SYSTEMS ENGINEERING NEWSLETTER

brought to you by

Project Performance International (PPI)

SyEN 56 – August 21, 2017

SyEN is an independent free newsletter containing informative reading for the technical project professional, with scores of news and other items summarizing developments in the field, including related industry, month by month. This newsletter and a newsletter archive are also available at www.ppi-int.com.

Systems engineering can be thought of as the problem-independent, solution/technology-independent, life-cycle-oriented principles and methods, based on systems thinking, for defining, conducting and controlling the engineering content of a technical project. The approach aims to maximize the benefit delivered to the enterprise, as influenced by the needs and values of applicable stakeholders.

If you are presently receiving this newsletter from an associate, you may wish to receive the newsletter directly in future by signing up for this free service of PPI, using the form at www.ppi-int.com. If you do not wish to receive future Systems Engineering Newsletters, please unsubscribe by clicking on the link at the bottom of this email.

We hope that you find this newsletter to be informative and useful. Please tell us what you think. Email us at syen@ppi-int.info.

The views expressed in externally authored articles are those of the author(s), and not necessarily those of PPI or its professional staff.

IN THIS EDITION

Quotations to Open On

Read More…

Feature Article

- More Effective Use of Prototypes in Systems Engineering, *by Dennis M. Buede*

Read More…

Article

- Advancing Software from a Craft to a Profession, *by Capers Jones*
- Integrating Program Management and Systems Engineering, *by Ralph Young*

Read More…

Systems Engineering News

- PivotPoint's SafetyML Safeguards Safety-Critical Systems
- LinkedIn Open-sources Tools for Managing Website Outages

Read More…

Featured Organizations

- American Society for Engineering Education
- INCOSE

Read More…

Conferences and Meetings

Read More…

Some Systems Engineering-Relevant Websites

Read More…

Systems Engineering Publications

- Systems Approaches to Management
- 'Don't Panic! The Absolute Beginner's Guide to Model-Based Systems Engineering'
- Facilitating the Project Lifecycle
- “Discovering the Strategy for Whole System Modeling”
- “Engineering the Virtual or Collaborative SoS”

Read More…

Systems Engineering Tools News

- CRADLE
- AgileCraft 10X

Read More…

Education and Academia

- Online Program Wins Engineering Education Award

Read More…

Standards and Guides

- Developments at the International Standards Organization
- IEEE Announces IT Healthcare Standard

Read More…

Some Definitions to Close On

- System(s) of Systems, System(s) of Systems Engineering (SoSE)

Read More…

PPI and CTI News

Read More…

Upcoming PPI and CTI Participation in Professional Conferences

Read More…

PPI and CTI Events

Read More…

QUOTATIONS TO OPEN ON

"Science is about knowing; engineering is about doing."

Henry Petroski

"Systems thinking is a discipline for seeing wholes, it is a framework for seeing interrelationships rather than things."

Peter Senge

"There is almost nothing that *must* be done within systems engineering. There are mainly things that we choose to do, or choose not to do, for exactly the same reason – achieving the best result in the circumstances, on the balance of probabilities"

Robert Halligan

FEATURE ARTICLE

More Effective Use of Prototypes in Systems Engineering

by

Dennis M. Buede

Innovative Decisions, Inc.

Abstract

The practice of prototyping in the world of today's designers and systems/software engineers is explored for the purpose of offering to the systems engineering community that they can be more effective than they are today. There is a long history of using prototypes in engineering with great effect. Manifestations of prototypes have evolved as computing power has increased and become nearly universal. Discussions of physical, software, and modeling and simulation prototyping in this paper describe what is possible and what benefits will result. Finally, several quick case study sketches demonstrate the value of designing the systems engineering process while designing the product system with an emphasis on what should be prototyped followed by how/when the prototypes should be accomplished.

Web site: www.innovativedecisions.com

Email: mailto:dbuede@innovativedecisions.com

Copyright © 2017 by Dennis M. Buede. All rights reserved.

Introduction

The V model (Rook, 1986; Forsberg and Mooz, 1992, Buede and Miller, 2016) of the systems engineering process has been described in a great many systems engineering references. While there are aspects of system engineering that the V model does not do justice, nonetheless it does provide useful guidance to many aspects of systems engineering. The major purposes of systems engineering on the left-hand side of the V include:

1. Define the life cycle system-level requirements and associated external interfaces requirements – accomplished by defining use cases for the system's major interactions within the meta-system of the system throughout the system's life cycle, and by defining the system's boundary.
2. Define derived requirements and associated internal interface requirements for subsystems, components, and configuration items throughout the system's lifecycle – accomplished by creating functional and physical architectures.
3. Define any associated manufacturing, training, deployment, refinement and retirement requirements associated with the system throughout its life cycle.

4. Define test and integration requirements associated with all (life cycle) system-level and derived requirements – accomplished by creating test and integration use cases.

5. Define the test and integration system that will accomplish the test and integration requirements.

Assertions:

A. The creation of a system architecture is very helpful in completing #2 such that the derived requirements can be shown to meet the system-level requirements in #1.

B. The creation of a life cycle model is very helpful in completing #3.

C. Success with # 4 and 5 depend upon success in #2 and 3.

Thesis of this article:

Prototyping on current systems development activities is insufficient to achieve the major purposes of SE on the left-hand side of the V. This lack of prototyping for elements of the system architecture, life cycle model, and test/integration system are major contributors to the cost and schedule overruns and reductions in capability associated with most systems development projects. This paper discusses the current state-of-the-art in prototyping relevant to systems engineering.

Prototyping can take many *forms*: physical device construction, software mock-ups of user interfaces, and modeling and simulation (M&S).

Similarly, there are many *value-creating purposes* for creating a prototype:

- Enable additional understanding of the problem by creating an object that stimulates discussion and reduces misunderstandings and ambiguities among users, developers and other stakeholders;
- Determine the feasibility of an innovative or risky solution approach quickly and cheaply;
- Clarify the scope and requirements for an approach by getting users involved in a meaningful way;
- Refine the detailed design of a component before other components to reduce risk; and
- Enable early testing to reduce risk.

The following sections discuss common approaches to prototyping using physical, software, and M&S approaches.

Physical Prototyping

Isa (2014) and Ulrich and Eppinger (2012) define four categories of models for physical prototyping: soft, hard, presentation, and prototype.

Soft modeling is a quick and dirty approach to create a physical, non-functional representation that is useful for interactions with users or customers for gauging the size, proportion, and shape of a system or component. The model may be made of any material that facilitates "quick and dirty" effort, such as sculpting foam.

Hard modeling requires more effort than soft because the purposes are to explore the look and feel of the exterior of the component or system. Materials such as plastic, metal, and wood are commonly used. While a hard model is not functional, it may expose designers and users/customers to potential enhancements and user interaction devices such as movable buttons or sliders.

A *presentation model* is a complete model of a component or system with fully detailed physical (non-functional) representations of configuration items (for a component) or subsystems/components for a system. Such a presentation model is built from a Computer-aided Design (CAD) drawing; an example would be the wooden representations that aircraft manufacturers build to explore the conflicts among interfaces and components for space. When conflicts among interfaces and other interfaces or components were found, "chainsaw engineering" physically cut components and interfaces out and moved them to less dense parts of the wooden model. More recently aircraft manufacturers have been building these models using software tools. Virtual reality systems are enabling engineers to walk within the software version of the model.

While it is unusual to create a subclass that is the same as the class, Isa's fourth subclass of prototype is "*prototype*". This prototype subclass is a functioning product or very high quality model of the product that will be used for testing and evaluation by representative users or customers. This class also includes focused prototypes in which a relative final version of the product focused on a subset product or system's dimensions of interest.

Another type of physical prototype is the very old concept of a "*breadboard*", which is a working model of at least partial functionality that demonstrates that proves the design concept can work even though aesthetics and scale are not representative.

Software Prototyping

Software prototypes became very popular in the 1980s and have evolved dramatically since. Andriole (1993) and Borysowich provide some detail of current software prototypes: concept, feasibility, horizontal, vertical, and functional.

A *concept* prototype using software is a high-level vision of the "look and feel" of the user interface, overall scope associated with the system, and any other factors that deal with vision for the system. This concept prototype(s) can be created with software or on paper or both. This concept prototype is naturally undertaken early in the requirements phase, typically during the development of the system's use cases

and system level requirements. A prototype at this early stage can help achieve early consensus and defuse long debates.

The *feasibility* prototype is very similar to the physical presentation model. The purpose of the feasibility prototype is to demonstrate that the pieces fit together, verifying that the critical components integrate in a productive manner. This is an example of the benefit of substantial software reuse because a feasibility prototype with significant software reuse can come together much more rapidly and efficiently than when the entire feasibility prototype has to be created from scratch. A feasibility prototype for the software of the control system is an excellent idea during the development of the system-level requirements.

The *horizontal* prototype extends the user interface prototype of the concept prototype to address the entire user interface through the details of all use cases. This horizontal prototype is used during extensive interviews with representative users and customers to work out detailed requirements at the subsystem and component levels. No functional capabilities of the system need be part of this horizontal prototype.

The creation of reduced version of the core functions of the system are the *vertical* prototype. In addition to this core functionality, the vertical prototype provides access to data for the users and system/software engineers to explore. Since data access is provided, a detailed data model must have been developed. Access to core functionality and data means that vertical prototypes are only commonly available late in the requirements phases to finalize requirements. Without a vertical prototype the designers are betting that all core functions (including key control systems) will work well enough to pass verification and validation testing cost effectively, which is a large risk.

The *functional* prototype also addresses functionality but is more limited in addressing sequential work flows associated with information presentation. The functional prototype is typically an extension of the horizontal prototype and can be created in an evolving manner. The functional prototype has traditionally been created using storyboards but now many of the software products for prototyping make functional storyboards much simpler. There may be 20 to 400 frames in the functional storyboard. Like the horizontal prototype there is no functionality behind the storyboard frames.

Exemplary software prototyping tools for horizontal, vertical, and functional prototypes are Atomic, Axure RP, Balsamiq Mockups, Fluid, HotGloo, Indigo Studio, InVision, JustInMind, Marvel, Mockplus, Origami, POP, Protoshare, UXPin, Webflow, and Webydo.

Other authors have extended this categorization of four software prototypes to nearly 20. Some of the more interesting extensions are:

- Diagonal – horizontal prototype for most of the system but vertical for selected areas.
- Evolutionary – a functional prototype that has evolved over time to contain some functionality and become usable for a subset of the system's capability.

- Minimum viable product – a combination of the functional and vertical prototypes that addresses sufficiently critical parts of the system's capabilities that users are willing to use it for real work.

M&S Prototyping

Systems engineers have used software models and simulations (M&S) since the 1960s to investigate design concepts for performance issues related to timing, range, throughput, probability of success, and others. These M&S tools abstract the hardware and software that make up the system to a set of equations that can either be solved analytically (model) or via simulation using random numbers to capture uncertainty or discrete steps through time when uncertainty is not present (simulation). M&S tools have also addressed issues of control systems, time synchronization, reliability- availability – maintainability, weight distribution, and cost. During the Apollo program in the 1960s and early 1970s that took U.S. astronauts to the moon, every aspect of the hardware, software, trajectory from the Earth to the Moon was examined via M&S tools. In the past 30 years, it is a rarity to find software products examined via M&S tools prior to release. For systems comprised of hardware and software the use of M&S tools remains present but is not as extensive as in the 1960s and 1970s.

