“If it passes test, it must be OK”

Common Misconceptions and The Immutable Laws of Software

Girish Seshagiri, Advanced Information Services Inc.

Abstract. As the saying goes, “If it passes test, it must be OK.” Common misconceptions about managing software inhibit changes to the way software projects are planned, audited and assured for cost, schedule, and quality performance. This article describes the immutable laws of software development as articulated by SEI Fellow Watts Humphrey and based on the author’s considerable professional experience in managing software technical teams. The author describes the impact of each of the immutable laws on an organization’s ability to deliver very high quality software solutions on a predictable cost and schedule. The author provides data from his company’s projects to illustrate many of the laws.

Introduction

After retiring from IBM, Watts Humphrey made an “outrageous commitment” to change the way software applications development services are acquired, sold and delivered. In addition to the CMM®, Watts was the principal architect of the Team Software Process (TSP) and the Personal Software Process (PSP) [1, 2, 3]. My company AIS was one of the early adopters of TSP and PSP. I was fortunate to work closely with Watts and built AIS’s software development business making quality the number one goal.

After listening to many of Watts’s presentations, and augmented by my personal experience in managing more than 200 software technical teams, I compiled a list of the immutable laws of software development. In this article, I discuss the implications of the laws and what acquirers, development management, and software teams need to be aware of to ensure consistent delivery of very high quality software systems and services on a predictable cost and schedule. I illustrate many of the laws with examples from AIS projects.

Software Engineering’s Persistent Problems

Software Engineering like other engineering professions has had a beneficial impact on society. Arguably, the high standard of living in today’s interconnected world is not possible without advances in software and software engineering. And yet there is ample evidence to suggest that software engineering as a profession has not been able to solve major persistent problems:

1. Exponential rise in cybersecurity vulnerabilities due to defective software
2. Unacceptable cost, schedule, and quality performance of Enterprise Resource Planning (ERP) and legacy systems modernization projects
3. Cost of finding and fixing software bugs (i.e. scrap and rework) as the number one cost driver in software projects
4. Arbitrary and unrealistic schedules leading to a culture of “deliver now, fix later”
5. Inability to scale software engineering methods even for medium size systems
6. Lack of understanding of the impact of variation in individual productivity
7. Absence of work place democracy and joy in work

Unless the software engineering professional community begins to systematically address these persistent problems, costs and risks to society will continue to increase [4].

The Immutable Laws of Software Development

Part of the reason for the persistent problems can be attributed to common misconceptions about managing the software work [5]. Organizations are either unaware of or are not willing to change practices to deal with the immutable laws.

I have listed some of the immutable laws and described the impact of each of the immutable laws on an organization’s ability to deliver very high quality software solutions on predictable cost and schedule. Where applicable, I have provided data from our company’s projects to illustrate many of the laws.

- **The number of development hours will be directly proportional to the size of the software product.**

While this is obvious, many projects do not estimate the size of the product before making a commitment for cost, and schedule. The implication of this law is that if an organization does not maintain a history of previous projects including the size of the product delivered and the effort in staff hours, the organization will make cost and schedule commitments with no relationship to the organization’s historic capability. The cost and schedule commitment will be a guess based on the organization’s desire to capture the business and not on what the organization can actually deliver. Which leads to the next law.

- **When acquirers and vendors both guess as to how long a project should take, the acquirers’ guess will always win.**

In the beginning, neither the customer nor the developer knows how big the project is or how long it should take and at what cost. As Watts used to point out tongue in cheek, customers want their product now at zero cost. Customers usually have to deal with time-to-market pressures and they require the product in time frames that are arbitrary and unrealistic for the software team to produce a product that works. The developers now have a choice to make. They can try to guess what it would take to win the business. Or as rational management would require, elicit enough of the project requirement to be able to make a conceptual design, estimate the size, and use organization historic data to predict development time and cost. The TSP institutionalizes this behavior in the TSP team launch process in which all the developers participate in estimating and planning the project. The result is that teams make realistic and aggressive commitment that the team can meet. The implication of this law is that when faced with arbitrary and unrealistic schedule pressures, developers should have the skills to make a plan before making the commitment and the conviction to defend it. Otherwise, the customers’ arbitrary and unrealistic schedule demand will become the team’s commitment. Management should trust the team to develop an aggressive and realistic schedule, and not commit teams to a date that the team cannot meet. This leads to the next law.

