22 Invloed van de factor mens op softwarekwaliteit

22.1 Inleiding

Het wordt steeds duidelijker dat mensafhankelijke factoren bepalend zijn voor de kwaliteit van het product of de dienst die geleverd moet worden. Er kan nog zoveel geïnvesteerd worden in verbetering van processen, methoden en technieken, de investeringen leveren weinig op als de medewerkers die in deze verbeterde processen en met deze nieuwste methoden, technieken en tools moeten werken, niet in staat zijn adequaat hiermee om te gaan of niet bereid zijn hiermee te werken. Gebrek aan deskundigheid, verwaarlozing van opleiding- en trainingsprogramma's , een groot verloop, ziekteverzuim en het niet kunnen binnenhalen van de juiste mensen zijn allemaal factoren die ervoor kunnen zorgen dat goed bedoelde 'improvements' niet opgepakt kunnen worden. Behalve dit soort 'zakelijke' investeringen in medewerkers dient een organisatie ook 'psychologisch' te investeren in mensen. Behalve dat medewerkers niet in staat zijn (Kunnen in figuur 20.2) de potentiële rendementen van verbeteracties te benutten, kan het ook zijn dat medewerkers hiertoe niet bereid zijn (Willen in figuur 20.2). Verbeteracties leiden veelal tot veranderingen in de manier van werken, soms tot reorganisaties. Medewerkers moeten voorbereid en begeleid worden om deze veranderingen te kunnen maken. Vaak gaan veranderingen in werkwijzen gepaard met noodzakelijke gedragsveranderingen. Een softwareafdeling die de draai wenst te maken van een informele, reactieve, adhoc organisatie naar een ISO-gecertificeerde organisatie waarin wordt gemeten, gedocumenteerd en geëvalueerd, vraagt van haar medewerkers een aanzienlijke gedragsverandering. De culturen van beide typen organisaties verschillen enorm.

Van zaken als motivatie, betrokkenheid, gedrag, houding, opleiding, salariëring, toekomstperspectiefen, veilig voelen, etc. mag worden verwacht dat ze een duidelijke invloed hebben op softwarekwaliteit. Maar net als bij organisatieafhankelijke factoren geldt ook voor de meeste mensafhankelijke factoren dat er sterke indicaties zijn voor een relatie, maar dat onvoldoende bekend is hoe die relatie precies is; met andere woorden welk gedrag, welke stimulus, welk opleidingsprogramma e.d. vereist is om een bepaald niveau van kwaliteitsverbetering te realiseren. Vanuit de (organisatie- en gedrags)psychologie worden in meer algemene zin uitspraken gedaan over de invloed van dit soort factoren op de prestaties van professionals. In sommige gevallen kunnen deze bevindingen min of meer rechtstreeks vertaald worden naar de softwareontwikkelomgeving. We hebben echter binnen de softwarewereld nog te weinig zicht op de wetmatigheden van het werk van softwareontwikkelaars om zekere uitspraken te doen over wat wel en wat niet vanuit de psychologie van toepassing is. Voor het management van een organisatie is het een belangrijke vraag hoe zij met ontwikkelaars moeten omgaan, 'aan welke knoppen zij moeten draaien' om de prestaties van ontwikkelaars en daarmee de kwaliteit van het resultaat van hun werk te optimaliseren. Bij de behandeling van de factor motivatie in de volgende paragraaf zal dat dilemma worden toegelicht.

De aanpak die in dit hoofdstuk wordt gevolgd is min of meer identiek aan die in het vorige hoofdstuk. Aan de hand van een bespreking van enkele mensafhankelijke factoren wordt duidelijk gemaakt dat vooralsnog alleen maar uitspraken gedaan kunnen worden in de trant van 'er bestaat een sterk vermoeden dat ...' of 'door ontwikkelaars meer ..... bestaat er een redelijke kans dat...'. Onderzoeksresultaten die aan de hand van empirisch cijfermateriaal kwaliteitsverbetering door gerichte beïnvloeding van factoren als motivatie, gedrag, opleiding, etc. aantonen, zijn schaars. We zullen in paragraaf 22.2 laten zien dat het bijzonder