On the positive side, there are now model integration tools (called model-based engineering) that enable engineers of all disciplines to work with the tools they know and trust but enable the engineering team to integrate these models into a workflow that creates an M&S product for most or all of the product system or development system. Now engineers can model and simulate the effect of a few changes in one area to the performance of the entire system.

Conclusions

All forms of prototyping can be thought of inexpensive, early verification and validation. What a bargain and yet this bargain is being ignored by many in our industry.

There are some motivational historical examples of successful systems engineering due to prototype use:

- The Wright brothers used a kite and three gliders before they built their first airplane, which they flew during their multi-day test session in 1903. They did not derive the requirements for the engine until they had determined they had sufficient lift and control for the aircraft (Buede, 2002).

- The Sidewinder missile was designed, tested, and built in the 1950s by a small team of engineers led by Dr. William McLean at China Lake. Their design process was characterized as "build a little, test a little" and included constant prototyping. The result was an air-to-air missile architecture that has survived to this day. Early in the design process McLean knew they would have to manufacture the missile in two pieces because it was too long to be stored in one piece on Navy ships. They could not finalize any requirements until they determined that the design was controllable with canards (control surfaces near the front of the missile). Nearly every major component of the Sidewinder was prototyped before the requirements were defined (Powell and Buede, 2006).

- The Apollo program to take U.S. astronauts to the moon in 1969 and beyond was built around a series of prototypical hardware and software that was refined from Apollo 2 through Apollo 10 in preparation for the landing on Apollo 11 (Orloff and Harland, 2006).

A useful lesson learned from the Ariane 5 launch system built by the European Space Agency involves testing software that is being reused (Buede and Miller, 2016). Software was being reused from Ariane 4 for the Inertial Reference System. A major purpose in creating Ariane 5 was to facilitate launching rockets into new orbital trajectories around the earth. Simple M&S or physical prototypes could have been created to test software issues but were not. These software issues resulted in the loss of the first Ariane 5 and its payload.

Summary and Key Takeaway

In summary, prototyping enables systems engineers to:

- Achieve faster and more valuable feedback from the problem owners about design concepts. This feedback enables systems and software engineers not only to find better concepts faster but to also obtain a deeper understanding the problems that the product system is to address.
- Achieve a faster and more accurate understanding of the problem space so that exploration of the trade space of solutions is more productive.
- Explore a far larger portion of the trade space in a given period of time, which will likely yield a better solution.
- Communicate key design concepts in some depth to the problem owners and users in terms that these problem owners and users will understand clearly.
- Conduct testing on the left-hand side of V thus finding problems earlier when it is cheaper to fix them.

Finally, the *key takeaway* from this article is that systems engineers **who are working on a product system and are NOT also designing the systems engineering systems will find that FAILURE is the most likely outcome**. Many of the success stories in systems engineering used the *incremental development with single delivery* V, as seen below. This approach to systems engineering begins the system-level definition, but then delays decomposition of parts of the system while one or two parts are explored in detail with derived requirements and handed off to the design engineers. The figure below shows two additional decompositions for derived requirements, based upon the derived requirements of previously decomposed components before the final system is integrated and tested. This approach does usually take longer but typically has a lower failure rate.

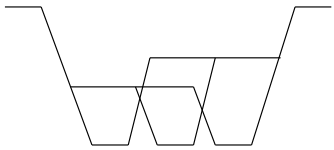

Figure 1: Incremental Development: Single Delivery

References

Andriole, S.J. *Rapid Application Prototyping: The Storyboard Approach to User Requirements Analysis*. Wiley, 1993.

Borysowich, C. *Prototyping: Types of Prototypes*. http://it.toolbox.com/blogs/enterprise-solutions/prototyping-types-of-prototypes-14927.

Buede, D.M. "The Concepts of Systems Engineering as Practiced by the Wright Brothers." *2002 INCOSE Symposium Proceedings*, Las Vegas, 2002.

Buede, D.M. and Miller W.D. *The Engineering Design of Systems: Models and Methods.* (3rd edition) Wiley, 2016.

Forsberg, K., and Mooz, H. "The relationship of systems engineering to the project cycle". *Engineering Management Journal Vol.* 4, No. 3, 1992, pp. 36-43.

Isa, Siti Salwa. "Classifying Physical Models and Prototypes in the Design Process: A Study on the Economical and Usability Impact of Adopting Models and Prototypes in the Design Process." Dubrovnik, Croatia: Proceedings of the 13th International Design Conference (DESIGN 2014), Culley, S.J., Lindemann, U., McAloone, T.C., Weber, C., and Marjanovic, D. (eds.), May 2014.

Orloff, R.W. and Harland, D.M. (2006). *Apollo: The Definitive Sourcebook*. Springer, 2006.

Powell, R.A. and Buede, D.M. (2006). Innovative Systems Engineering: A Creative System Development Approach, 2006 INCOSE Symposium, Las Vegas, NV.

Rook, Paul. "Controlling Software Projects" in *Software Engineering Journal* (Vol 1, No 1, pp. 7-10, January 1986, ISSN 0268-6961.

Ulrich, K.T. and Eppinger, S.D. *Product and Design Development*, fifth edition. McGraw Hill Companies, 2012. ISBN 978-007-108695.

ARTICLE

Advancing Software from a Craft to a Profession

by

Capers Jones

Vice President and CTO

Namcook Analytics LLC

Version 2.0 – March 29, 2017

Abstract

As of 2017 software is a major industry, but also a troubling industry. Software projects have many failures and even more cost and schedule overruns. Poor software quality remains an endemic problem, in part because software quality measures are generally incompetent.

Software education is deficient in teaching effective metrics and effective professional skills. In surveys of corporate CEO's and C-level executives by the author, software engineering is regarded as the least professional and most troublesome form of engineering in major technology companies due to frequent failures and numerous cost and schedule overruns.

Unlike medical practice and other forms of engineering, the software industry has been running blind for over 60 years with little or no accurate quantitative data on productivity or quality and no empirical data that proves the value of software tools, methodologies, or programming languages. You can hardly be deemed a profession if you can't measure economic productivity or quality, or measure the effectiveness of the tools, languages, and methodologies in common use.

Web site: www.Namcook.com

Email: Capers.Jones3@gmail.com

Copyright © 2017 by Capers Jones. All Rights Reserved.

Keywords

Profession, craft, software engineering, software education, software failures, software benchmarks, accurate software metrics, inaccurate software metrics, medical diagnoses, medical measures, medical metrics, software diagnoses, software risk analysis.

Introduction

Before dealing with the professional status or craft status of software engineering, the author wants to recommend an important book that provides key insights into how a minor craft evolved into a true and major profession.

The book is The Social Transformation of American Medicine, Basic Books, 1982 by Paul Starr. This book won a Pulitzer Prize for non-fiction in 1984 and also a Bancroft Prize in 1984.

In the 19th century medicine was a craft with sparse and questionable academic training, no political power, and no licensing or board certifications. There were few medical schools and they had two-year curricula.

Medical schools did not require either college degrees or even high school graduation to attend. Essentially the ability to pay the fees of these for-profit medical schools was the only criterion for admission. Many young physicians did not attend medical schools but worked as apprentices for older practicing physicians.

While in medical school student physicians never even entered hospitals because hospitals had their own closed groups of physicians and did not allow external physician at all. Even regular physicians could not visit their patients once they were admitted to hospitals. In England and some other countries women were barred from becoming physicians.

There were no state medical licenses and no board certifications. There were no regulations against medical malpractice, and not even any way to monitor or report bad medical decisions. Harmful drugs such as laudanum could be freely prescribed in spite of serious side effects such as opium addiction. In fact laudanum, liquid opium, was even available over the counter for calming down noisy children!

Paul Starr's book on medical transformation starts around 1760 and continues through 1980. At the start of the book medical doctors were an offshoot of barbers and had little professional training other than apprenticeship with older doctors, mainly because of the lack of American medical schools.

The Wikipedia list of U.S. medical school start dates shows The University of Pennsylvania medical school starting in 1765, Columbia University in 1767, Harvard in 1782, Dartmouth in 1797, University of Maryland in 1807, Yale in 1810, and Brown University in 1811. It was not until after the Civil War that American medical schools began to increase in numbers and to top 100 academic medical institutions.

Paul Starr's book shows the gradual evolution of medical training and medical professionalism, including the interesting history of the American Medical Association (AMA).

In a nutshell, the AMA grew from a small inconsequential organization of a few hundred members to become a major professional association with over 230,000 members circa 2017. One of the methods used for this rapid growth was for the AMA to reach out to various state and local medical societies and offer reciprocal memberships.

This technique raised AMA membership from a few hundred physicians to over 80,000 in a few decades and began to give the AMA enough mass to lobby state governments for requiring medical licensing. The

larger membership also gave the AMA enough power to influence medical school curricula, and to even force the closure of some of the least competent medical schools.

It would also benefit software to have reciprocal memberships among the various software associations such as the Project Management Institute (PMI), the IEEE Software Engineering Society, the Society of Information Management (SIM), the Association of Computing Machinery (ACM), the International Function Point Users Group (IFPUG), and quite a few others.

It took almost 100 years for medicine to evolve from a craft into a leading profession. With Paul Starr's book as a guide, software engineering could probably evolve into a true profession in another 25 years or less, starting today.

What are the Indicators of a Profession?

The Professional Standards Council says, in paraphrase, that a profession is "*A group of individuals who receive formal education in the knowledge of a field based on valid research, and who adhere to a code of ethical behavior.*" Professionals should be able to "*apply their training and knowledge with high levels of competence and integrity*". All of us in software would do well to remember the line in the medical Hippocratic Oath, "*First do no harm.*" We should also remember Tom DeMarco's famous line, "*you can't manage what you can't measure.*"

It is interesting that the Wikipedia list of professionals has over 100 job titles, but software engineering is not listed. Examples of some 15 of these professions include:

1. Accountants
2. Air line pilots
3. Anesthesiologists
4. Attorneys
5. Biologists
6. Chemists
7. Clergymen
8. Dentists
9. Engineers
10. Educators – college professors
11. Educators – school teachers
12. Nurses

13. Pharmacists

14. Physicians

15. Physicists

All of these 15 have formal education based on large bodies of knowledge, and all require some kind of certification or licenses before professional work can be performed. Most require continuing education after starting work to keep current on the latest advances in the fields.

Software in 2017 has dozens of kinds of certification for specific tools and for general categories such as certified tester or certified quality assurance. However, there is little empirical data that shows certification actually improves job performance, although education certainly should be helpful.

Why Software Engineering is not yet a Profession?

Those of us who work in software can be proud of the many accomplishments created by software engineers including but not limited to medical diagnostic software, banking and financial software, operating systems and embedded software, military and defense software, and hundreds of other important applications.

However, software also has many canceled projects and a very large number of cost and schedule overruns. A survey by the author of CEO's and other C-level executives found that software was regarded by corporate CEO's and other executives as the least professional and least competent of any form of engineering. If our own CEO's don't regard us in software as professional, it is hard to claim that we are.

Lyman Hamilton, a former Chairman of ITT, noted in a public speech that it took about three years of on the job training before graduate software engineers could be trusted with serious projects, as opposed to about one year for other kinds of engineers such as mechanical, electrical, or telecommunications engineers.

