The challenges of emerging software eco-systems

Summary of a keynote speech

Neil G. Siegel
Northrop Grumman Corporation
Carson, California, USA
neil.siegel@ngc.com

***Abstract* — New opportunities for software-intensive system configurations are arriving on the market; these include cyber-physical, cyber-social, and cloud structures. Because of the convenience and cost-savings opportunities they offer, these capabilities and configurations will be adopted, most likely quickly and at large scale. Some of these configurations, however, have the potential to create (or already are creating) significant unintended problems and vulnerabilities. The author identifies a range of such unintended problems and vulnerabilities, and indicates the types of research and new insights that will be needed so as to allow society to obtain the benefits promised by these emerging opportunities.**

***Categories and Subject Descriptors* – D2: Software Engineering**

***General Terms* – Design, Management, Measurement, Performance, Reliability, Verification**

***Keywords* – Cloud, cyber-physical systems, cyber-social systems, unplanned dynamic behavior, reliability of software, reliability of systems, latent defects, software-intensive systems, software ecosystems, system safety.**

I. Introduction

A significant opportunity for software practitioners is upon us: the advent of new and highly-capable software-intensive systems configurations. These, when joined with all of the necessary tools, processes, training, reference designs / design patterns, and so forth, can be considered to form a series of new software "eco-systems". By extending the range of societal problems that can be addressed by software, these new configurations provide us the opportunity to increase further our profession's value to society. By opening up new levels of equipment-use and power-use efficiency through sharing of resources (e.g., the "cloud"), we can save money and avoid pollution[1]; we also can enable our users to make use of their data at an increasing range of locations and situations. By reaching beyond purely computational and data configuration to the world of physical devices (the "internet of things"[1]), we can create new functionality and convenience, while at the same time increasing the efficiency of society in many ways – reducing traffic congestion, improving health-care outcomes, and so forth.

Because of the convenience and cost-savings that these opportunities offer, these capabilities and configurations will be adopted, most likely quickly and at large scale. But experience with systems engineering has taught us that such complex structures often exhibit *emergent behavior*, that is, behavior that arises from the interaction of the components and features that is more complex than the original components and their apparent interactions[2]. Some of this emergent behavior is desirable; in fact, obtaining this "1+1=3" is often the reason for interconnecting formerly-separate elements together into a larger system. But many systems also exhibit unintended emergent behavior, some of which has significant adverse effects; I term this "unplanned dynamic behavior[3]". Within the context of the software-intensive system configurations that are considered herein, some of this unplanned dynamic behavior has the potential to create (or is already creating) problems and vulnerabilities for society. Herein, I identify a set of such unintended problems and vulnerabilities, and indicate the types of research and new insights that will be needed so as to allow society to obtain the benefits promised by these emerging opportunities.

I assert that we in the software and systems business have a responsibility to take the initiative in response to the challenge posed by such adverse emergent behavior; our users do not have the requisite skills to address these issues themselves, and whether we intended for our products to be used in this fashion or not – it appears, for example, that the original developers of the internet did ***not*** envision it being used to control and synchronize vast numbers of safety-critical physical devices, nor did they consider the possibility that some of the users of the internet would use the technology in destructive or criminal fashion[4] – we are

[1] (Koomey 2008) reports that data centers alone use 2% of the total U.S. electric consumption, and therefore are a significant source of pollution, comparable to the commercial airline industry. Personal computers, mobile computing and communications devices, etc., all add to the energy and pollution foot-print of our software-enabled devices.

the only ones who can find a safe and effective way for society to realize the benefits that are anticipated from our systems. Furthermore, we are the ones who can select the most appropriate and most cost-effective solutions; if we let politicians or regulators do it, experience suggests that the outcome may be far less desirable than if we were to do it ourselves.

I will discuss some examples of these new software eco-systems, the unintended problems that have emerged, and for each, try to identify the types of improvements required.