aannemelijk is dat de kwaliteit van software beter wordt naarmate ontwikkelaars meer gemotiveerd zijn. Het is vervolgens evenwel niet duidelijk waardoor ontwikkelaars gemotiveerd raken, c.q. 'wat hen drijft'. Onderzoeken op dat vlak spreken elkaar tegen. Hetzelfde geldt voor de factor gedrag. Het is enerzijds meer dan aannemelijk dat een bepaald gedrag in een bepaalde situatie leidt tot betere prestaties, maar het is anderzijds onduidelijk welke gedragsveranderingen door het management bij softwareontwikkelaars teweeggebracht moeten worden zodat ze optimaal presteren. In paragraaf 22.3 wordt op deze problematiek nader ingegaan.

Hoewel het lastig is om het verband tussen softwarekwaliteit en mensafhankelijke factoren aan te geven, zullen we enkele handvatten aanreiken. Zo wordt in paragraaf 22.4 het belang van de factor opleiding onderstreept en wordt stilgestaan bij het Personal Software Process-model (PSP). PSP is een concreet opleidingsprogramma, ontwikkeld door Watts Humphrey. Het model heeft als doel de prestaties van de softwareontwikkelaar te verbeteren of analoog aan het CMM, op een hoger niveau te brengen. Het PSP-model heeft als voordeel dat het bijzonder concreet en praktijkgericht is en op korte termijn gerealiseerd kan worden. Dat er ook nadelen kleven aan het PSP wordt in paragraaf 22.4 eveneens toegelicht.

Een factor die in het verlengde van opleiden ligt en die een langere aanloop vraagt, is professionalisering. Opleiden is een van de vele facetten die ervoor zorgt dat de software engineer kan uitgroeien tot een ware professional. In paragraaf 22.5 wordt nader ingegaan op dit onderwerp.

22.2 Motivatie als kwaliteitsbepalende factor

22.2.1 Beschrijving van motivatie

Uit tal van gedragswetenschappelijke studies blijkt dat motivatie van medewerkers en hun betrokkenheid bij het werk tot de belangrijkste factoren behoren die de kwaliteit van het werk c.q. het product bepalen. De hamvraag is echter waardoor professionals gemotiveerd raken, wat hen drijft. Uit het onderzoek van Hackman (1980) komt naar voren dat voor professionals de volgende motivatieprikkels belangrijk zijn:

- *Verscheidenheid aan verlangde deskundigheid*
  Dat wil zeggen dat een professional meer gemotiveerd raakt naarmate er een groter beroep wordt gedaan op zijn deskundigheid en naarmate de activiteiten die hij uitvoert een grotere bijdrage leveren aan de ontplooiing van zijn eigen mogelijkheden.
- *Identiteit van de taak*
  Een professional raakt meer gemotiveerd naarmate hij in staat is een klus van het begin tot het eind uit te voeren c.q. hij verantwoordelijk is voor een complete taak.
- *Belangrijkheid van de taak*
  Voor een professional is het belangrijk dat zijn werk zinvol is en dat zijn werk een wenselijke en merkbare invloed heeft op het leven van andere mensen, zowel binnen als buiten de organisatie. Hoe meer dat het geval is, hoe gemotiveerder hij zal zijn.
- *Autonomie*
  Een professional hecht grote waarde aan vrijheid, onafhankelijk en beslissings-bevoegdheid in zijn werk en vindt het belangrijk dat hij staat wordt gesteld zelf te bepalen hoe het werk wordt ingedeeld, uitgevoerd en over de resultaten wordt gerapporteerd. Hij meer autonomie hij hierin krijgt des te hoger zijn motivatie zal liggen.

- *Terugkoppeling van de taak.*
  Voor een professional is het belangrijk dat hij snel, liefst direct informatie terug ontvangt over de effectiviteit van zijn werk. Naarmate dit beter gebeurt, zal zijn motivatie toenemen.

