

PEOPLE CMM EN TEAM CMM,

nieuwe loten aan de CMM stam

Fred J. Heemstra

Rob J. Kusters

Contactadres: fred.heemstra@ou.nl

Samenvatting

Toen ik zo'n zeven à acht jaar geleden aan een groep postdoctorale IT studenten vroeg wie bekend was met het Capability Maturity Model (CMM) waren dat er op een aantal van twintig, hooguit drie. Stel ik die vraag nu weer dan liggen de verhoudingen precies andersom. Bijna iedereen in de software wereld is bekend met het CMM en een behoorlijk aantal studenten heeft er ervaring mee. Stel ik diezelfde vraag maar nu naar de bekendheid van de CMM derivaten PSP (People Software Process) en TSP (Team Software Process) dan lijkt de geschiedenis zich te herhalen. Reden genoeg om eens stil te staan bij deze twee nieuwe producten uit de CMM-school. Die reden wordt versterkt door de vele reacties op het artikel 'Softwarekwaliteit, aandacht voor niet-technische factoren' in het oktober nummer van Informatie vorig jaar. Veel reacties en vragen richtten zich op de mogelijkheden en onmogelijkheden van PSP en TSP.

In deze bijdrage willen we in hoofdlijnen aangeven wat beide CMM producten inhouden, wat je er als organisatie mee kunt c.q. mag van verwachten en ervaringen tot nu beschikbaar zijn.

1 Waarom PSP en TSP?

Reifer laat zien hoe anno 2000 aangekeken wordt tegen organisaties die software maken en leveren. Dat is een beeld om niet blij van te worden.

Tabel 1. Software Management problemen

Vraag	ja (%)	nee (%)
1. Worden de organisaties die software maken en/of leveren goed gemanaged?	18	82
2. Leveren deze organisaties wat zij beloven?	34	66
3. Zien klanten/gebruikers de geleverde/gemaakte software als een kwaliteitsproduct?	25	75
4. Wordt bij het maken van software klassieke management tools en technieken gebruikt?	55	45
5. Levert de software een bijdrage aan de primaire taak van de organisatie?	30	70
6. Worden software engineers als professionals beschouwd?	85	15

Reifer constateert dat de afgelopen tien tot twintig jaren geleden weinig is veranderd; een pijnlijke constatering. Software organisaties hadden en hebben weinig tijd om hun werk goed uit te voeren, gewekte verwachtingen wat betreft kosten en levertijd waren en zijn weinig realistisch en software ontwikkelaars zijn vaak niet te volgen. De enquêteresultaten van Reifer kunnen eenvoudig met praktijkcijfers worden onderbouwd. Het Software Engineering Institute (SEI) is het meest overtuigend over de huidige prestaties van software ontwikkeling en kan zijn uitspraken bovendien baseren op veel empirisch materiaal. Het SEI heeft namelijk op basis van zogenoemde 'best practices' het Capability Maturity Model (CMM) ontwikkeld. In het CMM worden vijf niveaus van volwassenheid onderscheiden waarop een software organisaties zich kan bevinden. Op het laagste niveau is absoluut geen sprake van software management. Naarmate een software organisatie meer volwassen wordt doorloopt hij de verschillende niveaus van het CMM. De software organisatie krijgt gaandeweg steeds meer

inzicht in haar processen en is steeds beter in staat voorspelbare kwaliteit op tijd en binnen budget te maken.

Uit de talloze CMM evaluaties komt steeds weer hetzelfde beeld naar voren. Tussen de 70 en 80% van de organisaties die zich bezighouden met softwareontwikkeling bevindt zich op het laagste niveau van het CMM (zie tabel 2).

Tabel 2: Verdeling van 58 geëvalueerde organisaties over de vijf CMM niveaus.

CMM niveau	Verdeling in 1991	Verdeling in 1994
Primitief niveau	81%	73%
Herhaalbaar niveau	12%	16%
Gedefinieerd niveau	7%	10%
Gemanaged niveau	0%	0,6%
Optimaal niveau	0%	0,3%

Het realiseren van software binnen de afgesproken tijd en budget en met de juiste kwaliteit is meer een kwestie van geluk dan wijsheid is. Het succes van een project is te danken aan de kwaliteiten van de projectleden. Deze organisaties schieten te kort op de meest elementaire management activiteiten zoals:

- software projectplanning
- software projectbeheersing
- software configuratie management
- software subcontracting management
- software kwaliteitsmanagement
- software requirements management

Dit alles roept onherroepelijk vragen vraag op als ‘hoe is het zover kunnen komen’ en natuurlijk: 'wat valt er aan te doen'. Op de eerste vraag zijn talloze antwoorden te geven; meestal gaat het om een combinatie van oorzaken zoals de onvolwassenheid van het vakgebied, de dynamiek van de softwaremarkt, de complexiteit van de op te lossen problemen, en de specifieke kenmerken van software. Een achterliggende oorzaak heeft alles te maken met een gebrek aan professionaliteit en een onderschatting van zaken als gedrag, motivatie, commitment, opleiding, teamsamenstelling, teambuilding op de kwaliteit van het werk van de software ontwikkelaar. En dat is precies de reden waarom benaderingen als het PSP en het TSP zijn ontstaan. In de volgende paragraaf geven we de essentie van beide benaderingen weer, laten we zien wat toepassing ervan oplevert en geven we enkele kritische kanttekeningen.