II. Discussion of specific emerging software eco-systems

Cloud. Let me start with the cloud. Herein, I use the term "cloud" to mean the aggregation of computing and storage into a small number of centralized sites, with a dynamic allocation of computing resources taking place in real-time, in response to dynamic offered load, access controls, and service policies. The goal is to achieve cost-efficiencies and power-efficiencies through higher average utilization levels, while simultaneously (a) decreasing total computing resource required (not all peaks of demand will occur at the same time), (b) enabling more access methodologies (and thereby, user access from more locations, and with higher levels of reliability), (c) allowing much faster provisioning times to new demand (in the limit, essentially provisioning a user or enterprise with additional computing and storage capability in real-time, and also allowing them to release capacity almost instantly when it is no longer required), and (d) doing all of the above while simultaneously improving overall access control, privacy, and security. Whether or not you view this as "back to the past" or something radically new (or a bit of both), the benefits to users and enterprises promised along these lines are so significant that adoption is occurring at a great rate, whether or not the capability is really at an appropriate level of maturity.

Some example indicators that the capability may ***not***, in fact, be yet at an appropriate level of maturity include:

- Frequent difficulty in achieving consistently high average utilization levels; the products appear to hit "turbulence" well below the anticipated levels[2].
- High levels of configuration errors, and difficulty in diagnosing and correcting them.
- Intense confusion and fear among our users and potential users about the privacy and security aspects of co-mingling their data with that of others, and ignorance of what is happening inside of the cloud.

This collection of indicators manifests itself in a pronounced tendency for enterprises to decide that they need to establish their own cloud(s), rather than sharing a resource with others. In some real sense, the very existence of the term "private cloud[5]" is an oxymoron, and a powerful indicator of the lack of confidence held by our users.

What improvements might correct these issues? Here are some ideas:

- *Fine-grained access control, yet with a feasible administrative burden*. Many of our administrative tools either are not fine-grained enough to accommodate the needed situational flexibility, or provide that flexibility only at the cost of being complex, labor-intensive, and error-prone to administer. In the limit, one can envision a need for control down to the level of N users x M objects in the addressable computing space. But N and (especially) M are likely to be large enough (and dynamic enough) that establishing, enforcing, and maintaining such fine-grained access control is not feasible. The answer probably lies in the declaration of some dynamic set of access-control policies that can result in a situation where the number of such policies is far smaller than N x M, but at the same time their use does not constrain what the users and their enterprises consider to be the right balance of access and protection, is simple enough that it can be administered by the available personnel (a very material concern), and there exists some method to "prove" / convince the users and enterprise owners of its efficacy. The last item is particularly noteworthy, as it is a key enabler to address the lack of confidence held by many potential users of the cloud (especially the general public).
- *High-reliability operations (expressed in terms of availability of service), while never losing or corrupting data*. In fact, methods exist to improve availability, prevent data loss, and prevent data corruption, but they all tend to involve replication of some sort, which at the limit runs the risk of diluting the economies-of-scale with the enterprise operators consider the principal point of using the cloud in the first place. Better ways of achieving these goals without diluting the economic advantage of the cloud are required, and more tailorability of the trade-offs to the desires of each enterprise operator are probably required.
- *Key management in configurations with unprecedented numbers of encryption keys*. I believe that – for all of the obvious reasons – we are heading towards a world where all data will be encrypted,

[2] For example, benchmarks performed by our company report that overhead in processing / resource allocation results in many commodity cloud computing elements seldom being able to exceed 40% average utilization.

both in motion and at rest[6]. Yet today's encryption management approaches do not appear to scale to the challenge. We need to know who is actually using a device and requesting to gain access to data – today's authentication methods are clearly inadequate. We need to create, distribute, backup, recover, and manage huge numbers of encryption keys, without imposing unreasonable burdens for memorizing password, PINs, and the like. We need to encrypt data at rest and in motion, without such encryption becoming a source of data loss (through loss of key, etc.) or access delays. We need all of the above to be usable by the administrators and end-users, without requiring unrealistic levels of training and/or expertise. And, again, we need methods to convince the users and enterprise owners of its efficacy.

