Synopsis --> PSP/TSP vs. Agile Methods

(Please Note: SW-CMM, CMMI, PSP, and TSP are registered service marks of Carnegie Mellon University ...)

**Article Objectives**

1. Discuss the merits of PSP/TSP vs. Agile Methods with enterprises who are exploring PSP/TSP and Agile Methods as "future, to-be" enterprise approaches.

2. Address enterprise groups who are responsible for "current, as-is" methodologies based on code inspections (who question the quality control practices of projects using Agile Methods).

3. Address PSP/TSP folks who are concerned about the quality control practices of Agile Methods (and convinced PSP/TSP are perfect and Agile Methods are a deliver fast, fix it later methodology).

4. Continue establishing the theoretical underpinnings of Traditional vs. Agile Methods.

5. Explore the key metrics and measurement paradigms of Traditional vs. Agile Methods.

6. Address the Lean/Kanban community who are favoring Traditional Methods, Metrics, and Measures (as key paradigms).

7. Embolden the Agile Methods community to continue fighting the good fight (because they are becoming lukewarm, embracing Traditional Methods more and more, considering Lean/Kanban which has Traditional overtones, and gun-shy about furthering the propagation/proliferation of Agile Methods).

**Positive Feedback**

1. Pentagon official in-charge of acquisition policy for the Intelligence Community.

2. European enterprise-level decision maker who decided not to pursue PSP/TSP in lieu of Lean-Agile Methods (after reading this article).

3. U.S. Air Force acquisition leader who accused Agile Methods of being the Anti-Christ last month (but has since capitulated after reading this article).

4. PSP/TSP certified instructor who's leading a Waterfall-based methodology for a major government agency (who agreed with my assessment of PSP/TSP).

5. Die-hard NASA systems engineering leader, who previously rejected all talk of Lean and Agile Methods in lieu of their well-entrenched CMMI culture (and is now willing to consider Agile Methods).

**PSP/TSP Beginnings**

PSP/TSP have an interesting back story, because they were conceived by Watts Humphrey in 1989 and formalized in the mid-1990s.

Watts Humphrey was an IBM physicist in the 1960s, who helped create today's trillion dollar commercial software industry by advising IBM lawyers not to sue independent software vendors writing software for IBM mainframes in 1970 (of course, the U.S. Department of Justice was about to breakup IBM into a hardware and software company, if they didn't stop suing independent software companies).

Watts Humphrey rose to become a vice president in charge of IBM mainframes in the 1970s and 1980s and then sold the idea of the SW-CMM (which became the CMMI) to the U.S. Air Force in 1986 upon his retirement from IBM.

(SW-CMM was originally conceived by IBM in the 1960s for quality control of shop-floor lathes, adapted to quality control of administrative personnel by Philip Crosby of ITT who coined the term "Zero Defects" in the 1970s, adapted for quality control of computer programmers by IBM-North Carolina in the early 1980s, and then sold to the U.S. Air ...
Force by Watts Humphrey in 1986 for U.S. DoD acquisitions, i.e., ironically humans are not lathes ...)

Watts Humphrey was very sensitive to criticism and people chastised him for creating an organizational software process improvement paradigm in SW-CMM that was irrelevant to individual computer programmers doing day-to-day work.

So, Watts Humphrey set out to create a software project management life cycle that could be applied to individual software engineers (vs. organizations as the SW-CMM and CMMI were intended).

**What is PSP?**

PSP is simply a collection of software engineering best practices created by IBM from the 1960s and 1970s for creating computer programs to minimize CPU time associated with testing by performing manual inspections of punch cards that would execute perfectly the first time (i.e., PROBE estimating, scheduling, EVM, Cleanroom-style box-structured specifications, desk checking/code Inspections, unit testing, etc.).

However, PSP itself is not a methodology, but a 9-stage training curriculum (i.e., computer programmers are given nine consecutive computer programming exercises to code mathematical algorithms starting with no software practices and then asked to use more and more software practices with every successive programming assignment).

The notion was that the individual computer programmers would see their productivity, quality, and schedule performance increase with every successive practice they added (and then psychologically buy into applying disciplined practices vs. programming by the seat-of-the-pants).

A fundamental practice is the notion of manual code inspections performed by individual computer programmers of their own code (i.e., what programmers called "desk checking" in the 1950s).

PSP can be summed up in one statement (i.e., "Find twice as many defects before testing using desk checking as by code-and-fix based unit testing and achieve zero defects in-use").

**What is TSP?**

TSP is an extension of the PSP to small teams of software engineers (and is a full-blown software lifecycle methodology rather than a training curriculum like PSP or a set of organizational guidelines like SW-CMM or CMMI).

Absolutely nothing in PSP/TSP is new, but is merely a codification of 35 to 45 year old IBM best-practices (however, Watts Humphrey "patented" PSP/TSP, because he thought they were innovative, and sued anyone who copied them or performed unlicensed PSP/TSP consulting).

Watts Humphrey attempted to sell PSP/TSP to the U.S. Air Force (in hopes it would be a mandatory standard for all U.S. Air Force software engineers creating avionics software for fighter jets and bombers).