• **When management compresses schedule arbitrarily, the project will end up taking longer.**

It is unfortunate that otherwise rational managers do not realize that the defect potential in a project increases disproportionately to schedule compression as many studies have shown. In one study of data from a large number of projects, a 20% schedule compression had the effect of increasing defects during development by 66%. [6]. The logical reason is that when teams do not have the time to do the job right, they end up skipping the quality steps and try to meet an impossible schedule in a code and test mode which ends up taking longer. This leads to the next law.

• **When poor quality impacts schedule, schedule problems will end up as quality disasters.**

This is a classic pattern in major software project failures. For instance, in the case of Healthcare.gov, one can speculate that the contractor teams were working to meet a deadline they knew was impossible to meet. The teams probably did not employ the quality practices they knew they should use. (In fact, some of the contractors were appraised at high CMMI® Maturity levels.) Instead they probably went through increasingly long cycles of code, test, and rework. Because the amount of rework due to poor quality is unpredictable, the schedule problem gets progressively worse. The team was forced to deliver poor quality product on the committed date, thus turning the schedule problem into a world famous quality disaster. Healthcare.gov is not the first such spectacular software project failure, nor will it be the last, as seen in the next law.

This is also borne out by AIS's early history from 1988 – 1992. The company was not profitable because our projects were not predictable. The projects always seemed to be on schedule through code complete and before the start of integration, system, and acceptance tests. Due to the poor quality, teams spent significant amounts of time in test and rework. People worked long hours, and heroic efforts were needed to deliver on the committed date. The customer acceptance test phase was not a positive experience for either the customer or the team.

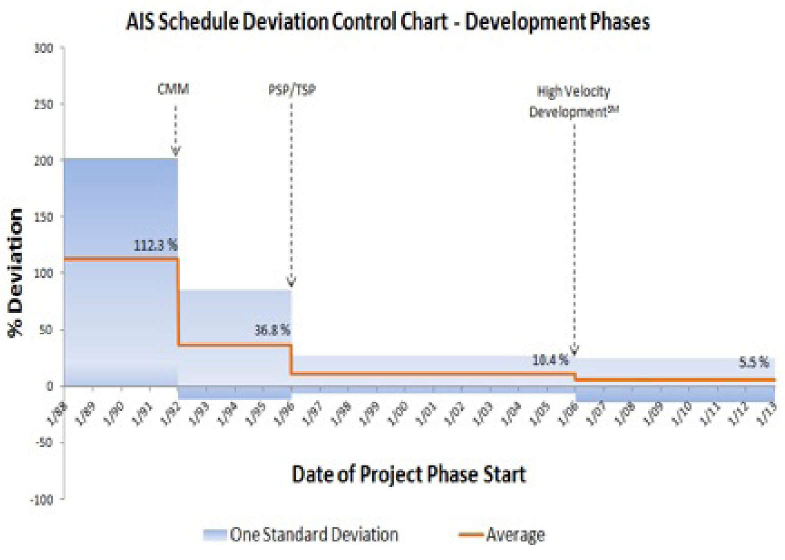


Figure 1: AID Schedule Deviation Control Chart – Development Phases

I realized that we had to change the way we managed the software work. What we needed was constancy of purpose with quality as the number one goal. Shown below is the schedule performance of AIS teams due to the improvement initiative I sponsored in 1992 based on the Capability Maturity Model (CMM) and later the TSP/PSP [7].

• **Those that do not learn from poor quality's adverse impact on schedule, are doomed to repeat it.**

The state of software practice will be much better for cost, schedule, and quality performance if only the c-level executives realize that poor quality performance is the root cause of most software cost and schedule problems. Remember SAM.gov, USAjobs.gov, and (ThriftSavingsPlan) TSP.Gov? These were noteworthy for cost and schedule overruns, the defects encountered in production and the long time it took to fix them, greatly inconveniencing the users of these applications. The government was doomed to repeat the experience in Healthcare.gov. This is not to single out government IT projects. Just that government projects get adverse publicity when they fail. It is probably not unreasonable to speculate that the commercial world is not immune to such quality disasters as documented in reports such as the Chaos report [8].