- *Detecting when people have (perhaps inadvertently) created security holes*. Today's software-intensive devices and systems have large numbers of settable parameters and configuration options. It is far too often the case that some combinations of settings create security vulnerabilities in those systems. It is not reasonable to expect our users (and often, even our system administrators) to be able to acquire the expertise to avoid such situations; we have to create the tools and mechanisms that can detect the creation of such vulnerabilities automatically, and then correct them through a mechanism that can scale to the necessary size and capacity.

Big data. I almost hesitate to talk about "big data", because so much has already been written about this topic. But there are a few issues that I believe need additional attention:

- One interesting aspect of emerging big data constructs is that as developers, we will know less than we have historically known about the range of data to be processed, its formats, and especially its provenance. How we ought to be going about testing such systems – when the range of data to be encountered, their timing, their capacity are all unbounded – is not clear. What is clear is that delivering systems that create wrong outputs, or fail in some significant manner, is not a good option. There is a rich opportunity here for tools, processes, methods of assessing completeness of testing (and hence, residual risk) that address this issue. For example, how will we get our V&V / testing processes to scale with the larger range of potential input data? Simply doing more testing using conventional approaches is likely to be too expensive and take too long, and will still leave us with significant latent defects. And in any case, our current V&V / testing approaches pretty much assume that we know the range and types of input data *a priori*; it seems at least possible that such an assumption will not be correct in many emerging circumstances. So what ought we to do? More formal design methods, so as to assure predicted responses to unexpected inputs and data? Techniques – such as the "crowd-sourcing" of test processes – that might scale better? Note that one might argue that the beta-release process widely used in the software business is in fact a form of crowd-sourcing, but it is not clear that this creates use and testing over a range of conditions, rather than having a lot of people repeatedly find the same defect over and over. How do we stimulate off-nominal data inputs and usage in crowd-sourced testing?
- A special case of the above is that increasingly-large portions of the data that our systems need to process comes from unvetted sources (e.g., "the net"). How can we provide some sort of "quality / trustworthiness" assessment of the data we are processing, and of the "answers" that we generate? This is a big issue in computer science generally; the internet can provide its users with lots of "answers", but provides almost no assessment of which of those answers are correct and credible.

Cyber-physical systems. In my own opinion, the most exciting and most important emerging software eco-system is the cyber-physical one, that "internet of things" described by Ashton[1]. I say this because it appears that there are very material conveniences and efficiencies for society through the interconnection of formerly-stand-alone physical systems to our information networks and data bases, and society intends to obtain those benefits. At the same time, there are gigantic potential liabilities – consider, just as one example, the potential for hackers disabling the brakes on modern automobiles[7]. I believe that society rightly assumes that since our community has special knowledge about how these systems work (and how they could go wrong), we have a special responsibility to take a pro-active role in driving these systems to safe, reliable, and transparent operation; authors such as Wetmore[8] seem to agree. This creates a significant *social* role for our profession, in addition to our traditional technical role.

Here are a few of the unsolved problems that I see in the cyber-physical opportunity space:

- *Authentication*. If we are allowing ever-increasing remote and over-the-net mechanisms to send commands and receive status from physical devices, it becomes increasingly important that we actually know who is issuing those commands, and who is asking to receive that status and other data. We all know how weak the standard user-name / password combination is – once captured or compromised by any of a large number of mechanisms, there is essentially no back-stop to prevent "borrowing" an identity – but our profession has not come forward

with feasible, scalable, affordable alternatives. Biometrics have appeared at times to be a candidate for this role, but various obstacles (real and perceived) have apparently prevented large-scale adoption. We need better authentication methods that can be used by ordinary people, meet emerging societal standards for privacy, and do not fail catastrophically as a result of a single compromise.

- *Built-in protection against unreasonable use*. In many circumstances, there are behaviors that are so far off-nominal that local blocks and checks probably should be provided. Examples include disabling the brakes in a passenger car, ordering mechanical devices beyond their designed usage range, and so forth. Yet, in general, our profession does not provide local controls to prevent such usage. If a generator, centrifuge, or other mechanical device has inherent physical limitations (e.g., acceleration, speed, etc.), shouldn't we be providing inherent protection against command actions that would put the device beyond these physical limitations?