For software to take the step from being a craft to becoming a true profession we need to solve a number of critical endemic problems that have plagued software since the beginning. The fifteen steps to achieve software engineering professional status:

1. Reduce the many software failures.

2. Reduce the many software cost and schedule overruns.

3. Improve poor software quality after deployment.

4. Improve poor software productivity and shorten schedules.

5. Improve poor software security and reduce cyber-attacks.

6. Stop using inaccurate and invalid metrics that distort reality.

7. Adopt accurate metrics and measurement practices.
8. Reduce inaccurate and optimistic estimates prior to starting.
9. Eliminate inaccurate status tracking during development.
10. Reduce high maintenance costs after deployment.
11. Reduce or eliminate frequent litigation from unhappy clients.
12. Improve undergraduate and graduate software education.
13. Improve post-graduate and on-the-job software education.
14. Introduce software licensing and board certification.
15. Move from custom manual software to standard reusable components.

Let us evaluate these 15 professional needs in a bit more detail.

Topic 1: Reduce the many software failures

Software projects are very risky. In fact, large software projects seem to have the highest failure rates of any industry in human history. As software pioneer Dr. Gerald Weinberg observed years ago, "*…if buildings were built the way software is built, a woodpecker could destroy civilization...*"

From the author's examination of about 26,000 successful and unsuccessful software projects, failures are directly proportional to application size measured in function points as shown in Table 1:

Table 1: Software Failure % by Size

Size in IFPUG 4.3 FP	Failure Average
1	1.00%
10	1.86%
100	3.21%
1,000	10.14%
10,000	**31.29%**
100,000	**47.57%**
1,000,000	**82.29%**
Average	**25.34%**

Below 1000 function points software projects are only slightly risky. Above 10,000 function points they have perhaps the highest risk of any manufactured product in human history. This does not speak well for the professionalism of software engineering.

Poor quality control is the main cause of software failure. Poor quality control, combined with unplanned requirements creep, can double planned schedules and these delays cause return on investment (ROI) to switch from positive to negative so projects are terminated without being completed.

Topic 2: Reduce Cost and Schedule Overruns

Of the software projects above 1000 function points that are not canceled, about 70% run late and 60% exceed planned budgets. This is a key reason that CEO's and other C-level executives don't trust their software engineering teams.

(Software cost and schedule estimates are usually excessively optimistic, due in part to the poor accuracy of informal manual estimates rather than to using accurate parametric estimates.)

Poor quality control that stretches out test cycles combined with unplanned requirements creep in excess of 1% per calendar month are the two main causes of cost and schedule overruns.

Incidentally to calculate average U.S. software schedules a simple rule of thumb is to raise application size in function points to the 0.4 power. The result will be the schedule in calendar months. For example 1,000 function points raised to the 0.4 power shows a schedule of 15.84 calendar months.

Projects with good technology stacks and expert teams could use a power exponent of 0.38. Projects with poor technology stacks and novice teams could use a power exponent of 0.42. Large defense projects which create about three times the volume of paper documents compared to civilian projects might use the 0.44 power, since paperwork is the #1 cost driver for defense software. Defense projects also have large volumes of non-functional requirements. One of the advantages of function point metrics is their ability to quantify software results better than the older lines of code metric.

Topic 3: Improve software quality after deployment

A majority of large software systems have marginal to poor quality after deployment. A majority of companies and government agencies do not measure software quality, or know the effectiveness of various kinds of defect removal activities such as inspections, static analysis, testing, etc.

A majority of universities have no empirical data on software quality control and hence are not able to teach state of the art methods to undergrads. Lack of quality measures and poor understanding of software quality control are two key reasons why software engineering is not yet a true profession but still a craft.

In order to improve software quality two key factors need to improve: 1) Defect potentials need to be reduced from today's average of about 4.25 per function point down below 2.50 per function point; 2) Defect removal efficiency (DRE) needs to increase from today's average of about 92.5% to over 99.00%.

(These quality results are based on observations of about 26,000 software projects combined with 15 lawsuits involving poor quality where the author has been an expert witness.)

Testing alone is not sufficient to achieve high quality because most form of testing are only about 35% efficient or find one bug out of three. Static analysis is about 55% efficient and formal inspections are about 85% efficient. A synergistic combination of inspections, static analysis, and formal testing can top 99.50% defect removal efficiency (DRE). Even better, high DRE levels > 99.00% also have shorter schedules and lower costs than low DRE < 85.00%.

Lowering defect potentials requires advanced methods such as reuse of certified components that approach zero defects, requirements models, and automated proofs of correctness. These can lower defect potentials from over 4.25 bugs per function point down below 2.50 bugs per function point.

Topic 4: Improve today's low software development productivity and long schedules

Custom software designs and manual coding are intrinsically slow, expensive, and error prone no matter what methodologies are used and what programming languages are used. Today in 2017 the U.S. average for software development productivity is only about 8.00 function points per staff month or 16.5 work hours per function point and software reuse is below 15% on average.

What the software industry needs to achieve are consistent productivity rates > 25.00 function points per staff month or 5.28 work hours per function point. The only known way of achieving such high productivity rates is to shift from custom designs and manual coding to ever-larger percentages of certified reusable components.

Table 2 shows the impact of software reuse on a project of a nominal 1000 function points in size coded in Java by average teams:

Table 4: Impact of Software Reuse on Productivity and Quality

Reuse %	Wk Hrs per FP	FP per Month	Defect Potential per FP	Defect Removal Percent	Delivered Defects per FP
95%	2.07	63.63	1.25	99.75%	0.003
85%	2.70	48.94	1.68	98.25%	0.029
75%	3.51	37.65	2.10	96.78%	0.068
65%	4.56	28.96	2.53	95.33%	0.118
55%	5.93	22.28	2.95	93.90%	0.180
45%	7.70	17.14	3.38	92.49%	0.253
35%	10.01	13.18	3.80	91.10%	0.338
25%	13.02	10.14	4.23	89.74%	0.434
15%	16.92	7.80	4.65	88.39%	0.540
5%	22.00	6.00	5.08	87.06%	0.656
0%	33.00	4.00	5.50	85.76%	0.783

Methodologies such as agile are marginally better than older methodologies such as waterfall but are barely 15% better in terms of measured productivity and measured schedules. Modern languages such as Ruby and Python and Swift are better than older languages such as FORTRAN and COBOL but improve productivity and quality by less than 12%.

The only truly effective way of improving software productivity and quality at the same time is to eliminate custom designs and manual development and shift to construction from standard and certified reusable components.

Topic 5: Improve poor software security and reduce cyber-attacks

Software security and cyber-attacks are modern problems that are becoming more serious every day. The criminals who are attempting cyber-attacks are no longer individual hackers, but now members of sophisticated organized crime groups and even worse, operating under the control of hostile military cyber-warfare units in countries such as North Korea and Iran.

Cyber security is a highly complex topic and every major company and government organization that builds or runs software needs both internal security teams and also access to top security consultants, plus links to government cyber-security specialists at the FBI, Homeland Security, and also State and Municipal government security groups.

Topic 6: Stop Using Inaccurate and Invalid Metrics that Distort Reality

The software industry has been running blind for over 60 years with bad metrics and bad measurement practices. The author believes that the software industry has the worst metrics and measurement practices of any industry in human history.

It would be surprising if more than 5% of the readers of this article know their own company's actual productivity and quality levels at all using any metric. Less than 1.00% would know their productivity and quality results within 10% precision using function point metrics. Bad metrics and bad measurement practices are endemic problems for software and a professional embarrassment. Software can never be a true profession without knowing how to measure results with high accuracy.

This is too big a topic for a short article. Suffice it to say that the "lines of code" (LOC) metric penalizes high-level languages and makes requirements and design invisible. A metric that makes assembly language look better than Ruby or Swift in a professional embarrassment.

The "cost per defect" metric penalizes quality and is cheapest for the buggiest software. Applications with thousands of bugs have lower costs per defect than applications with only a few bugs.

Fixed costs are the reason for both problems. Every other industry except software knows that *if a development process has a high percentage of fixed costs and there is a decline in the number of units produced, the cost per unit will go up*.

If you use LOC as a unit and switch to a high-level language you reduce the number of units produced. But the work on requirements and design act like fixed costs and drive up cost per LOC.

If you use bugs as a unit and have effective quality control with static analysis and formal testing the fixed costs of writing and running test cases drives up cost per defect. These are not just casual statements but have been be proven mathematically.

Other hazardous but common software metrics include story points (undefined and highly erratic), use-case points (undefined and highly erratic), and technical debt, which no two companies seem to measure the same way in 2017. The author regards poor metrics as a sign of professional malpractice, and software can't be a true profession until we adopt accurate metrics such as function points.

Topic 7: Adopt Accurate Metrics and Effective Measurement Practices

As of 2017 the best metrics for understanding software productivity are work hours per function point. This metric can be applied to individual activities as well as total projects; i.e. requirements, design, coding, and everything else can be measured to show the fine details of software development.

The best metrics for understanding software quality are defect potentials measured using function points combined with defect removal efficiency (DRE). DRE is the percentage of bugs found and eliminated prior to release of software to customers. The current U.S. average for DRE is only about 92.50% but best in class projects top 99.50%.

It is a matter of historical interest that all three of these effective metrics were developed within IBM during the years 1970 to 1975. IBM deserves thanks from the software industry for spending money on building a metrics family that allows high accuracy in both estimating projects before they start and measuring projects after they are delivered.

As to poor measurement practices, self-reported data tends to "leak" and omit over 50% of true software costs. Among the author's clients the accuracy of self-reported development data is only about 37%.

The major omissions from self-reported data include unpaid overtime, project management costs, project office costs, and the work of part-time specialists such as quality assurance and technical writers.

Because self-reported data leaks the best way of collecting accurate historical data is to use one of the

commercial benchmark organizations of which there are about 35 in the United States and quite a few in Europe. These benchmark groups will assist clients in collecting data for 100% of work performed instead of just a small fraction of work such as "design, code, and unit test" (DCUT) which is only about 30% of total effort.

The major cost drivers for software include the costs of finding and fixing bugs and the costs producing paper documents. These costs should definitely be part of historical benchmark data, as well as coding costs and testing costs.

The International Function Point Users Group (IFPUG) introduced a new metric for non-functional requirements in 2012. This is the SNAP metric (Software Non-Functional Assessment Process). SNAP is too new to have much empirical data so it will not be discussed at this time. Non-functional requirements are things like government mandates such as Sarbanes-Oxley governance of financial applications. These add substantial costs but are not user requirements.

Topic 8: Improve inaccurate and optimistic estimates before starting projects

As a disclosure to readers, the author is the developer of 10 software parametric estimation tools. Six of these were proprietary and built for specific companies such as IBM, ITT, and AT&T. Four of these have been marketed commercially.

Below about 250 function points in size manual estimates by experts and parametric estimates by tools such as COCOMO, ExcelerPlan, KnowledgePlan, SEER, SLIM, Software Risk Master (SRM), SPQR/20, and TruePrice are roughly equivalent.

However, as application size grows larger, manual estimates have a tendency to become progressively optimistic while parametric estimates tend to hold their accuracy up to 100,000 function points or larger.

If you build software applications below 250 function points most forms of estimation are acceptable. But if you build large systems above 10,000 function points in size you should stop using manual estimates and switch over to one or more of the commercial parametric estimation tools. (Some leading companies use two or three parametric estimation tools for the same application and also have formal software estimation teams.)