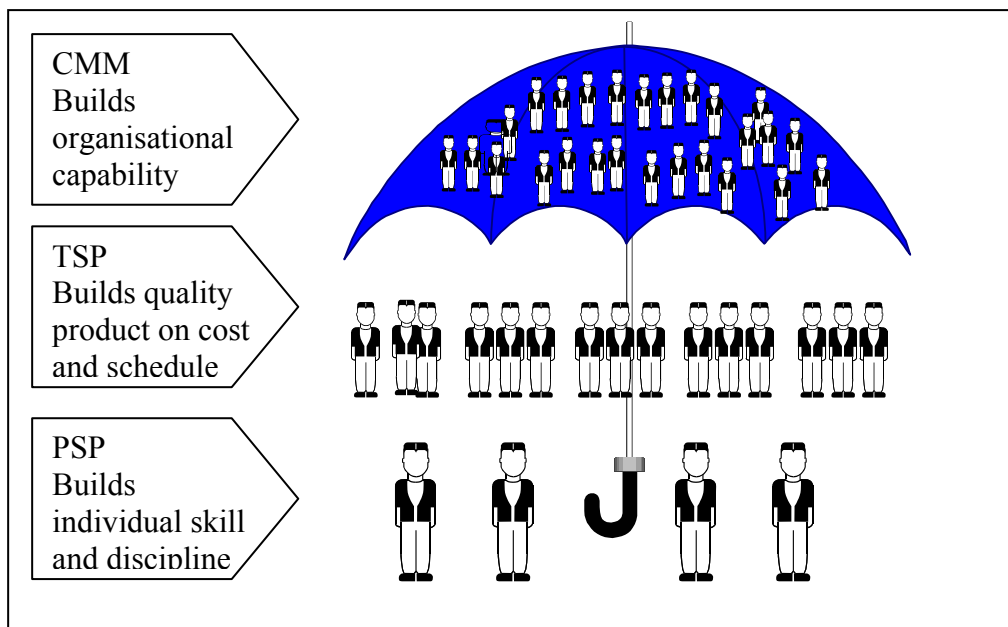
2. PSP en TSP op hoofdlijnen

2.1. CMM, PSP en TSP: samenhang

Net als het CMM zijn de PSP en TSP modellen afkomstig uit de koker van het Software Engineering Institute c.q. Watts Humphrey. In de visie van Humphrey kan een softwareontwikkelaarorganisatie alleen maar maximaal presteren als er wordt gewerkt met de best mogelijke teams die zijn samengesteld met de best opgeleide ontwikkelaars. CMM, PSP en TSP zijn ontwikkeld om deze visie te concretiseren. Humphrey illustreert de samenhang tussen de drie modellen aan de hand van figuur 1.

Het CMM biedt handvatten / schrijft activiteiten voor die uitgevoerd dienen te worden om procesverbetering op **organisatieniveau** te realiseren. Het CMM verschaft dus mogelijkheden

om het werk goed uit te voeren. Het CMM garandeert dat evenwel niet. Humphrey geeft zelf toe dat een van de nadelen van het CMM is dat het alleen maar aangeeft ‘Wat’ een organisatie moet doen en niet ‘Hoe’ een organisatie verbeteractiviteiten moet uitvoeren. Om de mogelijkheden van het CMM optimaal te kunnen benutten, moet het proces d.w.z. de activiteiten die het CMM benoemt, uitgevoerd worden door een ontwikkelaar die specifieke kennis, vaardigheden en een gedisciplineerde werkwijze in huis heeft. Op dit punt komt PSP in beeld. PSP gaat over het hanteren van de principes van procesverbetering op het **individuele** niveau van de ontwikkelaar met als doel sneller en goedkoper, betere software te maken. Om alles uit PSP te halen, moet de ontwikkelaar werken in een omgeving die hem die mogelijkheden biedt. Vandaar dat PSP het hoogste rendement heeft in softwareorganisaties die zich in de buurt van of boven het CMM niveau twee bevinden. Het TSP tenslotte is gericht op **teamniveau** en geeft aan hoe een effectief team gemaakt dient te worden en hoe een PSP opgeleide ontwikkelaar hierin als effectief teamlid kan functioneren. De combinatie van die modellen zorgt ervoor dat software organisaties in staat hoogwaardige kwaliteitsproducten op tijd en binnen budget te maken, aldus Humphrey.