A special case of cyber-physical systems are those that I term "cyber-social". An example would by the interconnection of electronic health-care records to health-care sensors, data banks with medical test results, billing systems, clinical-management systems, research systems, and so forth. There are a number of potential "1+1=3" effects that arise from such interconnections, and the stakeholders reach far outside of the development and procuring communities; every citizen, for example, has a stake in the emerging network of health-care data processing. Social factors are likely to dominate the discussions around such systems, rather than the technical factors. There is a body of literature that assesses these phenomena, and proposes various approaches, but this is far from a normal consideration within our field, and seldom does it truly drive technical decision-making. Yet the existing literature suggests that that is exactly what it ought to do. For example, Metlay and Sarewitz[9] address the problem of siting a facility for high-level nuclear-waste disposal, and show how attention paid (or not paid) to the social aspect of the problem can result in a remarkable difference in the efficacy of outcomes, with the social considerations dominating the technical considerations in those circumstances where the social considerations are ignored. Their case study compares U.S. and Swedish approaches, and demonstrates that in the U.S. case, the lack of a suitable approach to the social side of the problem has thus far prevented the technical merits from driving decision-making[10]. Similarly, the public outcry a few years ago over Intel's announced intention to place unchangeable serial numbers in each of its microprocessor chips[11] – an important enabler for many technical improvements – was evidently pursued without appropriate attention to the social aspects, and eventually (apparently) withdrawn. We are all familiar with the problem of rapidly-escalating health-care costs in the U.S.; many efforts have posited that the use of information technology is a key element of any corrective strategy, but studies[12] indicate that all such efforts to-date – despite impressive levels of spending – have failed. There are indications[13] that the social aspects of this problem at present dominate, and until addressed, will mask the potential effectiveness of technical solutions, just as they have for the nuclear-waste disposal problem.

Some[14] have used the term "wicked problem" to describe these systems-engineering situations where social aspects are on a par with the complexity of the technical aspects. Nixon[15] cites the following characteristics of these sorts of problems:

- Are ill-defined
- Involve many stakeholders with strong and opposing views
- Have conditions that change midstream
- Are misunderstood until a solution is in hand
- After solution, morph into new wicked problems

He further describes the following as key challenges with such "wicked problems":

- The problem might be too fuzzy / indistinct to permit the definition of complete requirements
- The problem might change mid-stream
- The stakeholders might not all agree
- We might not know all of the stakeholders

We must train our personnel to be adept at recognizing these situations, and then provide them with tools and training to cope with the mixture of social and technical elements. For example, Nixon summarizes his approach to addressing such problems as indicated in Figure 1[16]:

Figure 1. Nixon's approach to solving wicked problems.

However one views the merits of such an approach, this is not what most current computer science practitioners have

been taught. Note, however, that there is a conceptual similarity to Boehm's spiral method[17].

Furthermore, we must allow schedule time and funding within our projects plans to address the social issues, just as we allocate schedule time and funding to the technical issues. We must have profession-wide focus on this matter, which probably ought to result in profession-level guidance, just as we do for technical topics. Boehm[18], for example, says that ". . . the fact that the capability requirements for these products are emergent rather prespecifiable has become the primary challenge". He also states that "It is clear that requirements emergence is incompatible with past process practices . . . "[19], and "A system will be successful if and only if it makes winners of its success-critical stakeholders. Losers in win-lose situations will generally either retaliate or refuse to participate, usually leading to a lose-lose outcome"[20]. I would suggest that the implication is that it will become necessary for an increasing number of software and system development practitioners also to acquire domain knowledge of their user's problems and operating environment. The tendency has instead been, through mechanisms such as integrated product teams, to assume that such knowledge can be brought to the team by domain specialists who will work together with the software and system practitioners. I believe that approach has proven not to be sufficiently robust, and leads to too many poor designs; I believe that a better approach is to get our software and systems experts also to acquire domain knowledge of the problem they are trying to solve. Since Boehm also states[21] that today's users often "exhibit the 'I'll know it when I see it' syndrome" and "their priorities change with time", it is easy to see why integrating domain knowledge of the problem into the brains of our software and system designers could lead to more-consistently better results. Fortunately, Boehm has also given us a framework in which to actualize this approach: his 2005 version of the spiral model[22]. This version introduces stakeholder-centric objectives, and supports the approach of having each incremental spiral circle back through some sort of assessment with the user / stakeholders. This will allow a gradual convergence over the course of such a spiral development between our software-intensive systems and the emerging expectations of our increasingly non-technical stakeholders.