Topic 9: Eliminate Inaccurate Status Tracking

From working as an expert witness in a number of lawsuits where large software projects were cancelled or did not operate correctly when deployed, six major problems occur repeatedly: 1) Accurate estimates are not produced or are overruled; 2) Accurate estimates are not supported by defensible benchmarks. 3) Requirements changes are not handled effectively; 4) Quality control is deficient; 5) Progress tracking fails to alert higher management to the seriousness of the issues; 6) Contracts themselves omit important topics such as change control and quality, or include hazardous terms.

Depositions and discovery in these lawsuits found that software engineers and project managers were aware of the problems that later caused termination, but that monthly status reports did not alert higher management or clients to the existence of the problems. In fact, in some cases project management deliberately concealed the problems, perhaps in the hope that they could be solved before anyone found out about them.

The bottom line is that status tracking of large software applications needs significant improvement compared to 2017 averages. Automated project tracking tools and the use of project offices for large

applications can improve this problem. But until project tracking reveals problems instead of concealing them, software cannot be deemed a true profession.

Topic 10: Reduce high maintenance costs after deployment

As many readers know, for large software applications more than 50% of total costs of ownership (TCO) occur after release rather than during development. While some of the costs are due to normal evolution and growth in functionality, the majority of the post-release costs are due to bug repairs for latent bugs in the application when it was delivered, plus the costs of "bad fixes."

A "bad fix" is a new bug accidentally included in bug repairs. The U.S. average for bad-fix injection in 2017 is that about 7% of bug repairs add new bugs. But with modules having high cyclomatic complexity of over 25, bad-fix injections can top 30%. Bad fixes are an endemic problem, but completely treatable by using static analysis tools on all bug repairs before integrating them. Unprofessional quality measures and unprofessional quality control are the cause of bad-fix injections.

Another huge post-release cost is that of dealing with "error-prone" modules (EPM). IBM discovered that bugs are not randomly distributed but clump in a small number of very buggy modules. Other companies such as AT&T and Raytheon also noted EPM, which occur in most large systems.

In general, less than 3% of the modules in an application will contain over 50% of all bugs. Thus, about half of post-release bug repair costs are for EPM, and they can and should have been eliminated prior to release.

A synergistic combination of defect prevention, pre-test defect removal such as static analysis and inspections, combined with formal testing and formal test case design such a using cause-effect graphs can completely eliminate EPM before software is released. Running cyclomatic complexity tools against all modules is important too. It is due in part to the very poor quality measurement practices of the software industry and poor quality control that most EPM are not discovered until too late. In fact, quality measures are so bad that many companies have EPM but don't even know about them!

To achieve professional status software should have less than 0.1% bad-fix injections and have zero error-prone modules (EPM) since both of these endemic problems have technical solutions available.

Topic 11: Reduce or eliminate litigation from unhappy clients

Among the author's clients who commissioned us to study outsourced projects, we found that about 70% of the projects were reasonably successful and both the vendor and the client were generally satisfied. But for 30% of the projects there was considerable dissatisfaction by the clients for poor quality and schedule delays. For about 5% of the projects litigation was either about to happen or was in progress when we examined the projects. For quite a few projects the author was an expert witness in actual litigation.

Dissatisfaction and litigation occurs for other kinds of contracts such as home construction and home repairs, but software has more litigation than it should. Breach of contract litigation is the topic of this section, but software also has a lot of patent litigation and even some criminal litigation for things such as poor governance under Sarbanes-Oxley rules or causing death or injury as in the case of defective medical devices or automotive brake systems.

The root causes of breach of contract litigation seem to be four critical topics, all of which can be eliminated: 1) optimistic cost and schedule estimates before starting; 2) very poor status tracking during development; 3) inadequate quality control so that the application does not work well after deployment; 4) sloppy change control which tends to introduce "bad fixes" into software applications.

If software engineering were a true profession, all four of these common conditions that lead to litigation would be eliminated.

Topic 12: Improve undergraduate and graduate software education

About every two years the author does a study of software learning channels since there are quite a few of them. The study looks at effectiveness, convenience, costs, and other parameters. It is interesting that academic education does not rank very high among 17 learning channels as shown by table 3:

Table 3: Ranking of Software Learning Channels as of Spring 2016

Average Score	Form of Education	Cost Ranking	Efficiency Ranking	Effectiveness Ranking	Currency Ranking
3.00	Web browsing	1	1	9	1
3.25	Webinars/e-learning	3	2	6	2
3.50	Electronic books	4	3	3	4
5.25	In-house training	9	4	1	7
6.00	Self-study from CD/DVD	4	3	7	10
7.25	Vendor training	13	6	5	5
7.25	Commercial training	14	5	4	6
7.50	Wiki sites	2	9	16	3
8.25	Live conferences	12	8	8	5
9.00	Simulation web sites	8	7	13	8
10.25	Self-study from books	5	13	12	11
10.25	Journals	7	11	14	9
10.75	On-the-job training	11	10	10	12
11.75	Mentoring	10	12	11	14
12.00	Books	6	14	15	13
12.25	Undergraduate training	15	15	3	16
12.25	Graduate training	16	16	2	15

As already noted graduate software engineers seem to need more on-the-job training than other forms of engineering before doing major projects.

The fact that most graduate software engineers don't know the hazards of common metrics such as cost per defect and lines of code is a sign that academic training needs to be upgraded. Most graduates also

don't know the measured defect removal efficiency (DRE) values for things like formal inspections, static analysis, and testing.

Compare these gaps in software education with what physicians know about the effectiveness of various medicines and therapy procedures. Software is running blind because poor metrics and poor measurement practices conceal progress and make it difficult to judge the effectiveness of tools, methods, languages, and other software performance factors.

Topic 13: Improve post-graduate and on-the-job software education

A study by the author some years ago on the impact of post-employment professional education by companies found an interesting result. Companies that provide 10 days of professional internal training per year have higher software productivity rates and better quality than companies of the same size and type that provide 0 days of training per year. In other words, devoting 10 days a year to professional education benefits software results, even though 10 work days are used for this purpose.

Some companies have very impressive internal education programs for employees that the author thinks are often superior to academic education. Examples of these companies with excellent internal post-employment professional training include IBM, AT&T, Microsoft, Amazon, Apple, and Google.

Topic 14: Introduce software licensing and board certification

Physicians, lawyers, and many kinds of engineers have licenses and also various kinds of board certifications. As of 2017 software has many kinds of certification, but generally no formal state licensing.

There are computer engineer and software engineer licenses that may be needed for consulting engineers who work for public agencies and need to sign or stamp official government documents. These allow successful applicants to use "P.E." for "professional engineer" when signing contracts or reports. However, most kinds of software engineering personnel work under an industrial exemption clause and don't need licenses.

The basic issue is when and if software engineering licenses occur, what kinds of knowledge should be tested as part of the license process? Given the endemic problems that software has with poor quality control and bad metrics, it would be useful for software license exams to include topics such as knowledge of effective quality control and knowledge of effective quantification of software quality and productivity.

Topic 15: Move from custom and manual development to standard reusable components

Custom designs for software applications and manual coding by human programmers are intrinsically expensive, error-prone, and slow regardless of which programming languages are used and which development methodologies are used. Agile may be a bit faster than waterfall, but it is still slow compared to actual business needs.

The only effective solution for software engineering is to move away from manual custom development and towards construction of applications using standard certified and hardened reusable materials. The idea is to build software more like Ford builds automobiles on an assembly line rather than like the custom design and manual construction of a Formula 1 race car.

In fact when software reuse begins to top 75% on average then the same thing can happen with software that has already happened with automotive construction: robotic software development tools can replace human analysts and human programmers for the portions of software constructed from standard reusable components. Human analysis and human programmers will still be needed for creating custom designs and custom code for novel features, but not for generic applications constructed from certified reusable components.

An ordinary passenger car and a Formula 1 race car have about the same number of mechanical parts, but the race car costs at least 10 times more to build due to the large volumes of skilled manual labor involved. The schedule would be more than 10 times longer as well. Custom designs and manual construction are intrinsically slow and expensive in every industry.

If you compare the costs and schedules of building a 50-story office building, a 50,000 ton cruise ship, and a 50,000 function point software system the software is much more expensive and also much slower than the other two. When deployed the software is much less reliable than the other two and has many more defects that interfere with use than the other two. Worse, the software is much more likely to be attacked by external criminals seeking to steal data or interfere with software operation.

These problems are endemic but not impossible to cure. It is technically possible today in 2017 to build some software applications from standard reusable components. It is also possible to raise the immunity of software to external cyber-attack.

In the future, more and more standard components will expand the set of applications that can be assembled from certified standard parts free from security vulnerabilities rather than needing custom design and laborious manual coding that tend to introduce security flaws. Assembly from certified components can be more than 10 times faster and cheaper than the best manual methods such as agile, and also much more secure than today's norms where security vulnerabilities are rampant.

Summary and Conclusions on Software Professionalism

The author has worked in software since about 1965 and has seen quite a few technical advances over the years. For example, the author was at IBM during the period when function point metrics, formal inspections, and parametric estimation were first starting, as well as the creation of the relational data base model.

He also saw the introduction of the Apple I computer and the introduction of Microsoft Windows, as well as the introduction the early IBM PC followed by many others. The author has seen the available

programming languages expand from one (basic assembly) to over 3,000. He has seen the number of software methodologies grow from one (cowboy) to over 60 in 2017.

In the past, the author of this paper has worked as both the editor of a medical journal and of medical research papers for the Office of the Surgeon General and also as the editor and technical reviewer of a number of software journal articles and books for various publishers.

Medical papers devote about a third of the text to discussions of measures and metrics and include accurate quantified data. Software papers, on the other hand, devote hardly a paragraph to measures and metrics and seldom contain accurate quantified data.

As readers know medical practice has been the top learned profession for over 100 years. By contrast software is not even recognized as a true profession and is still classified as a craft circa 2017.

One reason for the low status of software is that software has failed to use effective metrics and measures. As a result, software has close to zero accurate data on software quality and productivity or the effectiveness of various methodologies and programming languages.

Software's current lack of a knowledge base of leading indicators for quality and costs is a professional embarrassment. Diagnosing software problems in 2017 is closer to medical diagnoses from 1817 before medicine adopted careful measures and accurate metrics.

From reading Paul Starr's book on the social transformation of American medical practice it was interesting to see that medicine was as chaotic and inept 200 years ago as software is in 2017.

Medical practices circa 1817 were alarmingly similar to software practices circa 2017. Both were unlicensed, unregulated, unmeasured, and both mixed quackery and harmful practices with beneficial practices without patients or clients having any way of knowing which was which.

References and Readings on Software and selected texts on medical practice

Control of Communicable Diseases in Man. U.S. Public Health Service, published annually. This book provided the format for the author's first book on software risks, Assessment and Control of Software Risks. The format worked well for both medical diseases and software risks. The format included frequency of the conditions, severity of the conditions, methods of prevention, and methods of treatment. A few topics such as quarantine were not used for software risks, although with cyber-attacks increasing in frequency and severity quarantine should be considered for software that has been attacked by viruses or worms both of which are highly contagious.

Starr, Paul. The Social Transformation of American Medicine; (Pulitzer Prize and Booker in 1984); Basic Books, 1982. This interesting book summarizes the steps used by the American Medical Association (AMA) to improve medical education and raise the professional status of physicians. The same sequence of steps would benefit software engineering.

Abran, A. and Robillard, P.N. “Function Point Analysis, An Empirical Study of its Measurement Processes”; IEEE Transactions on Software Engineering, Vol 22, No. 12; Dec. 1996; pp. 895-909.