Door Powell en Posner (1984) wordt benadrukt dat betrokkenheid bij het bedrijf de belangrijkste trigger is om motivatie bij professionals te bereiken. Dit betekent volgens hen dat de organisatie i.c. het management ervoor dient te zorgen dat medewerkers zich committeren aan hun werk, aan de producten, aan de doelstellingen, aan de organisaties. Pas dan kan hoge motivatie een feit worden. Voor het bereiken van een hoge betrokkenheid dient volgens Powell en Posner gestuurd te worden op de volgende variabelen:

- *Visie*
  In het bekende werk van Waterman (*In search of Excellence, 1982*) wordt aangegeven dat excellente ondernemingen zich onderscheiden door een uitgesproken visie hoe het bedrijf te runnen. Deze visie is een gemeenschappelijke goed van alle medewerkers, zij geloven hierin en dragen deze visie uit. De slogan van IBM 'IBM means service' drukt de prioriteit uit die IBM geeft aan de dienstverlening aan haar klanten.
- *Talenten*
  Het onderkennen en benutten van de talenten van medewerkers is een tweede factor die de betrokkenheid verhoogt. Bekend is de formule van de destijds succesvolle softwareorganisatie BSO. Het werk werd georganiseerd in kleine units, medewerkers binnen die units hadden een grote eigen verantwoordelijkheid, hadden volop de mogelijkheid eigen ideeën en creativiteit toe te passen, konden belangrijke beslissingen nemen, etc.
- *Terugkoppeling*
  Terugkoppeling door het management op het geleverde werk is een derde belangrijke factor om de betrokkenheid te verhogen. Net als bij motivatie gaat het niet om een financiële beloning maar om positieve aandacht, om waardering en het geven van het gevoel dat men presteert en een bijdrage levert aan het succes van de organisatie.

Het beeld dat naar voren komt is duidelijk: zelfstandigheid, zinvolheid, verantwoordelijkheid en geïnformeerd worden, zijn voor een professional belangrijke zaken. Wat dat betreft is er niets nieuws onder de zon en bouwt Hackman met zijn onderzoek voort op het bekende motivatieonderzoek van Fred Herzberg (Herzberg, 1959) uit de zestiger jaren. Herzberg gaat uit van het bestaan van zogenoemde 'satisfiers' en 'dissatisfiers'. De eerste groep heeft een positief effect op iemands motivatie, terwijl de tweede categorie dat nou juist niet heeft. In kolom 1 van tabel 22.1 worden deze 'satisfiers en dissatisfiers' weergegeven. Bijvoorbeeld factoren als 'resultaat van het werk' en 'erkenning als professional' zijn de meest dominante factoren, dat wil zeggen zijn het meest motivatieverhogend ('satisfiers') als ze positief worden ingevuld en het meest motivatieverlagend ('dissatisfiers') als er geen of onvoldoende aandacht aan wordt besteed. In een interessant motivatieonderzoek, weliswaar oud maar nog steeds actueel in dit kader, laat Fitz-enz (Fitz-enz, 1978) zien dat de bevindingen van Herzberg en Hackman voor een belangrijk deel eveneens gelden binnen de software engineering. Doel van zijn onderzoek was na te gaan 'waar een softwareontwikkelaar door gemotiveerd wordt, waardoor en waarvoor hij enthousiast wordt' en waarop door het management gestuurd moet worden om de kwaliteit van zijn werk te optimaliseren. Fitz-enz heeft hiervoor onderzocht hoe Herzberg's satisfiers en dissatisfiers van toepassing zijn op softwareontwikkelaars. In tabel 22.1 worden de resultaten van Herzberg en Fitz-enz vergeleken. De doelgroep van Herzberg waren medewerkers uit alle mogelijke hoeken van de

industrie, terwijl het bij Fitz-enz uitsluitend gaat om mensen werkzaam in de softwareontwikkeling.