Figuur 1: De samenhang tussen het CMM, het PSP- en het TSP-model

Het CMM, PSP en TSP bevruchten elkaar wederzijds. Het CMM en TSP bieden die gestructureerde en geordende omgeving die ontwikkelaars nodig hebben om hun werk zo goed mogelijk te doen. PSP-opgeleide ontwikkelaars kunnen het geleerde alleen maar optimaal gebruiken als zij kunnen werken in een daarvoor aangemeten CMM omgeving en binnen een optimaal ingericht TSP team.

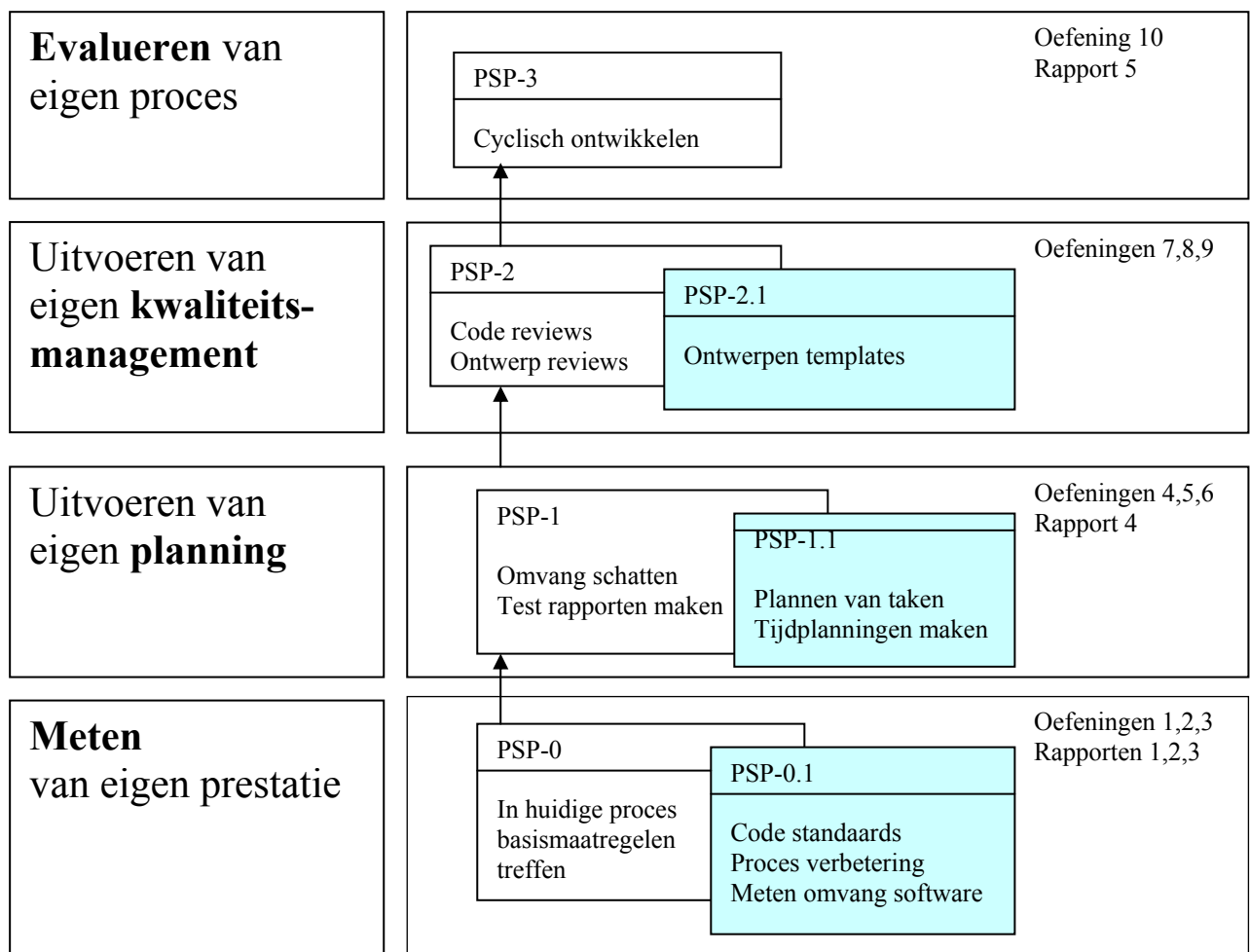
2.2. PSP in het kort

Het PSP-hanboek geeft aan dat PSP gebaseerd is op de volgende (elementaire) zaken:

- iedere ontwikkelaar is weer anders. Dat betekent dat een ontwikkelaar – om zo effectief mogelijk te presteren – zijn eigen werk moet plannen en dat die plannen gebaseerd moeten zijn op zijn eigen persoonlijke gegevens,

- Om zijn eigen prestaties op een consistente manier te verbeteren moet een ontwikkelaar gebruik maken van goed gedefinieerde en meetbare processen die door hemzelf op maat zijn gemaakt,
- Om hoogwaardige software te kunnen maken, moet een ontwikkelaar zich persoonlijk verantwoordelijk voelen voor de resultaten van zijn werk. Goede software ‘vallen niet uit de lucht’ maar is het resultaat van kwalitatief hoogwaardige arbeid,
- Het is efficiënter om fouten te voorkomen dan ze op te sporen en te herstellen,
- Als er al fouten zijn dan is het het goedkoopst deze vroeg zo vroeg mogelijk in het proces op te sporen en te herstellen,
- De juiste manier om een klus te klaren is altijd de snelste en de goedkoopste.