Austin, Robert D. Measuring and Managing Performance in Organizations. New York, NY: Dorset House Press, 1996. ISBN 0-932633-36-6, 216 pages.

Black, Rex. Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. Wiley, 2009. ISBN-10 0470404159, 672 pages.

Boehm, Barry. Software Engineering Economics. Englewood Cliffs, NJ: Prentice Hall, 1981, 900 pages.

Bogan, Christopher E. and English, Michael J. Benchmarking for Best Practices. New York, NY: McGraw Hill. ISBN 0-07-006375-3, 1994, 312 pages.

Brooks, Fred. The Mythical Man-Month. Reading, Massachusetts: Addison-Wesley, 1974, rev. 1995.

Brown, Norm (Editor). The Program Manager’s Guide to Software Acquisition Best Practices. Version 1.0, July 1995. U.S. Department of Defense, Washington, DC, 142 pages.

Campbell-Kelly, Martin. A History of the Software Industry: from Airline Reservations to Sonic the Hedgehog. Cambridge, MA: The MIT Press, 2003. ISBN 0-262-03303-8, 372 pages.

Charette, Bob. Software Engineering Risk Analysis and Management. New York, NY: McGraw Hill, 1989.

Charette, Bob. Application Strategies for Risk Management. New York, NY: McGraw Hill, 1990.

Cohen, Lou. Quality Function Deployment – How to Make QFD Work for You. Upper Saddle River, NJ: Prentice Hall, 1995. ISBN 10: 0201633302, 368 pages.

Constantine, Larry L. Beyond Chaos: The Expert Edge in Managing Software Development. ACM Press, 2001.

Crosby, Philip B. Quality is Free. New York, NY: New American Library, Mentor Books, 1979, 270 pages.

Curtis, Bill, Hefley, William E., and Miller, Sally. People Capability Maturity Model. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1995.

DeMarco, Tom. Peopleware: Productive Projects and Teams. New York, NY: Dorset House, 1999. ISBN 10: 0932633439, 245 pages.

DeMarco, Tom and Lister, Tim. Waltzing with Bears: Managing Risks on Software Projects. New York, NY: Dorset House Press, 2003.

Department of the Air Force. Guidelines for Successful Acquisition and Management of Software Intensive Systems, Volumes 1 and 2. Hill Air Force Base, UT: Software Technology Support Center, 1994.

Dreger, Brian. Function Point Analysis. Englewood Cliffs, NJ: Prentice Hall, 1989. ISBN 0-13-332321-8, 185 pages.

Gack, Gary. Managing the Black Hole: The Executives Guide to Software Project Risk Thomson, GA: Expert Publishing, 2010, ISBN10: 1-935602-01-9.

Gack, Gary. *Applying Six Sigma to Software Implementation Projects*. https://www.isixsigma.com/operations/information-technology/applying-six-sigma-software-implementation-projects/.

Gilb, Tom and Graham, Dorothy. Software Inspections. Reading, MA: Addison Wesley, 1993, ISBN 10: 0201631814.

Grady, Robert B. Practical Software Metrics for Project Management and Process Improvement. Englewood Cliffs, NJ: Prentice Hall, ISBN 0-13-720384-5, 1992, 270 pages.

Grady, Robert B. & Caswell, Deborah L. Software Metrics: Establishing a Company-Wide Program. Englewood Cliffs, NJ: Prentice Hall, ISBN 0-13-821844-7, 1987, 288 pages.

Grady, Robert B. Successful Process Improvement. Upper Saddle River, NJ: Prentice Hall PTR, ISBN 0-13-626623-1, 1997, 314 pages.

Humphrey, Watts S. Managing the Software Process. Reading, MA: Addison Wesley Longman, 1989.

IFPUG Counting Practices Manual, Release 6. Westerville, OH: International Function Point Users Group, April 2015, 105 pages.

Jacobsen, Ivar, Griss, Martin, and Jonsson, Patrick. Software Reuse - Architecture, Process, and Organization for Business Success. Reading, MA: Addison Wesley Longman, ISBN 0-201-92476-5, 1997, 500 pages.

Jacobsen, Ivar et al. The Essence of Software Engineering; Applying the SEMAT Kernel. Addison Wesley Professional, 2013.

Jones, Capers. A Guide to Selecting Software Measures and Metrics, CRC Press, 2017.

Jones, Capers. The Technical and Social History of Software Engineering. Addison Wesley, 2014.

Jones, Capers. Estimating Software Costs, 2nd edition. New York, NY: McGraw Hill, 2007.

Jones, Capers and Bonsignour, Olivier. The Economics of Software Quality. Boston, MA: Addison Wesley, 2011, ISBN 978-0-13-258220-9, 587 pages.

Jones, Capers. “A Ten-Year Retrospective of the ITT Programming Technology Center”. Software Productivity Research, Burlington, MA, 1988.

Jones, Capers. Applied Software Measurement, 3rd edition. McGraw Hill, 2008.

Jones, Capers. Software Engineering Best Practices, 1st edition. McGraw Hill, 2010.

Jones, Capers. Assessment and Control of Software Risks. Prentice Hall, 1994, ISBN 0-13-741406-4, 711 pages.

Jones, Capers. Patterns of Software System Failure and Success. Boston, MA: International Thomson Computer Press, December 1995, 250 pages, ISBN 1-850-32804-8, 292 pages.

Jones, Capers. Software Assessments, Benchmarks, and Best Practices. Boston, MA: Addison Wesley Longman, 2000, 600 pages.

Jones, Capers. Software Quality – Analysis and Guidelines for Success. Boston, MA: International Thomson Computer Press, ISBN 1-85032-876-6, 1997, 492 pages.

Jones, Capers. The Economics of Object-Oriented Software. Burlington, MA: Software Productivity Research, April 1997, 22 pages.

Jones, Capers. Becoming Best in Class. Burlington, MA: Software Productivity Research, January 1998, 40 pages.

Kan, Stephen H. Metrics and Models in Software Quality Engineering, 2nd edition. Boston, MA: Addison Wesley Longman, ISBN 0-201-72915-6, 2003, 528 pages.

Keys, Jessica. Software Engineering Productivity Handbook. New York, NY: McGraw Hill, ISBN 0-07-911366-4, 1993, 651 pages.

Love, Tom. Object Lessons. New York: SIGS Books, ISBN 0-9627477 3-4, 1993, 266 pages.

McCabe, Thomas J. “A Complexity Measure”. IEEE Transactions on Software Engineering; December 1976; pp. 308-320.

McConnell, Steve. Software Project Survival Guide. Redmond WA: Microsoft Press, 1998.

McMahon, Paul. 15 Fundamentals for Higher Performance in Software Development. PEM Systems, 2014.

Melton, Austin. Software Measurement. International Thomson Press, ISBN 1-85032-7178-7, 1995.

Multiple authors. Rethinking the Software Process, (CD-ROM). Lawrence, KS: Miller Freeman, 1996. (This is a new CD ROM book collection jointly produced by the book publisher, Prentice Hall, and the journal publisher, Miller Freeman. This CD ROM disk contains the full text and illustrations of five Prentice Hall books: Assessment and Control of Software Risks by Capers Jones; Controlling Software Projects by Tom DeMarco; Function Point Analysis by Brian Dreger; Measures for Excellence by Larry Putnam and Ware Myers; and Object-Oriented Software Metrics by Mark Lorenz and Jeff Kidd.)

Myers, Glenford. The Art of Software Testing. New York: John Wiley & Sons, 1979, ISBN 0-471-04328-1, 177 pages.

Paulk, Mark et al. The Capability Maturity Model; Guidelines for Improving the Software Process. Reading, MA: Addison Wesley, ISBN 0-201-54664-7, 1995, 439 pages.

Perry, William E. Data Processing Budgets - How to Develop and Use Budgets Effectively. Englewood Cliffs, NJ: Prentice Hall, ISBN 0-13-196874-2, 1985, 224 pages.

Perry, William E. Handbook of Diagnosing and Solving Computer Problems. Blue Ridge Summit, PA: TAB Books, Inc., 1989, ISBN 0-8306-9233-9, 255 pages.

Pressman, Roger. Software Engineering – A Practitioner's Approach, 6th edition. New York NY: McGraw Hill, 2005, ISBN 0-07-285318-2.

Putnam, Lawrence H. Measures for Excellence -- Reliable Software on Time, Within Budget. Englewood Cliffs, NJ: Yourdon Press - Prentice Hall, ISBN 0-13-567694-0, 1992, 336 pages.

Putnam, Lawrence H and Myers, Ware. Industrial Strength Software - Effective Management Using Measurement. Los Alamitos, CA: IEEE Press, ISBN 0-8186-7532-2, 1997, 320 pages.

Radice, Ronald A. High Qualitiy Low Cost Software Inspections. Andover, MA: Paradoxicon Publishing, ISBN 0-9645913-1-6; 2002, 479 pages.

Royce, Walker E. Software Project Management: A Unified Framework. Reading, MA: Addison Wesley Longman, 1998, ISBN 0-201-30958-0.

Rubin, Howard. Software Benchmark Studies For 1997. Pound Ridge, NY: Howard Rubin Associates, 1997.

Rubin, Howard (Editor). The Software Personnel Shortage. Pound Ridge, NY: Rubin Systems, Inc., 1998.

Shepperd M. "A Critique of Cyclomatic Complexity as a Software Metric". Software Engineering Journal, Vol. 3, 1988; pp. 30-36.

Strassmann, Paul. The Squandered Computer. New Canaan, CT: The Information Economics Press, ISBN 0-9620413-1-9, 1997, 426 pages.

Strassmann, Paul. Information Payoff. Stamford, CT: Information Economics Press, 1985.

Strassmann, Paul. Governance of Information Management: The Concept of an Information Constitution, 2nd edition; (eBook). Information Economics Press, Stamford, Ct; 2004.

Strassmann, Paul; Information Productivity. Stamford, CT: Information Economics Press, 1999.

Stukes, Sherry, Deshoretz, Jason, Apgar, Henry and Macias, Ilona. Air Force Cost Analysis Agency Software Estimating Model Analysis. TR-9545/008-2, Contract F04701-95-D-0003, Task 008, Thousand Oaks, CA: Management Consulting & Research, Inc., September 30, 1996.

Symons, Charles R. Software Sizing and Estimating – Mk II FPA (Function Point Analysis). Chichester: John Wiley & Sons, ISBN 0 471-92985-9, 1991, 200 pages.

Thayer, Richard H. (editor). Software Engineering and Project Management. Los Alamitos, CA: IEEE Press, ISBN 0 8186-075107, 1988, 512 pages.

Umbaugh, Robert E. (Editor). Handbook of IS Management, Fourth Edition. Boston, MA: Auerbach Publications, ISBN 0-7913-2159-2, 1995, 703 pages.

Weinberg, Gerald M. The Psychology of Computer Programming. New York NY: Van Nostrand Reinhold, 1971, ISBN 0-442-29264-3, 288 pages.

Weinberg, Gerald M. Becoming a Technical Leader. New York NY: Dorset House, 1986, ISBN 0-932633-02-1, 284 pages.

Weinberg, Dr. Gerald. Quality Software Management - Volume 2 First-Order Measurement. New York, NY: Dorset House Press, ISBN 0-932633-24-2, 1993, 360 pages.