Tabel 22.1: Vergelijking tussen resultaten van Herzberg en Fitz-enz

Herzberg (industrie)	**Fitz-enz (softwareontwikkeling)**
1. Resultaat van het werk	1. Resultaat van het werk
2. Erkenning als professional	2. Carrièreperspectief
3. Het werk zelf	3. Het werk zelf
4. Verantwoordelijkheid	4. Erkenning als professional
5. Ontplooiingsmogelijkheden	5. Ontplooiingsmogelijkheden
6. Salaris	6. Technisch overzicht, inzicht
7. Carrièreperspectief	7. Verantwoordelijkheid
8. Persoonlijke verhoudingen (in relatie tot ondergeschikten)	8. Persoonlijke verhoudingen (in relatie tot gelijken)
9. Status	9. Persoonlijke verhoudingen (in relatie tot ondergeschikten)
10. Persoonlijke verhoudingen (in relatie tot leidinggevenden)	10. Salaris
11. Persoonlijke verhoudingen (in relatie tot gelijken)	11. Persoonlijke leven
12. Technisch overzicht, inzicht	12. Persoonlijke verhoudingen (in relatie tot leidinggevenden)
13. Beleid van de organisatie	13. Zekerheid van werk
14. Werkomstandigheden	14. Status
15. Persoonlijke leven	15. Beleid van de organisatie
16. Zekerheid van werk	16. Werkomstandigheden

Hoewel er verschillen zijn, zijn deze niet erg in het oog springend. Wat in beide onderzoeken ondermeer opvalt is dat een professional zijn motivatie haalt uit zaken als het werk zelf, het product waarmee hij bezig is en zijn erkenning als professional. Bij softwareontwikkelaars staat het onderwerp carrièreperspectieven duidelijker hoger op de persoonlijke agenda dan bij andere professionals, terwijl salaris weer een beduidend lagere 'satisfier' is.

Splitsen we de resultaten van Fitz-enz op naar functies binnen softwareontwikkeling dan blijken bij sommige motivatiefactoren behoorlijke verschillen te bestaan. Verantwoordelijkheid bijvoorbeeld komt bij programmeurs pas op de negende plaats, bij projectleiders op de vierde en bij managers op de eerste. Ook blijken er duidelijke verschillen te zijn tussen verschillende leeftijdscategorieën en sekse. Wat betreft het verschil in sekse is het interessant te verwijzen naar het eerder genoemde onderzoek van Tracy Hall (1995). Zij komt in haar onderzoek tot de algemene conclusie dat vrouwelijke ontwikkelaars kwaliteit anders beleefden dan mannelijke ontwikkelaars. Deze verschillen kwamen het duidelijkst naar voren als het ging om:

- *de kwaliteit van het werk*
  Vrouwelijke softwareontwikkelaars waren consequent minder tevreden over de kwaliteit van hun eigen werk dan hun mannelijke collegae.

- *de effectiviteit van formele kwaliteitsmethoden en technieken*
  Vrouwen zijn minder overtuigd van het nut en effect van standaards, inspecties en softwaremetrieken.
- *de terugkoppeling over kwaliteit*
  Vrouwen waren meer tevreden over de terugkoppeling vanuit de organisatie over de kwaliteit van het geleverde werk naar het team waarvan ze deel uitmaakten en waren ook duidelijker meer tevreden over de persoonlijke terugkoppeling.

Wat verder opvalt in het onderzoek van Fitz-enz, nog sterker dan bij Herzberg, is dat de motivatiefactoren sterk egocentrisch gericht zijn. Verantwoordelijkheid, nummer 4 in Herzberg's top 5, is de enige factor die gericht is op het belang van de organisatie. Dat egocentrische komt nog sterker naar voren bij de beantwoording op de vraag 'in welke zaken betreffende de organisatie is men (in volgorde van belangrijkheid) het meest geïnteresseerd?'. Tabel 22.2 laat de antwoorden op deze vraag zien. Ook nu weer blijkt dat persoonlijke zaken voorop staan en dan pas worden opgevolgd door zaken betreffende het wel en wee van de organisatie.

Tabel 22.2: Antwoord op de vraag: 'in welke zaken betreffende de organisatie is men (in volgorde van belangrijkheid) het meest geïnteresseerd?'

Persoonlijke interesse van de softwareontwikkelaar
1. Informatie over de huidige prestatie in eigen werk
2. Informatie over toekomstige carrière mogelijkheden
3. Informatie over veranderingen binnen eigen organisatie-eenheid of eigen taken
4. Informatie over het personeelsbeleid dat voor hemzelf is uitgezet
5. Informatie over de winst(gevendheid) van de organisatie
6. Informatie over de organisatie strategie en de uitvoering daarvan
7. Algemene informatie over activiteiten van de organisatie.

