From Substandard to Successful Software

Martin Allen
Independent Software Consultant

Our global finance industries are in the grip of a fearsome tempest known to us as the credit crunch, triggered by bad subprime mortgages and toxic debt. Is there a lesson here for our industries and government agencies that have become reliant on software-intensive systems of systems (SISoS) and are susceptible to the potential ravages of inferior software? It is essential for the software industry to identify and tackle what I call substandard software: software life-cycle products that do not have basic quality attributes such as reliability, usability, accuracy, efficiency, adaptability, and testability. This article provides indicators and professional advice in a set of seven rules that will increase the probability of a successful software project.

In 2007, a financial earthquake started shaking the stability of global economies, with the epicenter situated in the subprime mortgage market. Banks and investment institutions with hundreds of years of displaying prudence disclosed that their very foundations were riddled with toxic debt. From this, the credit crunch was born. The scale of the problem was enough to bankrupt countries such as Iceland[1], and behemoths in the banking industry continue collapsing or are subject to nationalization. Given the tens of thousands of financial experts employed and the maturity of risk management processes in the credit industry, one question remains: How did the problem become so large and so widespread without earlier detection?

Very few of us are qualified to understand why the circumstances for economic meltdown were not avoided, nor are the majority of our citizens equipped to comprehend even the basics of the global banking industry and the risks in the credit systems. Nonetheless, some of us ask whether there is a warning in the credit crunch for the acquirers and providers of high-technology systems. Could unmanageable levels of substandard software paralyze the software industry the way that unmanageable levels of subprime loans paralyzed finance?

Substandard Software

Recently, we have witnessed a significant rise in the number of organizations relying heavily on the successful deployment of SISoS. In industries such as defense, transport, medical, communications, energy, space, entertainment, and finance, reliance on software keeps growing. An exponential increase in the magnitude and complexity of the systems attempted is also evident. Undoubtedly, the risks from substandard software have increased respectively. For instance, the volume and complexity of software systems in the average family car have increased substantially in the last 10 years; however, it is clear that the vast majority of these software systems are dependable because otherwise our roads would be strewn with inoperable vehicles. On the other hand, there have been a few well-publicized hiccups with embedded software in the automotive industry, culminating in the embarrassing and expensive recall of thousands of vehicles [1]. A glut of substandard software is certainly capable of damaging the reputation of a car manufacturer, and is theoretically capable of contributing to the demise of automotive giants. Articles such as "Software's Chronic Crisis" [2] and various other reports suggest that multiple industries are afflicted by substandard software.

"Far too many substantial projects still flounder when involving software-intensive systems."

Far too many substantial projects still flounder when involving software-intensive systems. Project success in terms of cost, schedule, capability, and user acceptance is too rare an occurrence (refer to the sidebar on page 31). Furthermore, significant numbers of software systems that are successfully deployed are later found to have issues with maintainability, portability, scalability, flexibility, and reliability. In part, this is due to the fact that the first thing to be sacrificed in the rush to deploy a late project is quality; this provides a perfect incubator for substandard software. Professional engineers are taught to sacrifice non-essential system capabilities first and quality last. If the constraints of a schedule or budget mean an *all-singing-and-all-dancing* system cannot be deployed, then at least it may still satisfy customers and users temporarily.

Causal Factors

One favored approach for participants in failed or troubled software projects is to perform a forensic analysis or lessons-learned exercise. From experience, this tends to inappropriately concentrate an organization's attention on symptomatic rather than causal factors. By way of analogy, take the medical profession: For many decades, they strived to move from treating symptoms to curing disease to disease prevention; treatment of symptoms is often necessary, but far from sufficient.

It is easy to understand why a systems acquirer or supplier team struggles to overcome inherent bias in order to focus on and expose fundamental technical and managerial weaknesses. An alternative for them is to work diligently on eradicating symptomatic factors on succeeding programs, while ignoring conveniently the endemic causes of project problems. In the wake of an expensive project failure, there often follows an inglorious frenzy to reinvent software engineering processes from first principles, or the latest project management tool is rolled out in true panacea fashion. Thereby, unsuccessful project teams are provided with a myriad of opportunities to spread the spores of substandard software into other areas. Most diseases and infections rely on some interaction of their human hosts in order to spread and multiply; likewise, the spread of substandard software depends on people. In short, lessons-learned initiatives aimed at organizational improvement, based solely on failed or troubled projects, may be simply acting as a Trojan Horse[2].