• **The less you know about a project during development, the more you will be forced to know later.**

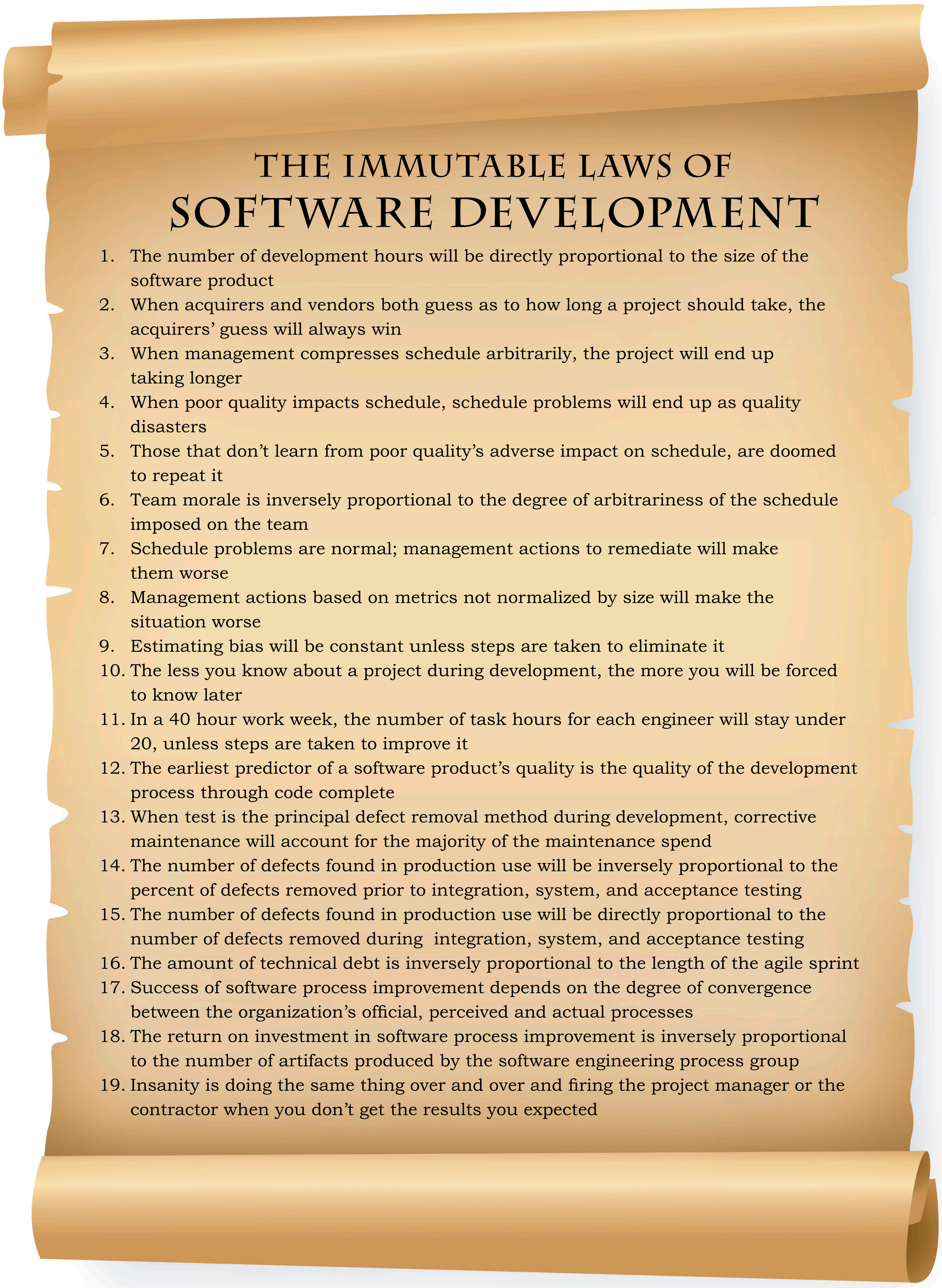
The implication of this law is that project teams need precise, accurate and timely information throughout development, to consistently deliver very high quality products on predictable cost and schedule. As Fred Brooks pointed out "Projects get to be one year late, one day at a time." When projects rely on the monthly status report as the only means of communicating what is happening in the project, they do not know enough to take timely corrective actions. When those projects fail, management relies on postmortems and audits to find out what went wrong.