Met dit soort gegevens in het achterhoofd kan het management bij de ontwikkeling van software gericht sturen op de motivatie van medewerkers om zodoende de kwaliteit van de software te maximaliseren. Couger (1988) is heel duidelijk in zijn advies richting management:

> *"There is good news for the managers: the number one motivating factor for software developers is the work itself. Software developers don't need a cheerleader – you can concentrate on improving their jobs".*

Hij trekt deze conclusie op basis van een motivatieonderzoek onder 1800 softwareanalisten en programmeurs.

Met dit gegeven in het achterhoofd is het verontrustend om waar te nemen dat uit een zogenoemd 'werknemertevredenheidonderzoek' blijkt dat IT'ers niet erg tevreden zijn met hun werk, in ieder geval minder dan werknemers in de algemene dienstverlening en andere sectoren in het bedrijfsleven (Sijstra, 2000). In tabel 22.3 worden enkele cijfers gegeven. Volgens Sijstra heeft de relatieve lage tevredenheid te maken met het hoge opleidingsniveau van de gemiddelde IT'er en de hoge marktwaarde van zijn specialisme. Uit het onderzoek komt verder naar voren dat de belangrijkste 'satisfier/dissatisfier' de manier van leidinggeven

is waarmee een IT'er wordt geconfronteerd. Sijstra merkt echter op dat elk individu een andere stijl van leidinggeven wenst.

Tabel 22.3: Resultaten van een 'werknemertevredenheidonderzoek'

	Binnen de IT	Buiten de IT
Oordeel werkomstandigheden	6,4	7,3
Oordeel over de werkzaamheden	6,7	7,3
Oordeel over collega's	8,1	8,4
Oordeel over ontwikkelingsmogelijkheden	6,7	5,4

Een soortgelijk onderzoek in Groot-Brittannië geeft ook te denken. Bijna driekwart van de Britse automatiseerders die meer dan 230.000 euro per jaar verdienen, is op zoek naar een nieuwe baan. Van hen staat 63% ingeschreven bij negen of meer verschillende wervings- en selectiebureaus (IT Research House, 2000). Slechts 9% staat bij geen enkel bureau ingeschreven. Een verassing is dat naast zaken als 'een interessante baan' en 'een uitdaging', 'geld' als de belangrijkste motivator worden beschouwd door de Britse IT'ers. Zaken als 'Status', 'Zekerheid' en 'Corporate Benefit' worden nauwelijks op prijs gesteld. Dit wijkt behoorlijk af van de eerder genoemde onderzoeksresultaten.

Als algemeen geaccepteerd is dat een tevreden werknemer een gemotiveerde werknemer is en dat de mate van gemotiveerdheid een belangrijke factor is die de kwaliteit van software beïnvloedt dan geven deze recente onderzoeksresultaten te denken.

22.2.2 Conclusies van motivatie als kwaliteitsbepalende factor

Zoals we in de inleiding al aangaven is de samenhang tussen softwarekwaliteit en motivatie een weerbarstig onderwerp. Uit legio onderzoeken komt naar voren dat er zeker een relatie is tussen beide, maar hoe deze relatie precies ligt, is niet altijd even duidelijk. Uit het onderzoek van IT Research House zou geconcludeerd kunnen worden dat motivatie iets te maken heeft met het persoonlijk gedrag van iemand. Door schaarste op de arbeidsmarkt is een IT'er de afgelopen jaren een schaars goed geworden en heeft de marktwerking ervoor gezorgd met zaken als geld, arbeidsvoorwaarden en carrièreperspectieven een sterke druk wordt uitgeoefend op de IT'er. Het zijn vervolgens sterke schouders die de weelde kunnen dragen en vele IT'ers laten zich blijkbaar verleiden door dit soort satisfiers. Het wel of niet hiervoor openstaan wordt voor een deel bepaald door een diepere schil dan motivatieprikkels en heeft iets van doen met overtuiging, gedrag, levenshouding en dergelijke. Hoewel interessant en zinvol, willen we niet zover gaan om dit soort filosofische onderwerpen verder uit te diepen. Waar we in de volgende paragraaf wel enkele woorden aan willen wijden is de invloed van de factor gedrag op de kwaliteit van software.