III. Reliability and Safety – a special concern

Modern commercial passenger jets now achieve more than 1,000,000 hours mean-time-between-hull-loss-failures[23]. While those of us who spend a lot of time flying from place to place are grateful, this forms a striking contrast to the software-intensive products that surround us in our everyday lives. Most software-intensive products are not even assessed or rated for mean-time-between-critical-failure (or any related sort of measure). No consumer is given such information as a guide for product selection. And in fact, few engineers and designers have access to such information, either, unless they did the testing and characterization themselves[24].

There is probably a good reason for this dearth of information – the numbers are awful. I have sponsored such measurements multiple times over the course of my career; here are a few indicative results:

- Actual measured mean-time-between-failures for single desktop computer / application-suite combinations are often in the range of 10 to 100 hours[25]. This is a far cry from the 1,000,000+ hours cited above for a modern commercial airplane, or even the thousand-hour-class mean-time-between-failure of a modern passenger car.
- Enterprises that collect service-level information (and many do) almost never incorporate end-point-device failures into their metrics. If a desktop computer (or similar device) crashes, and time has to be taken to diagnose, re-boot, and perhaps re-enter some data, information about that event seldom gets incorporated into the service-level calculations about availability, reliability, or actual cost impact that the enterprise uses to make decisions. We are very good about collecting service-level information, but often this information is misleading, collecting information about the central elements of our enterprise, but not about the far-more-numerous user devices. In essence, we use our service-level metrics to justify spending less on reliability because we treat the cost of an entire class of failures as zero. Worse, through delegation of administrative and corrective actions ("self-service"), we transfer tasks properly associated with administration and maintenance to our users, and then assume that their time to perform these tasks is without cost.

Yet we increasingly use our computers, applications, devices, and networks for activities that have real societal impact, and increasingly, actual safety impact – think of that micro-controller on the CAN bus in your car[26] that determines how (and if) your brakes operate when you step on the pedal. I assert that we in the software-intensive-system business need to systematize our practice, especially to reduce variation in our practice, and hence, variation in our performance[27], and then improve the mean. This probably means standards, model implementations and reference designs, an educational component, and so forth. Before all of that, it will also require some real research into how to achieve these higher levels of reliability. How might we do it?

- *Many fewer source lines of code to implement a function.* Alan Kay, for example, has long advocated a specific method to accomplish this[28], one based on creating domain-specific language extensions,

particular ways of handling system time and control, and other methods.