The Seven Rules for Success

Analyzing and reporting the causal factors of failing software projects is often too onerous or uncomfortable an assignment for organizations. A complementary and more palatable strategy may be to concentrate on the fundamental factors that professionals know contribute to successful teams and projects. This is the transpose of a typical lessons-learned initiative. Could such an approach be influential in halting a substandard software plague[3]?

The following is a proposed set of seven rules to enhance the probability of success:

- **Rule 1:** Professionalism and software engineering competence should be assessed objectively and encouraged proactively by senior management.
- **Rule 2:** The number and seniority of software professionals employed within an organization should be commensurate with the magnitude and criticality of the required software systems.
- **Rule 3:** Good quality life-cycle products are the essential ingredients in the development, deployment, and maintenance of successful software systems.
- **Rule 4:** Mature industry standards are key to the production of high-quality, consistent life-cycle data products.
- **Rule 5:** Employ COTS software products with due care.
- **Rule 6:** The formality or weight of processes needs to be tailored and applied in accordance with project risks.
- **Rule 7:** Processes are necessary but not sufficient; competent people and practical life-cycle products are equally necessary.

In the following sections, the origins of these rules are considered in the context of the three elementary components of a project. I refer to these as the 3Ps: people, products, and processes.

People

In "The Mythical Man-Month," Frederick P. Brooks suggested that a tenfold productivity chasm existed between the most and least efficient software development teams [5]. This hypothesis was given further credence in Barry Boehm's "Software Engineering Economics" [6], and by subsequent studies. From assembly languages to Java, from structured methods to computer-aided software engineering tools, from process improvement to object orientation, each advance in software engineering has been fundamentally reliant on its application by competent managers and engineers (Rule 1). My extensive experience also helps confirm that a professional and competent staff, organized efficiently, is the primary influence on productivity.

Enterprises that depend on software systems should assess carefully and periodically whether the levels of education, training, experience, and seniority of personnel can accommodate the current and future needs of their industry. If anything is a causal factor in the success of projects, it is individual and team competence.

For some project teams, learning a best engineering practice is perhaps easier than abandoning systemic poor practices. When current project teams are found to be designing with flow charts, professionals must question why four decades of engineering progress are ignored. In "The Challenges of Complex IT Projects," professionalism and education are identified as having a major influence:

> A striking proportion of project difficulties stem from people in both customer and supplier organizations failing to implement known best practices. This can be ascribed to the general absence of collective professionalism in the IT industry, as well as inadequacies in the education and training of customer and supplier staff at all levels. [7]

Increasingly, there are calls for the competence of personnel working in safety-critical industries to be assessed and managed [8, 9]. At the time of writing this article, there were no legislative requirements in place in Europe to regulate the competence of software safety professionals.

What differentiates professionals and amateurs? Their behavior. Software professionals are characterized by a relevant education and continued learning; they have a holistic or system life-cycle focus; they work to industry standards; and they have a balanced approach to technical risk. In contrast, amateurs have no relevant software engineering education: They focus on implementation, coding, and tools, and have a naïve view of risk.

In some situations, organizations employing software professionals still find success elusive. This may be due to the pattern of seniority within the teams. In an environment where deployment of SISoS is essential to the business, software professionals should hold a commensurate level of senior roles (Rule 2). One possible alternative to the proliferation of process maturity models (e.g., CMMI) could be a framework for objective assessment of an organization's capability, based on the number and seniority of competent software engineers employed.

Along with the seniority of competent personnel, organizational structure is known to have a significant bearing on sustained success. The relative advantages and disadvantages of project-managed and discipline-managed structures have been acknowledged for many years. A hybrid matrix management structure is a reputable alternative that attempts to capture the advantages of both while reducing the disadvantages. Experience has shown that the matrix works effectively when the (software engineering) discipline is responsible for strategic decisions, while tactical decisions are best made by project teams. For example, the selection of a system architecture is a strategic concern. Organizational policy may be used to separate explicitly strategic and tactical concerns.