22.3 Gedrag en gedragsverandering als kwaliteitsbepalende factor

22.3.1 Beschrijving van gedragskenmerken

Zoals we in deel III hebben kunnen zien is een organisatie in staat haar proces van softwareontwikkeling op een hoger niveau van volwassenheid te brengen door te investeren in

de zogenoemde Key Process Areas. De kans om een hoger niveau te bereiken is groter als niet alleen het proces kwalitatief hoogwaardiger is maar als ook 'de factor mens hoogwaardiger' is (Weinberg, 1994) (Balla, 2000). De uitdrukking 'de factor mens hoogwaardiger' is natuurlijk een eigenaardige en vraagt om enige toelichting. We zullen deze toelichting geven aan de hand van het werk van Berkman (1999) en Weinberg (1994).

Berkman onderscheidt twee vormen van gedrag: gedrag dat hoort bij een organisatie op een lager 'maturity' level en gedrag dat hoort bij een hoger 'maturity' level.
Het gedrag van mensen op een lager 'maturity' level kenmerkt zich als volgt:
- op perceptie gebaseerde acties
- reactief gedrag (firefighting)
- focus op problemen
- weinig gebruik van Key Performance Indicatoren (KPI's)
- weinig verantwoordelijkheden op alle levels
- weinig communicatie.

Dit gedrag heeft tot gevolg dat er geen afstemming tussen de afdelingen is en dat men altijd achter de feiten aan loopt. De gevolgen hiervan zijn grote efficiëntieverliezen en matige of zelfs slechte kwaliteit van het eindresultaat (het product).

Het nieuwe, gewenste gedrag (op een hoger 'maturity' level) is als volgt te omschrijven:
- de te ondernemen acties zijn op data gebaseerd
- de organisatie is pro-actief
- de focus wordt niet meer op problemen maar op oplossingen gelegd
- het gebruik van KPI's wordt gestimuleerd
- verantwoordelijkheid is op lagere niveaus gelegd
- er wordt meer gecommuniceerd en afgestemd tussen afdelingen.

Een gedragsverandering van een lager naar een hoger 'maturity' level is echter niet eenvoudig te realiseren (Berkman, 1999). Investeringen in systemen en productiemiddelen kunnen betrekkelijk snel en eenvoudig zijn, zeker in vergelijking met een gedragsverandering. Een gedragsverandering kun je niet zomaar kopen; het is een complex 'goed'. Als een organisatie wenst te investeren in gedragsverandering als middel om de prestaties te verbeteren, dan moet er eerst een schets worden gemaakt van de huidige gedragssituatie. Er is namelijk een gat tussen de huidige en de gewenste situatie (behavorial gap). Om de 'behavorial gap' te dichten moet, zo vervolgt Berkman, aandacht besteed worden aan de volgende zaken.
- *Visie*
  Iedereen in de organisatie moet werken aan het bereiken van hetzelfde en bekende doel.
- *Vaardigheden*
  Om de doelen te kunnen bereiken moet het personeel beschikken over specifieke vaardigheden. Deze vaardigheden moeten dus bekend zijn.
- *Incentives*
  Zonder incentives zullen mensen hun gedrag niet veranderen. De vraag die mensen bij verandering vaak stellen, is: "wat zit er in voor mij'. In de vorige paragraaf hebben we laten zien dat zo'n incentive niet alleen betekent meer salaris, maar dat ook andere 'satisfiers', zoals interessant werk, ontplooiingsmogelijkheden e.d. belangrijk zijn.
- *Bronnen*
  Onder bronnen kan men tijd, geld, informatie, mensen en faciliteiten verstaan. Vooral tijd is onmisbaar. Immers een gedragsverandering kost tijd, veel tijd.