- *Formal methods*. Software development today largely follows an empirical method. Various attempts at introducing formal-based methods have been made (e.g., Dijkstra's 1998 open letter to the computer science community[29]), but these have never really caught on. Their widespread adoption would probably require higher levels of training and expertise on the part of the average practitioner (which is possibly why they have not caught on), but hold the potential for software-intensive systems with many fewer latent defects.
- *Model-based development*. A particular type of potentially formal development that has caught on in some circles is model-based development[30]. The goal is to describe the required functionality and desired structure of a system in fewer primitive statements (and those machine-readable, and ideally, in some sort of graphical form), both in order to support automatic analysis of the design, and (in some instances) automatic generation of the executable software, with the aspiration that this will result in fewer latent defects (and hence, higher productivity and better reliability). Results are somewhat mixed; there are many complaints that the resulting executable code is often mediocre, and other limitations (e.g., lack of versioning, etc.) have been cited[31].
- *Design-for-reliability methodologies*. Other domains have explicit methodologies for achieving reliability and accuracy. This is not a strong tendency within the software business; the focus of optimization is much more often on cost of development, time of development, and richness of user functionality. I suggest that the low mean-time-between-failure for software systems that I cited above is the direct result of this lack of focus. There are some striking examples of excellence; the design of the basic internet services, incorporating as it does replication of key data, link redundancy, and the ability to operate over any available path, is an example of a software-intensive system that is well-designed to achieve reliability. We should start a conversation about why many more of our software-intensive systems are not as well designed.
- *Far better validation and verification / testing*. Software products retain an enormous number of latent defects when they are transitioned into use; the actual number varies considerably with domain, but is almost always significant; based on the data I have seen, one latent defect per 1,000 source lines of code seems like a reasonable rule-of-thumb[32]. Since today's software-intensive systems have millions of lines of software code (more than 10,000,000 for a modern fighter jet[33] – this could be a *real* crash; more than 200,000,000 for Windows and Office together), that results in a lot of latent defects. It is not clear whether this latent defect rate reflects overly-complex design, poorly-designed testing, an inadequate amount of testing, or some other combination of causes; in any case, having tens of thousands of latent defects in software products performing important missions for society cannot be a desirable situation. Probably much more automation in creating a range of test cases and test data, and in performing the actual tests, will be required. One latent defect per 1,000 lines of software code probably sounded adequate when our products had a few tens of thousands of lines of code in them, but as our software products have scaled to 1,000x or more that sort of size, better methods of testing are required. Nor do we seem to have good methods for determining when we have (or have not) done enough testing.
- *We need to emphasize these points equal to – or even more than – time-to-market considerations*. My observation is that our business is very strongly connected to a time-to-market methodology. The U.S. President's Information Technology Advisory Committee report concluded that "The IT industry spends the bulk of its resources, both financial and human, on rapidly bringing products to market"[34]. Functionality, and especially quality, take second-place to getting to the market at the planned time. I suggest that this will lead to disaster, especially in the cyber-physical domain; latent defects in such systems can cause physical damage, not just require a software re-boot.

IV.A CANDIDATE SUCCESS STRATEGY

Having introduced unplanned dynamic behavior as a significant root cause of some of our industry's problems, I would like also to introduce one candidate method for creating designs that are less likely to suffer from such problems.

In (Siegel 2011)[3], I provide an analysis of one important mechanism through which unplanned dynamic behavior gets into our systems, and introduce a design-based technique to prevent that particular mechanism from causing such problems. The technique involves explicit design processes that address the design of how one controls dynamic behavior in your system, and is summarized in the Figure 2 (drawn from reference [3]):

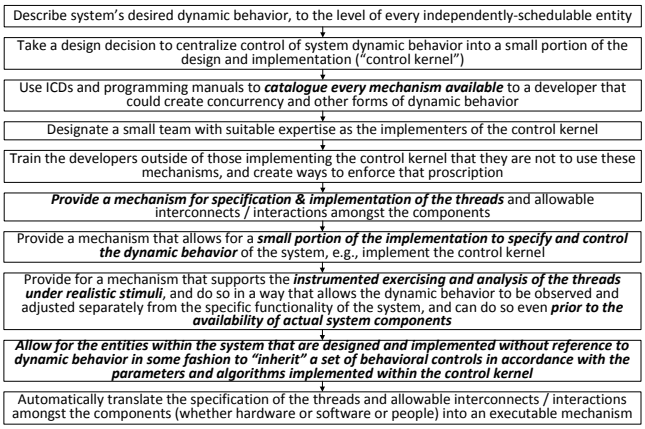

Figure 2. Methodology for controlling dynamic behavior.

In my role as chief engineer and program manager on a number of large system development efforts, I had the opportunity to employ this technique. I have also had the opportunity to compare outcomes of programs that used the technique with those that did not use the technique (normalizing for other factors). A material improvement in system quality resulted from the use of the technique, as indicated in Figure 3 (also drawn from reference [3]):

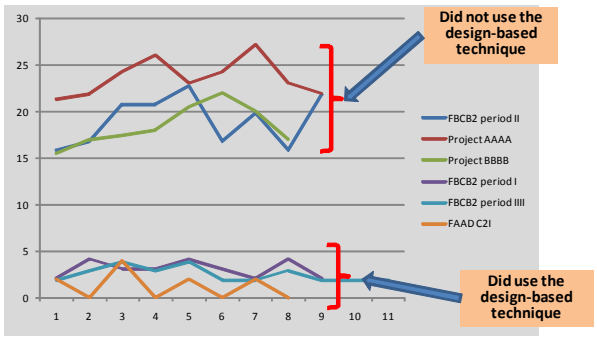

Figure 3. Comparison of a system quality measurement, with and without the use of the indicated design-based technique.