Products

If processes are viewed as the recipes for development of successful software systems, then life-cycle products are the vital *ingredients* (Rule 3). Good quality ingredients are essential in the creation of gastronomic delights, and there is a direct analogy here with software products. There is a finite limit to what can be achieved by processing or cooking with inferior ingredients.

Several mature software standards, particularly those originating from the DoD and IEEE, emphasize the production of data items (Rule 4). Software professionals recognized many years ago that the collection and management of cohesive, decoupled life-cycle data or information products is a crucial facet of the discipline. Tangible life-cycle data are required to support both verification and validation activities or processes.

Complementary ingredients are required to produce a decent meal, just as the availability of good quality code does not by itself satisfy the multiple criteria for a successful software system. Typical software system life-cycle products include:

- ConOps.
- System requirement specification.
- System architectural design.
- Software requirements specification.
- Interface requirement/design.
- Software architectural design.
- Source code.
- Executable code.
- A qualification test plan.
- A qualification test description.
- A development plan.
- A quality and configuration plan.

Superior standards tend to be prescriptive in the types of life-cycle data required, generic in the way the life-cycle data is collected and managed, and flexible enough to support tailoring for different categories of software projects. Effective tailoring is arranged from safety-critical, through mission- or business-critical, down to support software. Additionally, several standards provide evaluation criteria to direct the quality control of life-cycle products.

One proposed approach to the global software crisis—prevalent now in the defense communities—is to employ COTS software products (Rule 5). This remains a commendable strategy but is far from a panacea. There remain significant issues with the deployment of COTS-based applications into environments for which they were not intended (as indicated in [10]).

Processes

Software system processes and organizational process maturity have received a significant degree of academic and industry exposure in the 1990s and 2000s. The ubiquitous CMM and CMMI offerings from the SEI are familiar to software engineers around the globe. As stated previously in this article, processes are the essential recipes for the development of software systems.

However, it is a popular myth (cautioned in [10]) that processes alone can transform an enterprise from a Level 1 ad-hoc underachiever to a Level 5-optimized corporate machine. An established tenet is that overemphasis on processes and procedures makes an organization bureaucratic, inflexible, and overly reliant on checklists.

It has been suggested that the extraordinary interest in Agile methods has been fueled in part by a general dissatisfaction with heavyweight processes; the proponents of processes have fractured into two opposing entrenched camps: CMMI versus Agile. Quite simply, the weight or formality of processes applied should be commensurate with the project risks, especially with respect to magnitude and criticality (Rule 6). This is an established approach with safety-related standards such as DO-178B and IEC 61508. Such issues are also tackled in [10]. A measure of diligence is required because processes and techniques that are considered successful on small developments do not tend to scale-up. Software standards MIL-STD-498 and IEEE 12207, which are geared toward medium to large-scale SISoS development and integration, use project risk criteria as keys to the suitability of different life-cycle models. Grand design, incremental, and evolutionary models are included in these standards.

To all the proponents of software process—whether light or heavy, Agile or CMMI—a note of caution is appropriate. Stringent processes are very effective in volume manufacturing environments (from where the original concepts of process and quality control originated) where repeatability and consistency are gods. Stringent processes are also effective in software engineering. However, software engineering can never resemble a production line, and processes are necessary but not sufficient for success because competent people and practical life-cycle products are similarly indispensable (Rule 7).

Substandard Software Indicators

In 2007, I was involved in a large European collaboration project—with three international customers and one main system supplier—worth hundreds of millions of Euros. It was an environment that lent itself to creating substandard software. The main indicators were weaknesses in:

- **Standards Compliance.** The supplier organization had no demonstrable evidence of compliance on previous or current projects.
- **The Life-Cycle Model.** The supplier initially proposed the use of Agile methods, even though the application was large and safety-related [3].
- **Concept of Operations (ConOps).** After more than two years of project activity, the ConOps had not been considered.
- **Software and System Qualification Testing.** Contrary to all mature standards and test regimes, qualification testing did not feature in any project plans.
- **Data Modeling.** The system was predominantly a data management and distribution system, but no data model was available.
- **Competence.** The customers and supplier lacked competent software professionals in the project management teams.
- **System and Software Architecture.** Techniques to derive the system architecture were archaic and discredited (e.g., functional decomposition).