In modernizing one of the largest databases in government, an AIS team collected and reported precise and accurate data in the weekly team status meeting. The team reviewed the project's documented goals weekly to make sure the team is on track to meet them. The team also reviewed the status of risk mitigation actions on the top 5 or 7 risks. The team made decisions weekly based on performance metrics that matter, including but not limited to plan vs. actual data on staff hours, earned value, defects injected, defects removed, and efficiency of early defect removal through personal reviews and inspections.

In many projects, one of the major causes for schedule slippage is because team members' actual hours on task are less than planned hours, which leads to the next law.

• **In a 40 hour work week, the number of task hours for each engineer will stay under 20, unless steps are taken to improve it.**

In estimating project schedules, teams typically do not consider the hours spent by team members on non-project tasks. In many organizations, the actual number of hours devoted to project tasks is on average less than 20 hours in a 40 hour work week. The implication of this law is that only management can take actions to improve the number of weekly task hours by providing improved office layout, minimizing



THE IMMUTABLE LAWS OF SOFTWARE DEVELOPMENT

1. The number of development hours will be directly proportional to the size of the software product
2. When acquirers and vendors both guess as to how long a project should take, the acquirers' guess will always win
3. When management compresses schedule arbitrarily, the project will end up taking longer
4. When poor quality impacts schedule, schedule problems will end up as quality disasters
5. Those that don't learn from poor quality's adverse impact on schedule, are doomed to repeat it
6. Team morale is inversely proportional to the degree of arbitrariness of the schedule imposed on the team
7. Schedule problems are normal; management actions to remediate will make them worse
8. Management actions based on metrics not normalized by size will make the situation worse
9. Estimating bias will be constant unless steps are taken to eliminate it
10. The less you know about a project during development, the more you will be forced to know later
11. In a 40 hour work week, the number of task hours for each engineer will stay under 20, unless steps are taken to improve it
12. The earliest predictor of a software product's quality is the quality of the development process through code complete
13. When test is the principal defect removal method during development, corrective maintenance will account for the majority of the maintenance spend
14. The number of defects found in production use will be inversely proportional to the percent of defects removed prior to integration, system, and acceptance testing
15. The number of defects found in production use will be directly proportional to the number of defects removed during integration, system, and acceptance testing
16. The amount of technical debt is inversely proportional to the length of the agile sprint
17. Success of software process improvement depends on the degree of convergence between the organization's official, perceived and actual processes
18. The return on investment in software process improvement is inversely proportional to the number of artifacts produced by the software engineering process group
19. Insanity is doing the same thing over and over and firing the project manager or the contractor when you don't get the results you expected

Figure 2: The Immutable Laws of Software Development

number of meetings etc. But the engineers have to record their time accurately including interruptions, to make management aware of low task hour utilization and the causes. In AIS projects, PSP trained engineers record time precisely and accurately and report task completions and earned value weekly.

- **The earliest predictor of a software product's quality is the quality of the development process through code complete.**

Software products are usually built from a large number of small components that are individually designed, coded, and tested. The PSP enables the engineers to build very high quality components through personal reviews and team inspections of the component's design and code artifacts. PSP trained engineers compile data on their personal process by recording size, time, and defect data on the components they build. By analyzing the component development process data, teams can determine the likelihood of the component having defects in downstream integration, system, and acceptance testing. The adverse impact on project schedule due to test and rework cycles in integration, system, and acceptance testing can be estimated before integration testing begins. AIS teams have a goal of more than 90% of the components to be error-free in integration, system, and acceptance testing. The impact of this law is that putting poor quality products into test will have adverse impact on the project's schedule and cost.

- **When test is the principal defect removal method during development, corrective maintenance will account for the majority of the maintenance spend.**

The implication of this and the following two laws is that putting poor quality product into test and relying solely on test for defect removal, has adverse cost implications beyond development. The biggest consequence is that as more defects are found in production use, organizations spend a very high percentage of the maintenance dollars in fixing bugs (i.e. corrective maintenance) instead of spending for the more beneficial enhancements and new features (i.e. perfective and adaptive maintenance). According to Watts, one of the software misconceptions is "if it passes test, it must be OK" [5].

- **The number of defects found in production use will be inversely proportional to the percent of defects removed prior to integration, system, and acceptance testing .**
- **The number of defects found in production use will be directly proportional to the number of defects removed during integration, system, and acceptance testing.**

The impact of these two laws is that early defect removal through personal reviews and team inspections, will result in high quality product (smaller percentage of defects remaining in the product) going into integration, system, and acceptance test which in turn will result in even higher quality product going into production. Conversely, putting a poor quality product (majority of defects remaining in the product) into integration, system, and acceptance test will result in excessive unplanned rework. What comes out of test will be a patched up product which in production use will uncover more defects to fix, thus consuming most of the maintenance dollars for fixing and keeping it running.