The vertical axis in the above figure represents a measure of defect-density. As can be seen, the programs that did use the design-based technique had materially-better outcomes (in terms of this measure) than those that did not.

This might indicate that the "apparently-intractable" issues that recur continuously in our industry – such as low reliability, high latent-defect rates, and the other measures cited above – in fact can be addressed through suitable engineering techniques, and we can therefore in fact deliver better systems to our customers and to society.

V. Conclusions

New opportunities for software-intensive system configurations – I described those related to the cloud, to big-data, and to cyber-physical and cyber-social opportunities – are arriving on the market. Because of the convenience and cost-savings opportunities they offer, these capabilities and configurations will be adopted, most likely quickly and at large scale. Some of these configurations, however, have the potential to create (or already are creating) significant unintended problems and vulnerabilities.

I indicated examples of such problem areas, the topics for research, and in some cases, even potential avenues of correction.

I believe that we are approaching a point-of-inflection; computer privacy issues and hacking in general have generated considerable interest among the general public, and among politicians. If, on top of this concern, there is a major undesirable event of the sort indicated in this paper (e.g., physical damage through a cyber-physical system, continued large-scale privacy-loss events that are accelerated or attributed to the cloud, and so forth), our industry risks control and intervention from the political world that might actually be hugely counterproductive.

We need both to take steps to correct these vulnerabilities, and also to find ways to establish the credibility of our approaches to the general public. I assert that these goals are in fact more important to our industry and our society than the next "killer app".

We need to talk about how we get these topics on the table in our development methodologies and projects, and how we create the necessary theoretical, technological, and educational underpinnings.

VI. About the Author

Neil Siegel, Ph.D., sector vice-president & chief engineer at Northrop Grumman, has been responsible for the creation of many first-of-their-kind, large-scale, high-reliability data-processing systems, in addition to many complex cyber-physical systems. These include the U.S. military's Blue-Force Tracker; the U.S.'s first large-scale unmanned air vehicle system; and the world's first system that can protect against attack by rockets, artillery, and mortars. He is a member of the National Academy of Engineering, a Fellow of the IEEE, and the recipient of the IEEE's Simon Ramo Medal for systems engineering, among many other awards and honors.

VII. References

[1] Ashton, Kevin; *That 'Internet of Things' Thing*, RFID Journal, 2009.

[2] Rechtin, Eberhardt; *Systems Architecting*, Prentice Hall, 1991.

[3] Siegel, Neil; *Organizing Complex Projects Around Critical Skills, and the Mitigation of Risks Arising from System Dynamic Behavior*, University Of Southern California, 2011.

[4] Kleinrock, Leonard; personal communication, 2009.

[5] See, for example, National Institute of Standards and Technology, The NIST Definition of Cloud Computing, Special Publication 800-145 or http://www.gartner.com/it-glossary/private-cloud-computing/.

[6] National Institute of Standards and Technology, Guide to Storage Encryption Technologies for End User Devices, Special Publication 800-111.

[7] Koscher, Czeskis, et al; Experimental Security Analysis of a Modern Automobile; 2010 IEEE Symposium on Security and Privacy.

[8] Wetmore, Jameson W.; The Value of the Social Sciences for Maximizing the Public Benefits of Engineering, The Bridge, National Academy of Engineering, Fall 2012, makes a similar point.

[9] Metlay, Daniel, and Sarewitz, Daniel; *Decision Strategies for Addressing Complex, "Messy" Problems*, The Bridge, National Academy of Engineering, Fall 2012

[10] Also see *Blue Ribbon Commission on America's Nuclear Future, a report to the Secretary of Energy*, U.S. Department of Energy, 2012.