The lack of ability was a causal factor, whereas the other factors were symptomatic.

3Ps Aligned

Ensuring the alignment of the 3Ps by employing these seven rules is no guarantee, but it will maximize the probability of success. I have had the privilege of working on and witnessing several successful projects where competent people—analysts, designers, project managers, test professionals, and programmers—employed mature processes to produce dependable software products.

Commentators have remarked that many financial institutions will survive the present credit crunch, but will necessarily emerge with a different *modus operandi*, particularly with respect to acceptable risk. Perhaps too, our industries that are dependent on SISoS may have to go to the edge of a similar abyss before taking the opportunity to address the causal factors of substandard software. Alternatively, influential people and organizations could start applying the seven rules and may never need to explain why so many experts overlooked the substandard software epidemic.◆

References

1. Noon, Chris. “Okuda’s Toyota Recalls Prius Fleet Over Software Glitch.” Forbes. 14 Oct. 2005 <www.forbes.com/2005/10/14/toyota-prius-recall-cx_cn_1014autofacescan03. html>.
2. Wayt Gibbs, W. “Software’s Chronic Crisis.” Scientific American. Sept. 1994.
3. Boehm, Barry W., and Richard Turner. Balancing Agility and Discipline: A Guide for the Perplexed. New York: Addison-Wesley Longman, 2003.
4. Boehm, Barry W. A View of 20th and 21st Century Software Engineering. Proc. of the 28th International Conference on Software Engineering. Shanghai, China: 20-28 May, 2006.
5. Brooks, Frederick P. The Mythical Man-Month: Essays on Software Engineering. New York: Addison-Wesley Longman, 1974.
6. Boehm, Barry W. Software Engineering Economics. Upper Saddle River, NJ: Prentice Hall PTR, 1981.
7. The Royal Academy of Engineering and The British Computer Society. The Challenges of Complex IT Projects. London, UK: The Royal Academy of Engineering, Apr. 2004.
8. Health and Safety Executive, The Institution of Electrical Engineers, and The British Computer Society. Managing Competence for Safety-Related Systems. 2007 <www.hse.gov.uk/humanfactors/comah/mancomppt1.pdf>.
9. International Electrotechnical Commission: IEC 61508-1. Functional Safety of Electrical/Electronic/Pro-

Coming Events

June 29-July 2
ACM SIGMOD/PODS 2009
Providence, RI
www.sigmod09.org

July 1-3
21st International Conference on Software and Knowledge Engineering
Boston, MA
www.ksi.edu/seke/seke09.html

July 13-16
2009 International Conference on Software Engineering Theory and Practice
Orlando, FL
www.promoteresearch.org/2009/setp

July 19-23
International Symposium on Software Testing and Analysis
Chicago, IL
www.cse.msu.edu/issta09

July 20-24
33rd Annual IEEE International Computer Software and Applications Conference
Seattle, WA
http://conferences.computer.org/compsac/2009/

August 18-20
LandWarNet Conference
Ft. Lauderdale, FL
www.afcea.org/events/landwarnet/09

August 24-28
13th International Software Product Line Conference
San Francisco, CA
www.sei.cmu.edu/activities/splc2009/

2010
22nd Annual Systems and Software Technology Conference

SSTC Systems & Software Technology Conference

Salt Lake City, UT
www.sstc-online.org

Coming Events: Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.

grammable Electronic Safety-Related Systems. Appendix B. 1 Dec. 1998.
10. Boehm, Barry, Peter Kind, and Richard Turner. "Risky Business: 7 Myths About Software Engineering That Impact Defense Acquisitions." Project Management. May-June 2002 <www.dau.mil/pubs/pm/pmpdf02/boe-mj2.pdf>.