- **Success of software process improvement depends on the degree of convergence between the organization's official, perceived and actual processes.**

In every organization, there are usually three processes:

1. The official process, usually designed by the organization's software process engineering group, which describes the process the project teams should follow in their software projects.
2. The perceived process, which is what the software teams think how they do software work.
3. The actual process, which is how the teams actually work.

The implication of the law is that if the organization standard process is very different from the way the projects actually work, improving the standard process will be of little value. Project teams will continue to work the way they have in the past. In AIS, when we launched the continuous process improvement initiative, we first documented how the software teams were actually doing the software work. We used Watts Humphrey's Managing the Software Process book to establish a common vocabulary of process and process improvement. We empowered the engineers to make lots of small changes to the process by submitting simple but effective Process Improvement Proposals (PIPs). To-date AIS engineers have submitted more than 1400 PIPS of which more than 900 have been implemented. External SEI-authorized lead appraisers have appraised AIS's process maturity capability at CMMI Maturity Level 5 in 2007 and again in 2010 [7].

- **The return on investment in software process improvement is inversely proportional to the number of artifacts produced by the software engineering process group.**

The implication of this law is that if the process artifacts are produced by the software engineering process group and not the development teams, the artifacts may have little or no relationship to the actual work being done. The organization may pass maturity level appraisals without ever changing engineering behavior. Such organizations seldom produce very high quality products on predictable cost and schedule.

- **Insanity is doing the same thing over and over and firing the project manager or the contractor when you don't get the results you expected.**

This is a variation on the oft-used definition of insanity. The implication is that while people are extremely important, changing the people without changing the way the software work is managed is not likely to produce the expected results.

Conclusion

The relentless pressure to achieve a first-to-market advantage, has had the unfortunate side effect of developers more focused on meeting unrealistic schedule commitments than producing high quality software. We now have "deliver now, fix later" software development culture [9].

If the senior executives of software organizations understand the immutable laws and their impact, they will initiate the changes that are needed to consistently produce very high quality software on a predictable cost and schedule.

Disclaimer:

CMMI® and CMM® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University

ABOUT THE AUTHOR

Girish Seshagiri is a globally recognized subject matter expert and thought leader in software assurance, software quality management, software process improvement, and modern methods of managing knowledge work. He is a reputed conference speaker, coach, and instructor. He is the executive sponsor of AIS's continuous process improvement resulting in the company's receiving IEEE Computer Society Software Process Achievement Award and Capability Maturity Model Integration (CMMI) Maturity Level 5 certification. He is the author of the white paper "Emerging Cyber Threats Call for a Change in the 'Deliver Now, Fix Later' Culture of Software Development."

Girish has an MBA (Marketing), from Michigan State University.

E-mail: girish.seshagiri@advinfo.net
Phone: 703-426-2790

REFERENCES

1. Humphrey, Watts S. PSP: A Self-Improvement Process for Software Engineers. Addison-Wesley Pearson Education, 2005.
2. Humphrey, Watts S. TSP: Leading a Development Team. Addison-Wesley Pearson Education, 2006.
3. Humphrey, Watts S. TSP: Coaching a Development Team. Addison-Wesley Pearson Education, 2006.
4. Seshagiri, Girish "Is the Two-Week Agile Sprint, the Worst Software Idea Ever? - Management Issues in Software Assurance and Information Security." CSIAC Webinar, October 30, 2013.
5. Humphrey, Watts S. Managing the Software Process. Addison-Wesley, 1989.
6. Donald M. Beckett and Douglas T. Putnam. "Software Quality, Reliability, and Error Prediction." STN 13-1 (April 2010)
7. Seshagiri, Girish. "High Maturity Pays Off. It is hard to believe, unless you do it." CrossTalk (January/February 2012)
8. CHAOS Manifesto 2013: Think Big, Act Small. The Standish Group International Inc.
9. Seshagiri, Girish. "Emerging Cyber Threats Call for a Change in the 'Deliver Now, Fix Later' Culture of Software Development." White Paper (September 2013)