Om een klus op de juiste wijze uit te voeren moet een ontwikkelaar dus zijn eigen werk plannen voordat hij aan de slag gaat en moet hij gebruik maken van een uitgekristalliseerd proces om het werk überhaupt te kunnen plannen. Om inzicht te krijgen in zijn eigen capaciteiten en prestatieniveau, moet hij de tijd die hij nodig heeft voor elke stap van zijn werk meten, moet hij bijhouden welke fouten hij veroorzaakt en oplost, en moet hij de omvang van de software die hij maakt meten. Om blijvend goede software te blijven maken, moet een ontwikkelaar plannen, meten, kwaliteit bewaken en vanaf de start van de klus kwaliteit centraal stellen. Tenslotte moet hij de resultaten van elke klus analyseren en de analyse resultaten gebruiken om zijn proces te verbeteren.



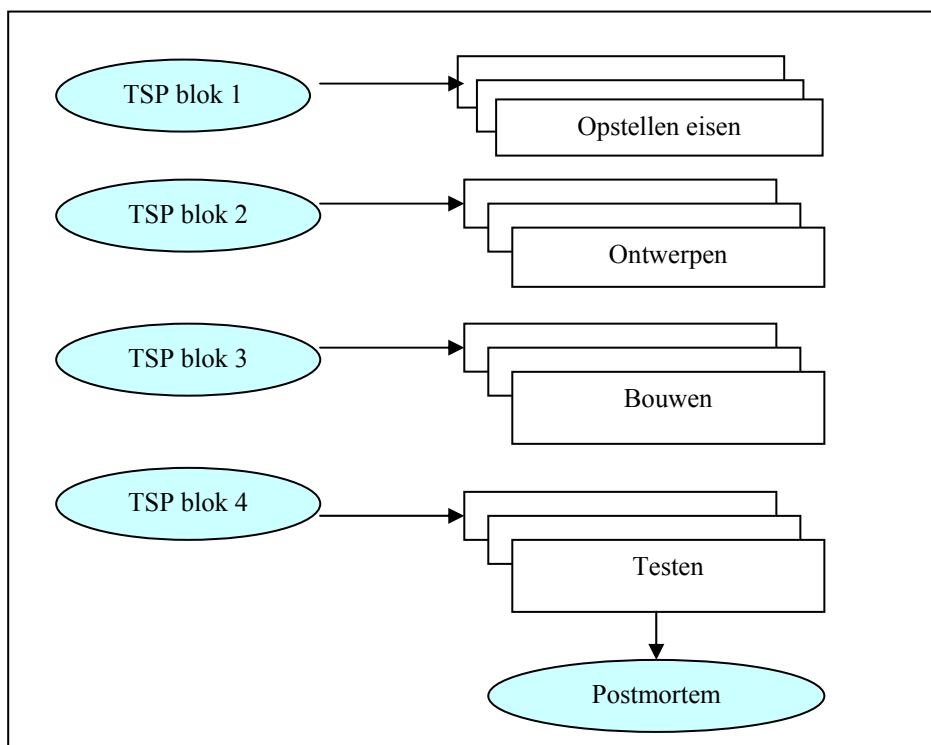
Figuur 2: De structuur van het PSP leertraject

Om zich deze manier van werken eigen te maken, moet een ontwikkelaar een zogenoemd PSP trainingsprogramma doorlopen. Zoals weergegeven in figuur 2, bestaat de training uit 4 blokken PSP0 tot en met PSP3. In ieder blok wordt het gebruik van projectmanagement-, procesmanagement- en kwaliteitsmanagement methoden geoefend. Tijdens de training moeten tien programmeeropdrachten en 4 verslagen worden gemaakt. Gaande van blok PSP0 naar PSP3 neemt de moeilijkheidsgraad toe en evolueert de ontwikkelaar naar een 'mature developer'. Een centraal thema binnen PSP is het meten van de eigen prestatie. Na iedere opdracht dient de ontwikkelaar het effect van het geleerde te meten en te evalueren. Omdat hij verplicht wordt gegevens te verzamelen en te analyseren, krijgt hij inzicht op welke wijze een gedefinieerd en meetbaar proces hem helpt om zijn werk beter uit te voeren. Voor een gedetailleerde beschrijving van het PSP leertraject zijn bijzonder goede documenten vrij te downloaden van de uitstekende website van het SEI (www.sei.cmu.edu).