**PSP/TSP Problems**

However, there were several problems with PSP/TSP that Watts Humphrey was never able to overcome:

1. Watts Humphrey patented, trademarked, and service marked PSP/TSP, was very protective of them, and sought litigation against anyone who overstepped the bounds of his personal control (effectively rendering them dead on arrival).

2. PSP/TSP textbooks were withheld from the marketplace for 10 years (and then published like crazy when PSP/TSP were long obsolete).

3. It cost one month and $25,000 to go through the PSP course, another $25,000 for TSP training, and yet more to become a licensed PSP/TSP trainer (to be a licensed/legal user).

4. The U.S. Air Force rejected PSP/TSP, because the training courses were too long, the exercises were too hard, they didn't use any practices created after 1975, and, of course, contractors created software (not govees).
5. U.S. Navy government software engineers began using PSP/TSP in the early 2000s to maintain assembly code stuffed into 30-year old avionics boxes (rather than the U.S. Air Force).

6. Agile methods swept Europe and swept the globe in 2002, long before U.S. Navy engineers became excited about PSP/TSP (and take it from personal experience, it was pretty difficult to get the Navy excited about very much).

7. Again, PSP/TSP didn't consist of any practices invented after 1975 (and agile methods could do everything PSP/TSP could do and far more for "zero startup cost" vs. the $50,000 a person PSP/TSP cost using a small set of obsolete techniques).

8. PSP/TSP are manually-intensive processes and research has shown up to 70% of measurement data is recorded incorrectly anyway (so reliance on the manually-collected metrics data is problematic).

9. The Software Engineering Institute itself never seemed to believe in Watts Humphrey or PSP/TSP, instead pouring hundreds of millions of taxpayer dollars into SW-CMM, CMMI, Product Line Management, Cyber Security, etc.

10. While a good "technical" orator and writer, Watts Humphrey was "not" a good marketer, and surrounded himself by math and statistics experts (rather than market-savvy people to communicate the business value of PSP/TSP).

11. Watts Humphrey relied on a tight circle of close friends rather than embracing and leveraging the worldwide community of programmers to promote and proliferate PSP/TSP, he kept PSP/TSP hidden under a rock, and he used legal action against anyone who attempted to modify or teach PSP/TSP outside of his immediate group.

**PSP/TSP vs. Agile Methods**

Let's do a quick comparison of PSP/TSP vs. Agile Methods:

1. A high-performing PSP-trained programmer can create 10,000 lines of code in 400 hours with zero defects with the $25,000 training as a prerequisite (assuming he doesn't get bored using 35 year old practices).

2. A high-performing Agile programmer can create 10,000 lines of code in 200 hours with zero defects and zero startup cost (and develop what the customer really needs without becoming a legal/licensed PSP/TSP user for $50,000).

3. A high-performing TSP-trained team can create 10,000 lines of code in 2,000 hours with zero defects with the $50,000 per head training as a prerequisite (assuming they can be successful with 35 year old practices and assumptions).

4. A medium-performing Agile team can create 10,000 lines of code in 1,000 hours with zero defects and no startup cost (and develop a profitable product that meets specific market/mission needs without becoming a legal/licensed PSP/TSP user for $50,000).

**PSP/TSP Assumptions**

Of course, PSP/TSP are based on some outdated assumptions:

1. Good methodologies should be proprietary in order to maximize their efficiency and effectiveness (which they shouldn't).

2. 100% of project/product scope and requirements can be known and specified in-advance (which they can't).

3. Productivity and quality do not break down with scale and size (which they do)

4. Perfect software can be built from perfect specifications without customer interaction, demonstrations, and involvement (which it can't).

5. PSP/TSP are scalable to any size problem (which they aren't).

6. Manual processes and inspections are superior practices (which they aren't).

7. Customers don't change their mind during the project or after the project is done (which they do).

8. All knowledge is explicit and can be captured in documentation (which it can't).

9. A disciplined process can be substituted for talented engineers (which it can't).
10. Most people are left-brained, analytical types that naturally succeed with data, facts, figures, math, and statistics (which they aren't).

11. Zero defects is the most important measure of market/business value, project success, and system quality (which it isn't).

12. Zero defects implicitly translate into secure software (which it doesn't).

13. Customer satisfaction is achieved by using in-process reviews of interim work-in-process/work products (i.e., requirements, designs, code, etc.), to bake-in quality over a long period of time until the product is just right long before the customer sees the final product (which it isn't).

**Agile Methods Assumptions**

Agile methods are based on some modern assumptions:

1. Good methodologies should be made available to the public domain (so they can be refined, perfected, improved, extended, and free for everyone to use).

2. Only about 1 to 3% of scope/requirements should be attempted (70% of requirements tacit, 20% wrong, and 7% not needed).

3. Direct customer interaction is needed to capture the 1% to 3% of valid, high-priority requirements that exist as tacit knowledge (which can't be done in a vacuum).

4. Requirements should be implemented and validated in small iterations (to continue eliciting tacit knowledge and requirements).