Wiegers, Karl A. Creating a Software Engineering Culture. New York, NY: Dorset House Press, 1996, ISBN 0-932633-33-1, 358 pages.

Yourdon, Ed. Death March - The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects. Upper Saddle River, NJ: Prentice Hall PTR, ISBN 0-13-748310-4, 1997, 218 pages.

ARTICLE

Integrating Program Management and Systems Engineering

by

Ralph R. Young, Editor, SyEN

Note: The following text relies heavily on that provided in *IPMSE*.

Introduction (for those who have not been regular readers of this column or who would like to refresh themselves concerning the Integration Initiative)

Every once in a great while, a book comes along that is destined to have a huge impact. *Integrating Program Management and Systems Engineering* (*IPMSE*, Wiley, 2017) is such a book. This new book was five years in the making, provides insightful new research, and is a

collaboration of the International Council on Systems Engineering (INCOSE), the Project Management Institute (PMI), and the Consortium for Engineering Program Excellence (CEPE) at the Massachusetts Institute of Technology (MIT). We at PPI believe that this book has the potential to significantly improve program results and the practice of systems engineering. Commitment is needed from program managers and systems engineers to build collaboration. I ask your consideration in becoming part of the "Integration Initiative" to strengthen collaboration and change mind sets (mental inclinations, beliefs, and attitudes).

In SyEN 52 (April 2017), we reported about the publication of this book and noted the availability of a webinar http://www.incose.org/ProductsPublications/webinars/2017/06/19/default-calendar/webinar-100-integrating-program-management-and-systems-engineering sponsored by INCOSE to provide and share more information about it. Another webinar was sponsored by INCOSE on June 19, 2017 to create additional awareness of the Integration Initiative as well as the follow-on activities that are being planned, including a virtual conference planned for September 27, 2017.

In SyEN 54 (June 2017), we provided a detailed review of "The Book". If you've not read this review, please consider doing so as a good starting point (see www.ppi-int.com/systems-engineering/SYEN14). In the Preface, written by Alan Harding (President of INCOSE) and Mark Langley (President & CEO, Project Management Institute), **they emphasize that senior executives within corporations, governments, professional associations, academia, and research must change their mindsets.** These leaders must see the connections between strategy, benefits, performance, and capabilities, and work within their organizations to remove gaps and improve performance. They must recognize the value that their organizations could gain from even incremental efforts to reduce wasted investments due to poor program execution. They must ensure their organizations learn from examples of success and failure (case studies of both are provided), and utilize that learning to continuously improve their own practices. **Most importantly, they must stand with their program managers and chief systems engineers and lead the change toward a new mindset.** We also provided a **link to a table: What's Different about an Integrated Approach? This table identifies 32 factors and describes how they differ between Less Successful Approaches and a Superior Approach**. We indicated that we plan to provide a series of articles to be published in the next several issues of SyEN. Our intent in doing this is to support the systems engineering profession, especially subscribers of SyEN, in taking advantage of the opportunities available to further strengthen and improve the professions of program management and systems engineering.

In SyEN 55 (July 2017), we noted that one of the many contributions of the book is the **rich set of references** identified during the research as well as studies made by the collaboration of PMI, INCOSE, and CEPE to investigate questions posed by the research. This collaboration was called the PMI/INCOSE/MIT Alliance Team. This group helped to design the vision for the book, helped organize and enable the research activities that supported the knowledge base for the book, organized the dissemination of findings at conferences and other venues, and assisted with development and publication. The **challenge** that was identified by PMI and INCOSE in 2011 was:

> **While program management has overall program accountability and systems engineering has accountability for the technical and systems elements of programs, some systems engineers and program managers consider their work activities as separate from each other rather than part of the organic whole.[1]**

We stated that it is our fervent hope that this series of articles will help keep this initiative alive in your mind and actions. The bottom line is that the current performance of program management and systems engineering is inadequate; we must pursue concerted efforts toward significantly improved integration. Each of us must make changes in our daily work. Our intent is to support the Integration Initiative and the systems engineering profession by providing continuing awareness and reinforcement concerning the themes presented in The Book**.**

In Search of Integrated Solutions

Part I of the book explores the importance of complex programs for society and the positive impact they can have when they perform well, as well as the downsides when they do not.

Toward a New Mindset

Chapter 1 of The Book uses the example of SpaceX Corporation to illuminate the difference in performance that can result from a unique approach to managing engineering programs. It explores how performance outcomes may result from a combination of factors, but ultimately are rooted in the capabilities that organizations have to be both innovative and efficient. Program Management and Systems Engineering are key players in creating the conditions for success or failure of these programs. Why is it that despite having advanced knowledge, tools, and capabilities, and even having demonstrated that it is possible to do amazing things, best efforts often end in disappointment or failure?

The new discipline of systems engineering rose to prominence because of the technical challenges of sending people to the moon and returning them back safely, the Apollo Program,

[1] Langley, Robitaille, and Thomas, *Toward a New Mindset: Bridging the Gap between Program Management and Systems Engineering,* 2011, p. 24.

and other program activities that touched research, education, defense, and ultimately commercial products. The integration of systems engineering with program management approaches proved to be critical to the success of the Apollo Program.

Although great things have been accomplished, often it appears that the inspiration to accomplish moon-shot like things is diminishing. The Space Exploration Technologies Corporation, better known as SpaceX, during its relatively short existence since its inception in 2002, has not only accomplished many important and inspiring things, but has done so in a way that significantly outperforms all competitors, both government and private, and seems poised to create a renaissance in the space sector. Perhaps as compelling as the impressive launch hardware and support systems that SPACEX has created is the way that it has been able to assemble teams and to design a work system that enables them to be incredibly productive and produce complex systems quickly. Chapter 1 of *IPMSE* provides a case study that describes how SpaceX has been able to accomplish this. The development time and cost of about 100 major flight-proven products are several times smaller than the competition. The reduced costs directly impact the marketplace for launch services. SpaceX's low costs enable the company to enjoy comfortable profit margins on each launch. The approach of using operational systems as test beds to learn, improve the products, and develop new technologies is intrinsic to the SpaceX development process. SpaceX has managed to build an organization that is capable of rapid learning. It has demonstrated that it can identify faults and implement corrective actions rapidly. Some of the factors that have enabled SpaceX to be so successful are focus on simplicity in the design, colocation, vertical integration, having mission assurance embedded in routine operations, and the culture of the company. The company's culture is arguably the most important factor; it promotes very high levels of teamwork, mutual support, coordination, and communication in the spirit of pushing the boundaries. Employees are encouraged to continuously seek better solutions. Information technology is regarded as a critically important activity, supporting efficient execution of all other activities and extraordinarily efficient communication, coordination, and approval tools.

SpaceX is not the only innovative company working to make the vision of expanded human presence in space a reality. There are a number of startups that are making inspiring progress in a number of different areas. However, SpaceX provides a good example of what can be accomplished using the right approach. Its success is a result of a new way of working together that combines management practices, engineering practices, product strategy, organizational processes and tools, and a leadership climate that encourages responsibility, innovation, learning, and high performance. The example of SpaceX demonstrates that organizations can bring together diverse skill sets to overcome challenging problems, and ultimately make miracles seem commonplace.

The Book *IPMSE* argues that these behaviors can be achieved in a wider array of settings where those in the management and the technical disciplines learn to work together in order to create the benefits needed by their customers and other stakeholders.

Despite the perceptions and realities related to program performance, one of the underlying challenges i*s building the will within organizations to invest in improving engineering program capabilities*. Many executive leaders often do not view programs within their companies as being strategic, and activities that are not strategic often do not receive executive-level attention. Enhancing engineering program capabilities must start with a better understanding of the linkages between engineering programs and strategy among executives. We also need to stress that investing in strengthening such capabilities could help reduce the millions of dollars at risk from poor performance. Achieving these objectives necessitates an organizational culture that links talent management with strategy along with ongoing investments to improve workforce capabilities and enabling systems, structures, and process.

Summary

Despite having the same roots and following similar paths to professionalism, the program management and systems engineering disciplines have experienced somewhat divergent evolutions. Today, elements of this divergent evolution appear to have an impact on the ability of those involved in the two disciplines to effectively align their work practices and collaborate. In addition, the lack of organizational systems also inhibits effective engineering program management and performance. Effective practices are critical for integrating efforts to deliver results in program environments. Engineering program environments require good planning approaches, proactive risk management, stakeholder engagement, and other, similar, capabilities. New research in *IPMSE* indicates that there is need for stronger integration and collaboration between and among systems engineers and program managers. This book blazes a new path by focusing on approaches to better enable collaborative work between program managers and systems engineers. The key enablers include recognizing the linkages between programs and strategy, and supporting program leadership in enabling alignment and collaboration within the program team.

Editor's Note: The INCOSE Technical Operations Briefing concerning this book is available here.

SYSTEMS ENGINEERING NEWS

PivotPoint's SafetyML Safeguards Safety-Critical Systems

PivotPoint Technology, introduces the SafetyML™ (Safety Modeling Language™) and Model-Based Functional Safety (MBFS) training for specifying the analyses, designs and architectures of safety-critical systems. The SafetyML v. 1.0 is an OMG UML™ and OMG SysML™ compatible profile and model library that supports the analysis, design, verification and validation of safety-critical functions across a broad range of safety-critical domains. These safety-critical domains include, but are not limited to, automotive, aerospace, medical devices and railway transportation. "Exponential increases in software code bases, proliferating safety regulations and the propagation of deep learning AI bots are creating a 'perform storm' for safety-critical systems that are also software-intensive," said Cris Kobryn, Founder and CEO of PivotPoint Technology. Kobryn has been architecting and designing secure distributed computer systems for three decades, and is known for successfully leading the UML 1.x, UML 2.0, and SysML international standardization teams. "Tracing and analyzing a system fault in Millions of Lines of Code (MLOC) using traditional safety analysis techniques is analogous to finding a proverbial 'needle in a haystack'. "A primary goal of the SafetyML and our Model-Based Functional Safety (MBFS) training is to improve how we analyze, design and architect safety-critical systems on the 'left side' of the System V-Model, so that so that we can efficiently verify and validate them on the 'right side' of the System V-Model." The SafetyML is implemented as an OMG UML profile (UML dialect) and model library so that it can be efficiently implemented in a wide range of architecture modeling tools that comply with the OMG UML and OMG SysML standards. The SafetyML also supports a broad range of international safety standards including, but not limited to, ISO 26262 (Automotive), DO-178-C (Aerospace), ISO 14971 (Medical Devices) and EN 50128 (Railway Transportation). In addition, SafetyML is designed to complement PivotPoint's CyberML™ (Cyber Modeling Language™) profile and model library, so architects and engineers can specify safety-critical systems that are also cyber-secure.

More information

LinkedIn Open-Source Tools for Managing Website Outages

LinkedIn's engineering organization has made a couple of key tools available as open source projects to help businesses deal with what happens when their applications fail. The tools help organizations automatically contact engineers to deal with issues that pop up in their applications. The tools are potentially relevant in an engineering environment.

Iris, named after the Greek messenger goddess, notifies users of alerts generated by company systems. For example, Iris can be set up to contact an on-call engineer and notify the site reliability organization if a production server goes down. If users don't respond to a first notification, Iris can be configured to send subsequent messages until it receives a response.

Who the system contacts is driven by Oncall, another project that was released today. That service lets companies lay out a schedule for who is responsible to deal with issues when they arise. Users lay out their schedule in a calendar, and Iris will use that information to drive notifications to the right people.