[11] See, for example, *Processor Serial Number Questions & Answers*, Intel Corporation, 2003 (available at http://www.intel.com/support/processors/pentiumiii/sb/cs-007579.htm), and McCarthy, Jack; *Intel to Phase Out Controversial Processor Serial Numbers*; IDG News Service, 2000 (available at http://www.networkworld.com/news/2000/0428intelnumber.html).

[12] See, for example, Bernstam and Johnson, *Why Health Care IT Doesn't Work*, The Bridge, National Academy of Engineering, winter 2009.

[13] For example, see Gold, Larry and Siegel, Neil; *Proteomics-Based Individualized Health-Care*, to appear.

[14] For example, Nixon, Steve; *Wicked Problems – the Challenge of Our Age*, TTI Conference, 2011, and the Metlay and Sarewitz article cited above.

[15] Nixon, opere citato, page 4.

[16] Nixon, opere citato, page 20.

[17] Decribed in Boehm, Barry, *A Spiral Model of Software Development and Enhancement*, Computer Magazine, 1988; and Brooks, Fredrick P., *The Design of Design*, Addison Wesley, 2010.

[18] Boehm, Barry W., *Some Future Trends and Implications for Systems and Software Engineering Processes*, Wiley Periodicals, Systems Engineering, volume 9, number 1, 2006.

[19] Boehm, opere citato, page 4.

[20] Boehm, Barry and Ross, Rony, "Theory-W Software Project Management: Principles and Examples," IEEE Transactions on Software Engineering, July 1989.

[21] Boehm, Barry W., *Some Future Trends and Implications for Systems and Software Engineering Processes*, Wiley Periodicals, Systems Engineering, volume 9, number 1, 2006, page 4.

[22] Boehm, Barry, W., and Lane, Jo Ann, *21st Century Processes for Acquiring 21st Century Software-Intensive Systems of Systems*, Cross Talk – The Journal of Defense Software Engineering, 2006.

[23] Boeing Aircraft Company; Statistical Summary of Commercial Jet Airplane Accidents, Worldwide Operations, 1959 – 2011; 2011.

[24] Perrow, Charles; *The Next Catastrophe – Reducing our Vulnerabilities to Natural, Industrial, and Terrorist Disasters*, Princeton University Press, 2007

[25] Siegel, 2011, opere citato.

[26] CAN (Controller Area Network) is a standard for a single-wire control and data bus in cars, promoted by the Society for Automotive Engineers. See http://standards.sae.org/j2411_200002/ for implementation details and definitions.

[27] Wheeler, Donald "Understanding Variation", SPC Press, 2000, discusses the essential role that reducing variation has in achieving improved performance.

[28] Kay, Alan, et al, *Steps Towards the Reinvention of Programming*, VPRI Research Note, RN-2006-002, 2006.

[29] Dijkstra, Edsger, *On the Cruelty of Really Teaching Computer Science*, an open letter, 1988. Available at http://www.cs.utexas.edu/~EWD/ewd10xx/EWD1036.PDF.

[30] Described in many places, including Schaetz et al, *Model-Based Development*, Technische Universität München, TUM-I0204, 2002

[31] Hann, Johan den; *8 Reasons why Model-Driven Development is Dangerous*, The Enterprise Architect, 2009. Available at http://www.theenterprisearchitect.eu/archive/2009/06/25/8-reasons-why-model-driven-development-is-dangerous.

[32] There are many sources for these sort of data. See, for example, Thangarajan (http://www.tataelxsi.com/whitepapers/SoftReliabilityPredictModel.pdf?pdf_id=SoftReliabilityPredictModel.pdf) or Jones, Subramanyam, Bonsignour (The Economics of Software Quality), 2011.

[33] See, for example, Charette, Robert N., *F-35 Program Continues to Struggle with Software*, IEEE Spectrum, 2012; *Fun Facts About the F-35*, The Center for Public Integrity; Reed, John, F-35's biggest problems: software and bad relationships, Foreign Policy, 2012 (available at http://killerapps.foreignpolicy.com/posts/2012/09/17/f_35s_biggest_problems_software_and_bad_relationships).

[34] President's Information Technology Advisory Committee, Report to the President, 1999.

ICSSP'13, May 18–19, 2013, San Francisco, USA