5. Small teams should work together to maximize exchange of tacit knowledge (which can't be captured explicitly in documentation no matter how hard you try, how many years take, how many people you use, or how much money you spend).

6. Plans, processes, documentation, and other project/technical data should be kept to a bare minimum and managed electronically (to minimize the overhead cost of creating wasteful documentation that is instantly obsolete after it is created and before the ink dries anyway).

7. Low-cost, flexible technologies should be used to minimize overhead expenses, maximize delivery speed and quality, and support rapid, iterative development, procurement, acquisition, and engineering.

8. Talented engineers are needed to succeed with disciplined processes.

9. Most people are right-brained, conceptual types who thrive on verbal communications, pictures, visions, motivation, empowerment, and making an impact on business and society.

10. Automate 90% or more of the process with free and open source technologies and perform 50 times more quality engineering tests than can be performed manually by placing the burden on machines to achieve zero defect software (rather than psychologically torturing programmers with monotonous/repetitive manually-intensive processes).

11. Software must be made secure on purpose, using deliberate software security engineering policies, staffing, skills, personnel, practices, tools, and measurements (i.e., it isn't an incidental or accidental side-effect or by-product of having zero defects).

12. Customer satisfaction is achieved by iteratively developing working products to obtain customer and market feedback as early and often as possible in order to rapidly refine the end-product that the customer and market actually needs rather than the interim work-in-process/work products (i.e., requirements, designs, code, etc.).

13. It's more important to focus on delivering a small set of customer needs that will improve profitability or mission effectiveness (than it is to have a large zero defect code base that nobody needs).

14. Profitability, business goal/objective satisfaction, mission effectiveness, capability validation, customer satisfaction, teamwork, process effectiveness and efficiency, and flexibility and adaptability to change are more important than "zero defects" (although validated requirements and low defects are measured and optimized as well).
Agile Phenomenon

Agile methods were the emancipation proclamation for computer programmers enslaved and held captive by IBM best practices from the 1950s, 1960s, and 1970s, and once freed from their intellectual slavery to obsolete mainframe computer programming practices, the pent up frustration of 25 million programmers was unleashed with a relentless fury from 1999 to 2002.

Today, there are over 200 textbooks, thousands of articles, and hundreds of free and open source tools in support of agile methods, their use swept the globe like a wildfire due to their efficiency, effectiveness, low startup and operational costs, and ease-of-use, and 80% of worldwide projects use them in every conceivable market sector, which dwarfed the size of what PSP/TSP were able to achieve by one or two orders of magnitude.

Miscellaneous Notes

1. I was susceptible to the PSP/TSP bug/virus/flu in the 1990s, because I took graduate classes in 1990 from an IBM systems engineer using the best practices that would become PSP/TSP on the Space Shuttle avionics software in the 1980s (when I worked on Space Station avionics).

2. I almost succeeded in getting a PSP/TSP pilot approved at a major gov’t agency in 2002 (but it was instantly killed by SW-CMM advocates).

3. My federal customer did pay over $50,000 to certify two govees as PSP trainers well over a decade ago (but everyone ignored them).

4. My first book published in 2004 was written in-part to convince my customer to invest in PSP/TSP (but it was ignored in favor of a marginally-effective $40 million CMMI initiative).

5. The SW-CMM/CMMI crowd never accepted PSP/TSP, because they believed SW-CMM/CMMI is all that is needed (and that belief is still pervasive among die-hard CMMI advocates).

6. There are still some people who believe that a 35 year old methodology stitched together from IBM best practices based on manual, individual code inspections to reduce the scarce mainframe CPU time associated with testing is superior to 100,000+ automated tests performed every in 10 minutes (using cheap/powerful commodity PCs and free and open source software).

7. Lockheed built the U-2, SR-71, and F-117A from practices that resembled agile methods rather than PSP/TSP (and the U-2 is still in-use today and preferred over UAVs as a multi-purpose ISR platform).

Summary

PSP/TSP are effectively dead and there is little to be gained from a 35 year old methodology that costs $50,000 per individual to learn, is based on manual code inspections, and assumes 100% of customer requirements can be captured in a vacuum and delivered without customer or developer interaction.

Watts Humphrey did win the U.S. National Medal of Technology for fathering SW-CMM, which became CMMI, although U.S. and worldwide firms spent billions of dollars applying them yielding only marginal benefits, as over 70% of SW-CMM/CMMI initiatives failed along with investments in them.

The time for PSP/TSP has passed away, and with them the hope of a small set of IBM best practices from the 1960s optimized to minimize CPU time with manual code inspections as the solution to 21st century global scale, enterprise, and market/business-driven solutions that agile methods were intentionally designed to address and have proven to do quite well over and over again.

Today, the Software Engineering Institute is leading the charge and waving the flag of agile methods, rumors are flying that CMMI is dead, and the SEI is pushing the boundaries of agile research and exploring its application on large-scale, mission and safety-critical systems in a show of support they never exhibited for PSP/TSP, probably because PSP/TSP were never openly shared with other researchers as much as they could have been.

(Please Note: SW-CMM, CMMI, PSP, and TSP are registered service marks of Carnegie Mellon University ...