These projects are designed to make it easier for companies to automate the process of notifying their engineers of outages and other issues. LinkedIn created Iris and Oncall as part of the company's move towards increasingly automatic notifications. Prior to the system's implementation, the company's Network Operations Center engineers notified on-call engineers manually when issues arose.

Deploying the systems inside a company is supposed to be fairly easy, according to the LinkedIn engineers who helped build them.

"Different companies might use different deployment tools, but Docker is a good example: You can get Iris and Oncall running in four commands total," said Daniel Wang, a site reliability engineer at LinkedIn. "So, we've tried to make that process as easy as possible."

More information

FEATURED ORGANIZATIONS

American Society for Engineering Education

Founded in 1893, the American Society for Engineering Education (ASEE) is a nonprofit organization of individuals and institutions committed to furthering education in engineering and engineering technology.

It accomplishes this mission by promoting excellence in instruction, research, public service, and practice; exercising worldwide leadership; fostering the technological education of society; and providing quality products and services to members. In pursuit of academic excellence, ASEE develops policies and programs that enhance professional opportunities for engineering faculty members, and promotes activities that support increased student enrollments in engineering and engineering technology colleges and universities. Strong communication and collaboration with national and international organizations further advances ASEE's mission.

ASEE also fulfills its mission by providing a valuable communication link among corporations, government agencies, and educational institutions. ASEE's 12,000+ members include deans, department heads, faculty members, students, and government and industry representatives who hail from all disciplines of engineering and engineering technology. ASEE's organizational membership is composed of 400 engineering and engineering technology colleges and affiliates, more than 50 corporations, and numerous government agencies and professional associations

More information

INCOSE Connect – Portal for Member Resources

INCOSE Connect is a resource that is available to INCOSE Members. To access INCOSE Connect, at the INCOSE web site, select the INCOSE CONNECT link at the upper left of the home page, and sign in with your INCOSE Username and Password.

The *Library* in INCOSE Connect includes:

- INSIGHT: INSIGHT is a publication of INCOSE in partnership with John Wiley & Sons. *INSIGHT*'s mission is to provide informative articles on advancing the state of the practice of systems engineering. The intent is to accelerate the dissemination of knowledge to close the gap between the state of practice and the state of the art as captured in *Systems Engineering*, the Journal of INCOSE, also published by Wiley. "*Discovering the Strategy for Whole System Modeling*" by Jack Ring that is highlighted in the SE Publications Section of this issue.
- eNote is a monthly update on one page delivered directly to members' email box. This compilation of INCOSE news, event announcements, and items will be of interest to our systems engineering community and is a benefit of membership. eNote features INCOSE information as well as notes from systems engineering and related fields.
- Tutorials (including Agile Working Group (WG), Requirements WG, and Training WG tutorials)
- Webinars (INCOSE Webinars, Chapter Webinars, WG Webinars, and BKCASE Town Hall Recordings.)

CONFERENCES AND MEETINGS

For more information on systems engineering related conferences and meetings, please proceed to our website.

SOME SYSTEMS ENGINEERING-RELEVANT WEBSITES

Engineering Systems

Engineering Synergy: learning together to join-up our thinking and create desired outcomes

http://myengineeringsystems.co.uk/

Systems Approaches to Management

Image source

by Michael C. Jackson

Book Description (from the Amazon web site):

Systems Thinking is a new paradigm set to revolutionize management practice in the 21st century. Systems Approaches to Management is the most comprehensive guide available to the application of this new paradigm in the field of management. It traces the emergence of holistic thinking in disciplines such as biology, control engineering, sociology, and the natural sciences. It provides a critique, based upon social theory, of the range of systems approaches, methodologies, models and methods. The book offers numerous case studies to illustrate systems thinking applied to management. It introduces critical systems thinking' as a coherent framework that brings unity to the diversity of different systems approaches and advises managers, consultants, scholars, and students on their use. It provides discussion concerning chaos and complexity theory, the learning organization, system dynamics, living systems theory, soft systems methodology, interactive management, interactive planning, total systems intervention, autopoiesis, management cybernetics, the viable system model, operations research (hard and soft), systems analysis, systems engineering, general system theory, sociotechnical systems thinking, the fifth discipline, social systems design, team integrity, postmodern systems thinking, critical systems thinking, and much more. Considers the work of Ackoff, Banathy, Beer, Capra, Checkland, Churchman, Eden, Emery, Flood, Forrester, Friend, Freire, Jackson, Jantsch, Linstone, Luhmann, Mason, Maturana, Miller, Mitroff, Prigonine, Rosenhead, Senge, Stacey, Trist, Ulrich, Varela, Vickers, von Bertalanffy, Warfield, Wheatley, Wiener, and many more.

More information

Don't Panic! The Absolute Beginner's Guide to Model-Based Systems Engineering

Image source

by Jon Holt and Simon Perry

INCOSE UK members, Prof. Jon Holt and Simon Perry, have co-authored the first book in the 'Don't Panic!' series titled 'The Absolute Beginner's Guide to Model-Based Systems Engineering'. The book is aimed at systems engineers, managers, board members, students and anyone interested in using modeling to enhance and improve their existing Systems Engineering activities.

Learn more and order your copy

Facilitating the Project Lifecycle

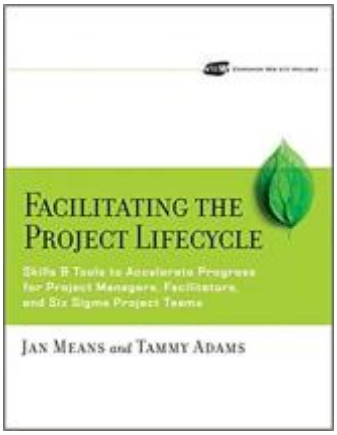

Image source

by Jan Means and Tammy Adams

Book Description (from the Amazon web site):

Step by step, *Facilitating the Project Lifecycle* guides the project manager/facilitator in making smart choices about when and how to pull key talent together to spell success for the project and ultimately the organization. The authors will help you understand the benefits of using facilitated group work sessions to get real work done during a project and get it done better and more efficiently than more traditional individual work approaches. In addition, the book includes:

- Recommendations for capitalizing on group knowledge to accelerate the building of key project deliverables and ensure their quality as they are built
- A work session structure for planning, delivering, and following up facilitated work sessions
- Guides for building key project deliverables
- Sample agendas
- Proven techniques for managing the group dynamics

More information

"Discovering the Strategy for Whole System Modeling"

by Jack Ring, INCOSE Fellow

This article was published in the June 2017 issue of *INCOSE INSIGHT* (VOLUME 20 / ISSUE 2). It provides an interim report of an inquiry into the strategy for evolving a whole system modelling capability. Society needs better interventions for a wide variety of problematic situations. A holistic approach to the rapid composition of a whole system model is needed. Better models initialize continuous improvement of the whole system. The author expects that whole system modeling will converge semantics, semiotics, and systemics to yield dramatic new levels of understanding, productivity, and innovation on the part of those who implement intervention systems. As system problems grow larger in extent, variety, and ambiguity, those charged with devising interventions become beset by two gremlins, cognitive overload and under conceptualization. Whole system modeling can forestall these two gremlins. The cost of whole system modeling can be significant but returns many-fold by sot avoidance downstream in the balance of the system realization activity. Objectives respond to societal needs for better, faster, cheaper systems that enable successful intervention in problematic situations. When executed an effective strategy can foster a whole system modeling capability that reaches worldwide, spans multiple domains, and achieves at least ten times the level of productivity and innovation typical of traditional systems engineering.

More information

"Engineering the Virtual or Collaborative SoS"

by Eric Honour, INCOSE Fellow

The largest, most complex, and most prevalent systems of systems (SoS) are virtual or collaborative in nature. In such a SoS, there is no central authority to perform systems engineering. Yet, the virtual or collaborative SoS does grow, adapt, and evolve in response to the needs of the many stakeholders. Changes in the SoS happen because of many changes in the constituent systems (CS), with each change occurring when the CS owners deem it useful to change. The systems engineering activities in the CS can

be used to guide and affect the SoS through a collaborative and competitive environment in which each CS systems engineer seeks to optimize the SoS behavior for their own purposes. This requires a different approach to the systems engineering activities, in which greater emphasis is on the higher-level analysis, architecting, and quantification. This paper describes the modified systems engineering approach, showing how to engineer the SoS from within the scope of a CS.

More information

SYSTEMS ENGINEERING TOOLS NEWS

CRADLE

Structured Software Systems Ltd (3SL) produces tools that address the entire lifecycle. "Cradle" is how 3SL provides tools as a product. Cradle is a tool to load, create, inter-link, and publish information for all stages in a systems engineering project, using agile, iterative, or phase-based approaches. It is user-definable, scalable, flexible, and secure. It can be deployed locally, to remote sites or partners, or delivered through SaaS from any private or public cloud. The most recent Cradle-7 product addresses an integrated requirements management and systems engineering environment.

More information

AGILECRAFT 10X

AgileCraft 10X is a new scaled Agile platform brought to market with the clear goals of accelerating agile transformation success, increasing strategic alignment, and delivering an experience that accelerates time to value.

Features include:

- Program and Portfolio-level hubs that enable Release Engineers, Program Managers, and Portfolio Managers to plan, manage, and report on all work in one integrated view.
- A new Product Management suite including Live Agile Roadmaps, Business Model Canvas, Vision Maps, Agile Mind Maps, Personas, and Ideation.
- A new “Why” button enabling any user to fully understand how their work advances the overall strategy of the organization.
- Mobile access including real-time analytics and key reports.

A new Agile Enablement Framework, supplementing and re-enforcing the agile knowledge and education organizations are already receiving.

Take a product tour

EDUCATION AND ACADEMIA

Online Program Wins Engineering Education Award

In collaboration with Boeing and edX, MIT has been honored with the 2017 Excellence in Engineering Education Collaboration Award by the American Society for Engineering Education (ASEE). The team was chosen for its design and development of a new four-course online professional certification program called Architecture and Systems Engineering: Models and Methods to Manage Complex Systems. The curriculum explores state-of-the-art practices in systems engineering and also demonstrates the value of models in enhancing system engineering functions and augmenting tasks with quantitative analysis.

The program launched last September and ran through March. Nine faculty members from MIT and more than 25 industry experts from Boeing, NASA, IBM, Apple, General Electric, General Motors, and other companies developed content for the courses. More than 1,600 professionals passed the four courses and earned a certificate in the first run of the program. Currently in its second run, the program is now accepting enrollments for September.

The program is delivered on the edX platform, with integrated peer-to-peer assessments, group projects, discussion forums, polls, and surveys. In course feedback on the program, more than 93 percent of survey respondents rated the instructors and materials as “good,” “very good,” or “excellent”.

To earn a certificate, students must complete four courses: Architecture of Complex Systems; Models in Engineering; Model-Based Systems Engineering: Documentation and Analysis; and Quantitative Methods in Systems Engineering. Upon completion, participants are expected to understand and analyze complex systems, perform model management, frame systems architecture as a series of decisions, articulate the benefits and challenges of model-based systems engineering, and demonstrate a comprehensive knowledge of the key aspects of systems engineering.

The award was presented at the 2017 ASEE Annual Conference in Columbus, Ohio, during the Industry Day Plenary Session on June 27.