2.2 TSP in het kort

In de praktijk blijkt dat teams veel tijd en energie verspillen aan het worstelen met vragen als "wat zijn onze doelen, wie is waarvoor verantwoordelijk, welke kwaliteit moeten we realiseren, hoe zetten we ons ontwikkelplan op, hoe rapporteren we aan de klant en onze baas, etc.?" Het vervelende is dat dit soort vragen al honderden keren zijn gesteld en uitstekende antwoorden al lang beschikbaar zijn. TSP geeft richtlijnen hoe een software ontwikkelaar ontwikkel- en onderhoudsactiviteiten in teamverband moet uitvoeren. Het model is een uitbreiding en verfijning van het CMM en PSP en. TSP heeft de volgende doelstellingen:

- Het realiseren van zelfsturende teams die hun eigen werk plannen en bewaken, hun eigen doelstellingen bepalen en verantwoordelijk zijn voor hun eigen processen en plannen.
- Managers inzicht geven hoe zij hun team moeten coachen en motiveren en hoe zij hun team moeten ondersteunen om maximaal te presteren.
- Versneld realiseren van 'Software Process Improvement' (SPI) door ervoor te zorgen dat de werkhouding en -invulling die behoort bij een CMM niveau 5 organisatie als normaal en algemeen geaccepteerd wordt beschouwd.
- Het geven van verbeterrichtlijnen om een organisatie op een hoog CMM niveau te krijgen.
- Universitair onderwijs faciliteren in het bijbrengen van teamvaardigheden.



Figuur 3. De TSP structuur

Als een team besluit volgens een TSP aanpak een project uit te voeren, wordt elke projectfase begonnen met een TSP blok. In figuur 3 wordt de TSP-structuur in beeld gebracht. In zo'n blok krijgt het team richtlijnen aangereikt op welke wijze het team (volgens de TSP principes) de betreffende fase moet plannen en organiseren.

Zo krijgt het team bij aanvang van het project bijvoorbeeld de volgende richtlijnen / opdrachten mee om uit te voeren:

- Neem de doelen van het project door met het management,
- Leg de rollen van de teamleden vast,
- Zorg dat je het als team eens wordt over de doelen van het team en leg deze vast,
- Zorg dat er volgens een ontwikkelstrategie wordt gewerkt die iedereen kent en iedereen accepteert,
- Definieer het ontwikkelproces waarmee het team aan de slag gaat,
- Maak een planning voor de middelen die nodig zijn om het team te ondersteunen,
- Stel een kwaliteitsplan op en bepaal kwaliteitsdoelen,
- Maak voor elke ontwikkelaar gedetailleerde plannen voor de volgende fase,
- Voeg die individuele plannen samen tot een teamplan,
- Breng de werkbelasting voor het team nu goed in balans,
- Beoordeel de projectrisico's en geef aan wie verantwoordelijk is voor het aanpakken van welk risico,
- Sluit dit TSP blok af met een evaluatie van het blok.

PSP-opgeleide ontwikkelaars hebben voldoende aan een vierdaagse workshop voor blok 1 (de start up); de start van elke volgende projectfase begint met een tweedaagse TSP-workshop. Als het project eenmaal loopt, belegt het team – als onderdeel van de TSP aanpak – wekelijks een teamoverleg en rapporteert het team periodiek aan het management en de opdrachtgever. Voor een uitgebreide beschrijving verwijzen we wederom naar de eerder genoemde website van het SEI.

3. Succes stories

Humphrey en de zijnen beweren dat het gebruik van PSP en TSP moet leiden tot:

- Betere schattingen van:
 - de omvang van de software,
 - de benodigde inspanning om de software te maken,
 - de benodigde tijd om de software te realiseren
- Betere software (minder fouten per 1000 regels code (KLOC)),
- Een beter uitvoerbaar proces (minder testtijd en minder testen nodig),
- Een hogere productiviteit (meer regels code per uur),
- Het verzamelen en analyseren van meer en betere procesgegevens (meer inzicht in datgene waarmee men bezig is),
- Meer vertrouwen in het eigen team.

Beide benaderingen zijn echter nog nieuw, veel ervaringscijfers zijn nauwelijks beschikbaar en de uitspraken van Humphrey zijn vooralsnog gebaseerd op een beperkt aantal positieve toepassingen. In deze paragraaf zullen we enkele bekende succesverhalen de revue laten passeren.

3.1 Resultaten van PSP gebruik.

PSP wordt nog niet, zoals het CMM, op grote schaal toegepast. We zien echter dat het aantal organisaties dat overgaat tot toepassing van PSP geleidelijk toeneemt (bijvoorbeeld Baan, Boeing, Motorola, Honeywell en Teradyne). Evaluatiegegevens wijzen op de voordelen van het PSP-trainingsprogramma.