Notes

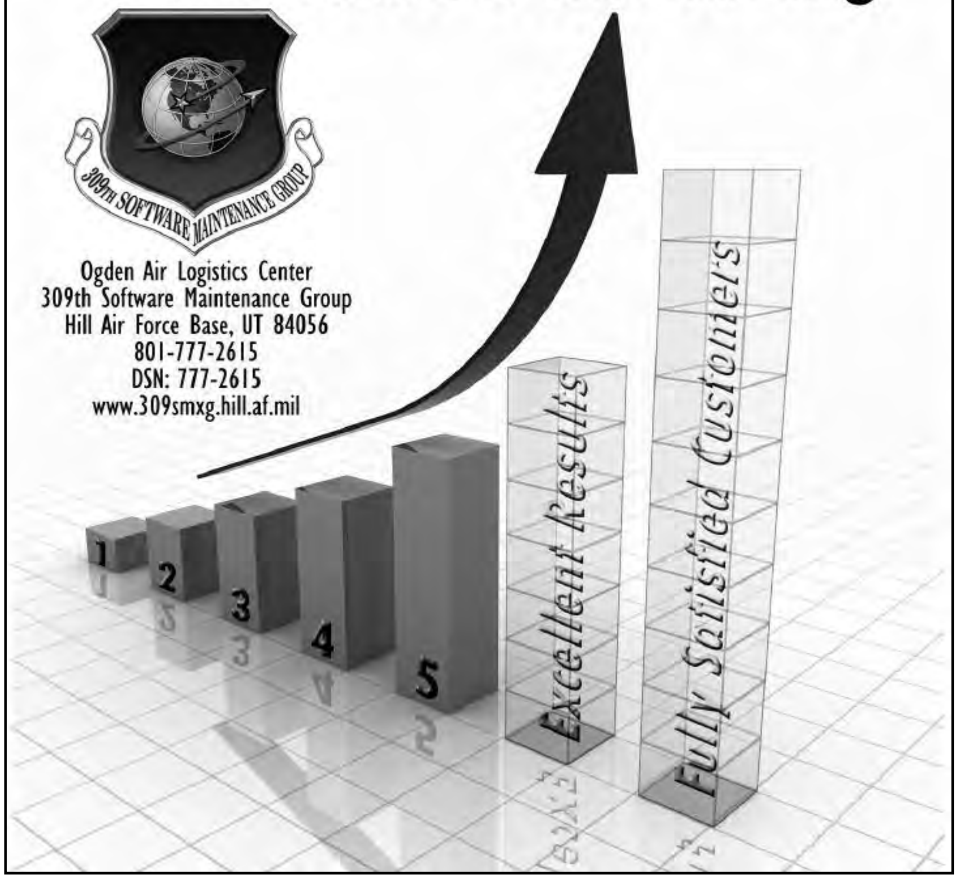
1. There are many articles on this topic, including <www.cnn.com/2009/ WORLD /europe/01/26/iceland.government>.
2. This is the more classic definition of "Trojan Horse" (see <http://en.wikipedia.com/wiki/Trojan_Horse>), not the computer malware.
3. Barry W. Boehm, the respected software engineering professor at the University of Southern California, hints at such a strategy in "A View of 20th and 21st Century Software Engineering." In a riposte to George Santayana's famous quote, "Those who cannot remember the past are condemned to repeat it," Boehm advises that failing to acknowledge and record past successes condemns an organization not to repeat them [4].

About the Author

Martin Allen is a software engineering professional with more than 28 years experience, mostly in the defense industry in the United Kingdom. He has worked on many successful software intensive systems for the British Royal Air Force and the Royal Navy. Allen has always had a strong interest in industry standards for the engineering of dependable systems. His other professional interests include risk management, software cost economics, requirements analysis, design methods, and software testing. Allen and his colleagues work on the boundary between the academic research of computer science and the practical application of software engineering.

**37 Priestfields
Fareham, Hampshire
United Kingdom, PO14 4TE
Phone: +0044 (0) 7881542031
E-mail: mjallen60@yahoo.co.uk**

CMMI Level 5 and Rising



Ogden Air Logistics Center
309th Software Maintenance Group
Hill Air Force Base, UT 84056
801-777-2615
DSN: 777-2615
www.309smxg.hill.af.mil