More information

International Standards Organization

The International Standards Organization (ISO) www.iso.org/home.html has established an Online Browser Platform (OBP), which supports viewing (no printing or saving) of the informative clauses in many standards. Normative text is not available, but this provides a preview of the introduction, scope, and other normative text that might be useful for determining the usefulness of a particular standard. The committee search page for the Online Browser Platform is at https://www.iso.org/obp/ui/#search. The main page is https://www.iso.org/obp/ui/.

ISO has made a number of standards “free available”, that is, free-of-charge. Many of these standards and technical reports provide frameworks and foundations for other standards projects, so are useful during the development process, but in other cases, ISO has agreed to requests by standards committees to make certain standards available without any cost recovery. See http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html.

Some information on standards projects in ISO/IEC JTC 1 Subcommittee 7, Systems and Software Engineering, is available at: http://isotc.iso.org/livelink/livelink?func=ll&objId=8914042&objAction=browse&sort=name. Much of the site is password protected, but the meetings and resolutions page provides reports presented at Plenary (generally in the spring) and interim (generally in autumn) meetings. A standard from SC7 that is in development is ISO/IEC 24748, with the general title Systems and software engineering — Life cycle management.

ISO/IEC 24748 consists of the following parts:

- Part 1: Guidelines for life cycle management
- Part 2: Guidelines for the application of ISO/IEC 15288 (System life cycle processes)
- Part 3: Guide to the application of ISO/IEC 12207 (Software life cycle processes)
- Part 4: Systems engineering planning [ISO/IEC/IEEE]
- Part 5: Software development planning [ISO/IEC/IEEE]
- Part 6: Guide to system integration engineering [ISO/IEC TS]

INCOSE participates in ISO/IEC JTC1 Subcommittee 27, IT Security Techniques, through the US Technical Advisory Group to SC27. This provides INCOSE an opportunity to participate without waiting for approval of official status as a Category A Liaison. Additional information is available

at http://www.din.de/en/meta/jtc1sc27. Development of positions is led by the INCOSE Systems Security Engineering WG.

ISO TC 292 recently established a public web site on security and resilience at http://www.isotc292online.org/ . There are descriptions of the many standards projects related to the topics.

IEEE Announces IT Healthcare Standard

IEEE and the IEEE Standards Association (IEEE-SA), announced the availability of IEEE 11073-20702™—Health informatics: Point-of-Care Medical Device Communication—Standard for Medical Devices Communication Profile for Web Services. IEEE also announced the approval of two new healthcare-related standards projects: IEEE P1708™—Standard for Wearable Cuffless Blood Pressure Measuring Devices, and IEEE P1752™—Standard for Mobile Health Data.

Sponsored by the IEEE 11073™ Standards Committee (EMB/11073), IEEE 11073 family of standards are utilized to structure data and enable data transmission across multiple healthcare devices, ensuring effective interoperability and communication between medical, health care and wellness devices, as well as with external computer systems. IEEE 11073-20702 defines a communication protocol specification for a distributed system of point-of-care (PoC) medical devices and medical IT systems that need to exchange data, or safely control networked PoC medical devices, by profiling Web Service specifications.

More information

SOME DEFINITIONS TO CLOSE ON

System(s) of Systems, System(s) of Systems Engineering (SoSE)

System of Systems: Modern systems that comprise system of systems problems are not monolithic; rather they have five common characteristics: operational independence of the individual systems, managerial independence of the systems, geographical distribution, emergent behavior and evolutionary development.

Source: Sage, A.P., and C.D. Cuppan. "On the Systems Engineering and Management of Systems of Systems and Federations of Systems," Information, Knowledge, Systems Management, Vol. 2, No. 4, 2001, pp. 325-345.

System of Systems: System of systems problems are a collection of trans-domain networks of heterogeneous systems that are likely to exhibit operational and managerial independence, geographical distribution, and emergent and evolutionary behaviors that would not be apparent if the systems and their interactions are modeled separately.

Source: DeLaurentis, D. "Understanding Transportation as a System of Systems Design Problem," 43rd AIAA Aerospace Sciences Meeting, Reno, Nevada, January 10-13, 2005. AIAA-2005-0123.

System of Systems: A “system of systems” (SoS) is an SOI whose elements are managerially and/or operationally independent systems. These interoperating and/or integrated collections of constituent systems usually produce results unachievable by the individual systems alone.
Source: INCOSE. 2015. Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, version 4.0. Hoboken, NJ, USA: John Wiley and Sons, Inc.

System of Systems: A system of systems (SoS) brings together a set of systems for a task that none of the systems can accomplish on its own. Each constituent system keeps its own management, goals, and resources while coordinating within the SoS and adapting to meet SoS goals.
Source: ISO/IEC/IEEE 15288, Annex G, 2015

Family of systems (FoS): is defined as a set of systems that provide similar capabilities through different approaches to achieve similar or complementary effects. The family of systems does not acquire qualitatively new properties as a result of the grouping. In fact, the member systems may not be connected into a whole.

Source: Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering. Systems Engineering Guide for Systems of Systems, Version 1.0. Washington, DC: ODUSD(A&T)SSE, 2008.

System-of-Systems Engineering (SoSE): System-of-Systems Engineering (SoSE) is a set of developing processes, tools, and methods for designing, re-designing and deploying solutions to System-of-Systems challenges.
Source: http://en.wikipedia.org/wiki/System_of_systems_engineering

System-of-Systems Engineering (SoSE): System of Systems (SoS) Engineering deals with planning, analyzing, organizing, and integrating the capabilities of a mix of existing and new systems into a SoS capability greater than the sum of the capabilities of the constituent parts.
Source: Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering. Systems Engineering Guide for Systems of Systems, Version 1.0. Washington, DC: ODUSD(A&T)SSE, 2008.

PPI AND CTI NEWS

Appreciation Award - CTI and Joshua Freeman

What a privilege for our General Manager Josh Freeman to have received a Certificate of Appreciation for CTI’s support of the INCOSE Chesapeake Chapter. CTI has previously provided it’s CSEP Exam

Preparation training to the chapter. This award was received at the INCOSE International Symposium in Adelaide recently.

PPI's Software Engineering Course

At the end of July, PPI ran a pilot delivery to a review team of our much anticipated, new and improved Software Engineering 5-Day training course. The pilot was conducted in gorgeous Stellenbosch, Western Cape, South Africa, with a review team of PPI leaders and invited reviewers from industry. The course will be available for on-site delivery in a few weeks' time, and public delivery later this year.

The course approaches software development from a perspective of both modern development methodologies and systems thinking: software engineering as an iterative and concurrent, interdisciplinary, collaborative approach between multiple parties in engineering both small and large, simple and complex software-intensive systems. Course subject areas provide actionable learning about creating cost-effective software solutions to practical software problems by applying scientific and practical knowledge to software development.

Our Professional Team Continues to Grow

We warmly welcome Mr. Hanno Retief and Mr. Paul Davies to the PPI/CTI team. Paul Davies will be initially expand CTI's capacity, in response to ever-increasing demand, to deliver our CSEP Examination Preparation Training. Paul has long been highly regarded for his expertise in systems engineering. Hanno Retief has lead the redevelopment and delivery of our new Software Engineering 5-Day Course. He will present this exciting new course worldwide.

PPI's Systems Engineering Goldmine (SE Goldmine)

SE Goldmine features:

- thousands of downloadable documents, presently at 2.9 GB and increasing almost daily
- searchable database by description, title, keywords, date, source, etc.
- extensive library of standards, and links to standards
- searchable systems engineering-relevant definitions, 7,800+ defined terms, also increasing daily.

This content is provided as a free service to the SE community. You may request complimentary SE Goldmine access by clicking here.

PPI's First Cohort of the INCOSE Technical Leadership Institute

In 2015, INCOSE Institute for Technical Leadership accepted members into the first cohort to participate in its technical leadership program. The two-year program seeks to accelerate the development of systems engineering leaders who will exemplify the best of the organization and the systems engineering discipline.

Two years later, at the 27th Annual INCOSE International Symposium held in Adelaide, Australia, members of the first cohort including CTI's David Mason and PPI's Suja Joseph-Malherbe were inducted as full members of the Institute.

Our Contribution to Systems Engineering Standards

PPI's Managing Director Robert Halligan CPEng FIE Aust continues his decades of contribution to the development of systems engineering standards, most recently acting as a reviewer of the DIS version of ISO/IEC 24748, Systems and software engineering — Life cycle management, Part 1: Guidelines for life cycle management. Another part of ISO/IEC 24748 on which Robert has recently worked is Part 6: Guide to system integration engineering, a standard that Robert recommends to PPI clients.

UPCOMING PPI AND CTI PARTICIPATION IN PROFESSIONAL CONFERENCES

PPI will be participating in the following upcoming events.

SWISSED 2017
4 September 2017
Lake Side, Zürich

NZDIA Forum 2017
(Exhibiting)
10 - 11 October 2017
Wellington, New Zealand

13th Annual INCOSE South Africa Conference
(Exhibiting)
11 - 13 October 2017
Pretoria, South Africa

11th Annual INCOSE Great Lakes Regional Conference (GLRC11)
11 - 14 October 2017
Twin Cities, Minnesota, USA

36th International Conference on Conceptual Modeling (ER 2017)
6 - 9 November 2017

Valencia, Spain

INCOSE UK, Annual Systems Engineering Conference (ASEC) 2017

(Exhibiting)

21 - 22 November 2017

Warwick, UK

8th CSD&M Paris

12 - 13 December 2017

Paris, France

PPI AND CTI EVENTS

Systems Engineering Public 5-Day Courses

Upcoming locations include:

- Washington, DC, United States of America
- Melbourne, Australia
- Las Vegas, NV, United States of America

Requirements Analysis and Specification Writing Public Courses

Upcoming locations include:

- Singapore
- Pretoria, South Africa
- Amsterdam, the Netherlands

Systems Engineering Management Public 5-Day Courses

Upcoming locations include:

- Melbourne, Australia
- Singapore
- London, United Kingdom

Requirements, OCD and CONOPS Public 5-Day Courses

Upcoming locations include:

- Amsterdam, the Netherlands

Architectural Design 5-Day Course

Upcoming locations include:

- London, United Kingdom
- Stellenbosch, South Africa

Software Engineering 5-Day Courses

Upcoming locations include:

- Amsterdam, the Netherlands
- Pretoria, South Africa
- London, United Kingdom

Human Systems Integration Public 5-Day Courses

There are currently no scheduled courses, feel free to enquire regarding an in-house proposal.

CSEP Preparation 5-Day Courses (Presented by Certification Training International, a PPI company)

Upcoming locations include:

- Laurel, MD, United States of America
- Las Vegas, NV, United States of America
- Chantilly, VA, United States of America

Kind regards from the SyEN team:

Robert Halligan, Editor-in-Chief, email: rhalligan@ppi-int.com

Ralph Young, Editor, email: ryoung@ppi-int.com

Suja Joseph-Malherbe, Managing Editor, email: smalherbe@ppi-int.com

Project Performance International

2 Parkgate Drive, Ringwood North, Vic 3134 Australia Tel: +61 3 9876 7345 Fax: +61 3 9876 2664

Tel Brasil: +55 11 3958 8064

Tel UK: +44 20 3608 6754

Tel USA: +1 888 772 5174

Web: www.ppi-int.com

Email: contact@ppi-int.com

Copyright 2012-2017 Project Performance (Australia) Pty Ltd, trading as Project Performance International.

Tell us what you think of SyEN. Email us at syen@ppi-int.info.