Bij het software bedrijf Advanced Information Services (AIS) kreeg een ontwikkelteam halverwege de uitvoering van een het ontwikkelproject een PSP-opleiding. Voor de opleiding gaven de ontwikkelaars tijdschattingen af die bijna 400% afweken van de werkelijke ontwikkeltijd. Na de PSP-opleiding bleken de ontwikkelaars veel realistischer schattingen te kunnen afgeven, namelijk schattingen met slechts een afwijking van 10% van de werkelijk bestede tijd. Ook de verschillen in benodigde testtijd voor en na de PSP-opleiding bleken aanzienlijk te zijn; in plaats van enkele maanden kon na de opleiding worden volstaan met testtijden van enkele dagen (voor componenten van ongeveer dezelfde omvang).

Bij Honeywell nam het aantal fouten dat bij testen werd gevonden als gevolg van een PSP-training met een factor twee af, wist men de tijd om een fout te herstellen terug te brengen van 9 naar 5,5 minuten en dat alles bij een min of meer gelijkblijvende productiviteit.

Op verschillende Amerikaanse universiteiten wordt al vanaf 1993 onder studenten en software ontwikkelaars geëxperimenteerd met de PSP-benadering. Zowel studenten als ontwikkelaars presteren beduidend beter na een PSP-opleiding; twee tot drie keer minder fouten per 1000 regels code en productiviteitsverbeteringen van gemiddeld 35%. Wat opviel was dat de grootste verbeteringen werden gerealiseerd door de meest ervaren studenten / ontwikkelaars.

Het Software Engineering Institute (SEI) heeft de PSP-opleidingsgegevens van 298 ontwikkelaars geanalyseerd. In tabel 3 worden de belangrijkste resultaten weergegeven.

Tabel 3: SEI-resultaten van de PSP opleiding (van Buren e.a., 1999).

Beoordelingsaspect	Verbetering
Schatting van inspanning	Verbetering met een factor 1,75
Schatting van omvang	Verbetering met een factor 2,5
Product kwaliteit	Verbetering met een factor 2,5
Proces kwaliteit	Verbetering met 50%
Productiviteit	Geen wijzigingen

3.2 Resultaten van TSP gebruik.

TSP is pas sinds 1999 op de markt en ervaringen met de toepassing ervan zijn nog maar beperkt. De beperkte evaluatieresultaten zijn echter hoopgevend. In tabel 4 worden de resultaten van vier experimenten samengevat.

Tabel 4: resultaten van het gebruik van TSP

Beoordelingsaspect	Zonder TSP	Met TSP
1. <i>Schatting van inspanning</i> (afwijking schatting en werkelijkheid) - percentage afwijking	16%	-4%

- range	-60% tot +100%	-36% tot +26%
2. <i>Schatting van tijd</i> (afwijking schatting van werkelijkheid) - percentage afwijking - range	33% -33% tot +157%	6% -8% tot +23%
3. <i>Betere software</i> (aantal fouten dat tijdens ontwikkeling wordt gevonden) - aantal fouten per 1000 regels code	1 tot 8	0,1 tot 1,1
4. <i>Betere software</i> (aantal fouten dat tijdens acceptatietest wordt gevonden) - aantal fouten per 1000 regels code	0,2 tot 1,9	0 tot 0,35
5. <i>Beter proces</i> (hoeveelheid tijd nodig om te testen) - aantal dagen per 1000 regels code	1 tot 7.7	0,1 tot 1.1

Gebruik van TSP bij het bedrijf Teradyne Inc. leidde tot significant betere schattingen van omvang, tijd en inspanning (schattingsafwijkingen van minder dan 8% van de werkelijkheid), betere softwarekwaliteit (verbeteringen met een factor 5).

Een TSP toepassing bij de 'Technology and Industrial Support (TIS)' van Hill Air Force Base resulteerde in minder benodigde testtijd; zonder TSP ging gemiddeld 22% van de ontwikkeltijd zitten in testen, bij gebruik van TSP was dan nog maar 2,7%. Tegelijkertijd nam de productiviteit met 123% toe en bleek het aantal fouten per 1000 regels code aanzienlijk af te nemen.

Een aantal projecten bij Boeing, uitgevoerd volgens de TSP-richtlijnen, liet zien dat de software kwaliteit, gemeten in het aantal fouten per 1000 regels code, met een factor 9,5 toenam. Bovendien wist men de testtijd met 94% terug te brengen en de doorlooptijd van softwareontwikkeling te reduceren.

Het software bedrijf AIS (Advanced Information Services) wist de spreiding in haar schattingen van 200% terug te brengen naar 25%, het aantal fouten in de software sterk te verminderen en de benodigde testtijd aanzienlijk in te korten.

Bij experimenten met studenten op de Embry-Riddle Aeronautical Universiteit bleken TSP-teams meer dan 99% van alle ontwikkelfouten uit de software te verwijderen voordat tot de systeemtest werd overgegaan.

