography

	Rob Cloutier is an Associate Professor of systems engineering in the School of Systems and Enterprises at Stevens Institute of Technology. He has over 20 years experience in systems engineering & architecting, software engineering, and project management in both commercial and defense industries. Industry roles included lead avionics engineer, chief enterprise architect, lead software engineer, and system architect on a number of efforts and proposals. His research interests include model based systems engineering and systems architecting using UML/SysML, reference architectures, systems engineering patterns, and architecture management. Rob holds a BS from the US Naval Academy, an MBA from Eastern College, and his Ph.D. in Systems Engineering from Stevens Institute of Technology.
	Mary Alice Bone has worked as a Systems Engineer for GE Transportation, BAE Systems, and Rockwell Collins. She holds a B.S. in Aerospace Engineering from the University of Missouri – Rolla and a M. Eng. in Systems Engineering from Iowa State University. She has been involved with internal company system process groups which has ignited the interest in producing system requirement processes that resolve the struggle between system process and project process while maintaining systems engineering integrity. She is currently pursuing a PhD in Systems Engineering from Stevens Institute of Technology.
	Dinesh Verma received the Ph.D. and the M.S. in Industrial and Systems Engineering from Virginia Tech. He is currently serving as Dean of the School of Systems and Enterprises and Professor in Systems Engineering at Stevens Institute of Technology. During his six years at Stevens he as successfully proposed research and academic programs exceeding $50m in value. Verma concurrently serves as Scientific Advisor to the Director of the Embedded Systems Institute in Eindhoven, Holland. Prior to this role, he served as Technical Director at Lockheed Martin Undersea Systems, in Manassas, Virginia, in the area of adapted systems and supportability engineering processes, methods and tools for complex system development and integration. Verma continues to serve numerous companies in a consulting capacity, to include Eastman Kodak, Lockheed Martin Corporation, L3 Communications, United Defense, Raytheon, IBM Corporation, Sun Microsystems, SAIC, VOLVO Car Corporation (Sweden), NOKIA (Finland), RAMSE (Finland), TU Delft (Holland), Johnson Controls, Ericsson-SAAB Avionics (Sweden), Varian Medical Systems (Finland), and Motorola. Dr. Verma has authored over 100 technical papers, book reviews, technical monographs, and co-authored two textbooks. He is a Fellow of the International Council on Systems Engineering (INCOSE), a senior member of SOLE, and was elected to Sigma Xi, the honorary research society of America. He serves as on the Core Curriculum Committee of the Delft University's Space Systems Engineering Program (Holland) and as an advisor to the Systems Engineering Center of Expertise at the Buskerud University College (Norway). He was honored with an Honorary Doctorate Degree (*Honoris Causa*) in Technology and Design from Växjö University (Sweden) in January 2007.
	Kim Sommer is an electronics engineer at Naval Surface Warfare Center – Crane Division working in the Advanced RF Branch. He holds a B.S. in Applied Physics (EE) from the University of Louisville Speed Scientific School and is working towards a Masters in Systems Engineering from Stevens Institute of Technology. He has worked on surface EW systems and development of EW training software, microwave tubes and vacuum electronics, incentive procurement from limited vendors, and distributed fleet training systems. He is now working on issues with testing and procedures for antennas used in ground EW systems.