4. Enkele kritische kanttekeningen

Hoewel de genoemde toepassingen van PSP en TSP soms tot enorme verbeteringen lijken te leiden, zijn enkele kritische opmerkingen op zijn plaats. De eerder genoemde hosanna verhalen zijn immers afkomstig van evaluaties van het SEI, dwz van de ontwikkelaars van beide benaderingen. Het wachten is op uitgebreide resultaten van andere, wellicht objectievere bronnen. Dat toepassing van PSP/TSP niet altijd succesvol is blijkt uit een onderzoek van Morisio. Het blijkt dat de inpassing van PSP in een organisatie een aanzienlijke aanpassing van de PSP-aanpak vereist. De standaard aanpak zoals beschreven en vermarkt door het SEI, blijkt onvoldoende rekening te houden met lokale omstandigheden. De aanpassingen leidden volgens het onderzoek tot additionele investeringen zodat de PSP-kosten de baten overtroffen. Daar kwam bij dat het PSP-project uiteindelijk op een fiasco uitliep. De ontwikkelaars maakte zich niet een zogenoemde PSP-gedrag eigen, veranderden nauwelijks hun manier van werken en maakten geen begin met meten en bewaken van de voortgang van hun werk. Bij dit alles kwam nog dat het gebruik van PSP niet leidde tot minder foutmeldingen van de gebruiker en dus niet vanuit gebruikersperspectief tot betere software.

Kanttelingen moeten ook worden geplaatst bij de scope van PSP en TSP. Beide benaderingen zijn gericht op met name het verbeteren van bouwtraject van software ontwikkeling, een activiteit die veelal niet meer dan 15 tot 20% van de totale ontwikkelingspanning vergt. Je zou kunnen zeggen dat dit iets heeft van suboptimalisatie. Verder kan ook worden opgemerkt dat met name PSP enigszins voorbij aan het gegeven dat ontwikkelwerk voor een deel een creatief en intellectueel proces is. Het rigide, enigszins mechanistische en bureaucratisch mensbeeld van een ontwikkelaar dat schuil gaat achter de filosofie van PSP is hiermee strijdig. Dit zijn dezelfde kanttelingen die critici bij het CMM maken. In tabel 5 worden vaak gehoorde kritieken op het CMM weergegeven en wordt aangegeven in hoeverre organisaties het hiermee eens zijn.

Tabel 5: Kritiek op het CMM en de mening van organisaties

Punt van kritiek op CMM	Niet mee eens
werkt contra productief	96%
laat belangrijke zaken liggen	90%
maakt organisatie bureaucratischer	84%
leidt tot risico mijndend gedrag	Hoog

We hebben eerder al gewezen op de sterke samenhang tussen het CMM, PSP en TSP. Zo kan PSP/TSP alleen maar succesvol geïmplementeerd worden en kunnen de beoogde voordelen worden gerealiseerd als de betreffende organisatie zich minimaal op niveau twee van het CMM bevindt. Van de 18 Key Process Areas die binnen het CMM worden onderscheiden, dienen voor succesvol gebruik van PSP 12 en van TSP maar liefst 16 gerealiseerd te zijn. Dat betekent dat een organisatie voorafgaand aan het gebruik van de TSP en PSP benaderingen kostbare en tijdrovende investeringen moet plegen om op CMM niveau twee te komen. Verder geldt dat implementatie van TSP alleen maar zinvol is als de organisatie beschikt over PSP opgeleide ontwikkelaars. Gebruik van TSP is dus onlosmakelijk gekoppeld aan het gebruik van PSP.

Als we het hebben over succesvol implementeren van PSP en TSP, dan is het opvallend te moeten constateren hoe weinig gegevens beschikbaar zijn over implementatie- en gebruikskosten. Het ontbreken van heldere kosten- en batengegevens zet de deur voor een ‘welis-nietis-spel’ wagenwijd open. Juist een goede onderbouwing van de opbrengsten van dit soort benaderingen geeft een enorme stimulans om op de ingeslagen weg voort te gaan. De weinige gegevens over PSP en TSP investeringen geven aan dat de effecten van het gebruik ervan pas na maanden en soms zelfs jaren consequent gebruik zichtbaar worden. Net zoals toepassing van het CMM, is het gebruik van PSP en TSP dus een kwestie van lange adem.

In die lange adem zit tevens een gevaar. Praktijkervaringen laten namelijk zien dat het bijzonder lastig is om PSP en TSP (maar ook het CMM) een structurele positie binnen een organisatie te geven. Wat dat betreft lijkt deze benadering aan dezelfde ziekte als allerlei andere pogingen om de kwaliteit van het werk te verbeteren, zoals het opstellen, gebruiken en blijven onderhouden van metrieken, kwaliteitssystemen e.d. Vaak valt of staat zo'n initiatief bij het enthousiasme van een enkeling. Die lange adem is nodig omdat toepassing van PSP en/of TSP niet een kwestie is van het volgen van een training en het toepassen van de geleerde PSP en TSP ‘trucjes’, maar een kwestie is van een cultuurverandering; en cultuurveranderingen kosten tijd, heel veel tijd.

Tenslotte willen we opmerken dat bij veel succesverhalen over het gebruik van PSP en TSP vraagtekens gezet kunnen worden of de positieve effecten alleen maar toe te schrijven zijn aan PSP en TSP. Door actief bezig te zijn met Software Process Improvement en door product- en proceskwaliteit hoog op de agenda te zetten, treden binnen een organisatie allerlei

veranderingen op die niet direct binnen het PSP en/of TSP raamwerk vallen maar wel een positief effect hebben op de aspecten die PSP en TSP central stellen. Men moet hierbij denken aan gedragsveranderingen bij medewerkers, aan motivatie, leereffecten etc.

5. Conclusies

In dit artikel hebben we een korte uiteenzetting gegeven van de ins en outs van de PSP en de TSP benaderingen. De eerste ervaringen duiden erop dat beide benaderingen een positieve bijdrage kunnen leveren aan het verbeteren van softwareproces- en softwareproductkwaliteit. Maar ondanks de vele hosanna verhalen vraagt een toepassing ervan om een afweging tussen de voors en tegens. De beschikbare publicaties geven weinig houvast bij dit afwegingsproces omdat de literatuur gedomineerd wordt door evaluaties van het SEI. Ervaringen los van het SEI zijn schaars en richtlijnen over 'hoe succesvol te implementeren' nog schaarser. Dat wil echter niet zeggen dat een software organisatie achterover moet leunen en deze nieuwe PSP en TSP ontwikkelingen aan zich voorbij moet laten gaan. Van de andere kant moet een organisatie beseffen waar het aan begint. Introductie van deze benaderingen vraagt om een forse investering en om een lange adem terwijl mogelijke besparingen pas op de langere termijn zichtbaar worden. Toepassing ervan snijdt diep in de organisatie. Willen de investeringen renderen dan vraagt dat van een organisatie een forse cultuurverandering, een andere visie op samenwerken, op procesinrichting, op professionalisme etc. De totale organisatie 'moet dus om' en dat kan alleen maar als de organisatie er ook echt voor wil gaan en de motivatie om te veranderen ook aanwezig is. Bij deze benaderingen geldt bovendien zoets als 'alles of niets'. Dat betekent enerzijds dat succes alleen maar bereikt kan worden door toepassing van zowel TSP, PSP als CMM. De drie benaderingen zijn feitelijk onlosmakelijk met elkaar verbonden. Anderzijds betekent het dat eenmaal begonnen aan de introductie, het traject ook afgemaakt moet worden. De initiële investeringen zijn te hoog en te ingrijpend om voortijdig afgebroken te worden. Kortom, bezint eer ge begint. Daarbij komt ook dat in onze optiek toepassing van deze benaderingen neigt naar een overkill, naar het klemmen van een organisatie in een rigide keurslijf waarbij het gevaar voor het bureaucratische en procedurele aanwezig is. Een organisatie die wil experimenteren met TSP, PSP en CMM moet in zijn overwegingen meenemen dat de heersende organisatiecultuur niet te ver af moet staan van die van de CMM derivaten. Voor een innovatie-driven organisatie lijken deze benaderingen daarom minder geschikt in tegenstelling tot omgevingen die gericht zijn op ontwikkeling, onderhoud en beheer van min of meer 'stabiele' systemen. Ondanks dit soort overwegingen van voors en tegens staan de uitgangspunten van de benaderingen, te weten

- **meten** wat je doet en daardoor inzicht krijgen in datgene wat je doet,
- **plannen** en **schatten** van wat je gaat doen op basis van dat verkregen inzicht,
- op basis van het verkregen inzicht **verbeteren van de kwaliteit** van datgene wat je doet en je maakt,
- **evalueren** van dat wat je doet (plannen, schattingen, kwaliteit),
- en al deze activiteiten uitvoeren als een **cyclisch** proces

als een huis en vormen ze een must voor software ontwikkeling op organisatie-, team en individueel niveau. De kracht van de PSP en TSP benadering is dat deze uitgangspunten verpakt zijn in een praktisch toepasbare aanpak. De documentatie is vrij beschikbaar en goed gedocumenteerd.

1 Referenties

We hebben in dit artikel vaak impliciet verwezen naar bronnen, zonder deze verder te vermelden. Zouden we dit wel hebben gedaan, dan zou er nu een lange lijst met referenties moeten volgen. Die treft u niet aan maar kunt u wel bij ons opvragen (fred.heemstra@ou.nl). Wel willen we wijzen op de al eerder genoemde en uitstekende website van het Software Engineering Institute (www.sei.cmu.edu). Hierin treft u talloze verwijzingen aan naar beschrijvingen van en ervaringen met beide modellen (waaronder de boeken en artikelen van Humphrey). Verder verwijzen we naar een special issue van IEEE Software in het november/december nummer van 2000.