Met name de wereld van de softwareontwikkeling is er één die voortdurend in verandering is. Proceskwaliteitsmodellen als het CMM, Bootstrap en SPICE propageren zelfs voortdurend veranderen door in een groeimodel door middel van 'continuous improvements' steeds andere en betere werkwijzen toe te passen. Een level 1 organisatie verschilt hemelsbreed van een level 3 of 5 organisatie. Hoewel in het CMM nadrukkelijk gewezen wordt op het belang van gedragsveranderingen bij het beklimmen van de CMM-ladder, wordt in dit soort modellen doorgaans te gemakkelijk van medewerkers verwacht dat zij die groei moeiteloos kunnen meemaken. Te weinig wordt beseft dat hiervoor aanzienlijke gedragsveranderingen nodig zijn.

Weinberg (Weinberg, 1994) onderscheidt niet twee maar zes menselijke gedragspatronen die min of meer corresponderen met het 'maturity' level van de organisatie waarin ze werken. In tabel 22.4 worden deze gedragspatronen met behulp van een paar trefwoorden omschreven en wordt aangegeven welke gedragsveranderingen moeten plaatsvinden om op een hoger level te komen.

Tabel 22.4: Weinberg's gedragskenmerken in relatie tot 'maturity' levels.

Level	**Gedragskenmerk**	**Overgangseis**
1. 'Oblivious'	'we weten zelfs niet dat we bezig zijn met de uitvoering van een proces'	*Nederigheid*; te realiseren door stil te staan bij alles wat je doet'
2. 'Variable'	'we doen wat ons op dat moment het beste lijkt'	*Vaardigheid*; te realiseren door training en ervaring
3. 'Routine'	'we werken volgens vaste regels (behalve wanneer we in paniek raken)'	*Stabiliteit*; te realiseren door Quality Software Management
4. 'Steering'	'we kiezen uit alle regels / voorschriften op basis van het resultaat dat we ermee kunnen bereiken'	*Waakzaamheid*; te realiseren door tools en technieken
5. 'Anticipating'	'we hebben regels en voorschriften gebaseerd op onze ervaring ermee'	*Aanpasbaarheid*; te realiseren door persoonlijke ontwikkeling
6. 'Congruent'	'iedereen is betrokken bij het doorlopend verbeteren van alles'	

Weinsberg's onderzoek maakt duidelijk dat gedrag en met name het vermogen om het gedrag aan te passen een belangrijke factor is om iemands prestatie en de kwaliteit van zijn werk te verhogen.

22.3.2 Conclusies van gedrag als kwaliteitsbepalende factor

De onderzoeken van Berkman en Weinberg maken duidelijk dat niet alleen een organisatie moet groeien naar een hoger 'maturity' level om betere software te kunnen maken, maar dat ook een persoon parallel aan dit groeiproces moet meegroeien naar een hoger 'maturity' level. En net zoals een organisatie een geleidelijk, stap-voor-stap groeitraject doorloopt, zo zal ook het gedrag van een persoon een soortgelijke, geleidelijke en stap voor stap groeitraject moeten doormaken. Er is nog maar weinig bekend hoe het gedrag van software engineers in de 'goede' richting veranderd kan worden. We hebben wel een idee van een gewenst gedrag maar tasten nog vaak in het duister hoe dit gewenste gedrag te realiseren. Daar komt bij dat door veranderende visies op het ontwikkelwerk zelf de ideeën over gewenst gedrag niet stabiel zijn en dat gedragsveranderingen veel, heel veel tijd vragen. Net zoals bij de factor motivatie geldt voor de factor gedrag dat er meer dan aannemelijk een verband is tussen softwarekwaliteit en gedrag, maar dat de stand van de wetenschap vooralsnog tekort schiet om hier zekere uitspraken over te doen.

Zeker is dat gedragsveranderingen alleen mogelijk zijn als de voorwaarden om te kunnen veranderen aanwezig zijn. Een van die voorwaarden is de beschikbaarheid van voldoende kennis en kunde om op een hoger niveau te kunnen opereren. In de volgende paragraaf zullen we stilstaan bij de relatie tussen opleiden en de kwaliteit van software. Voordat het zover is, willen we aan de hand van een kort intermezzo het belang van kennis en kunde, kortom van een goed opgeleide ontwikkelaar illustreren.

22.4 Het belang van kennis en kunde

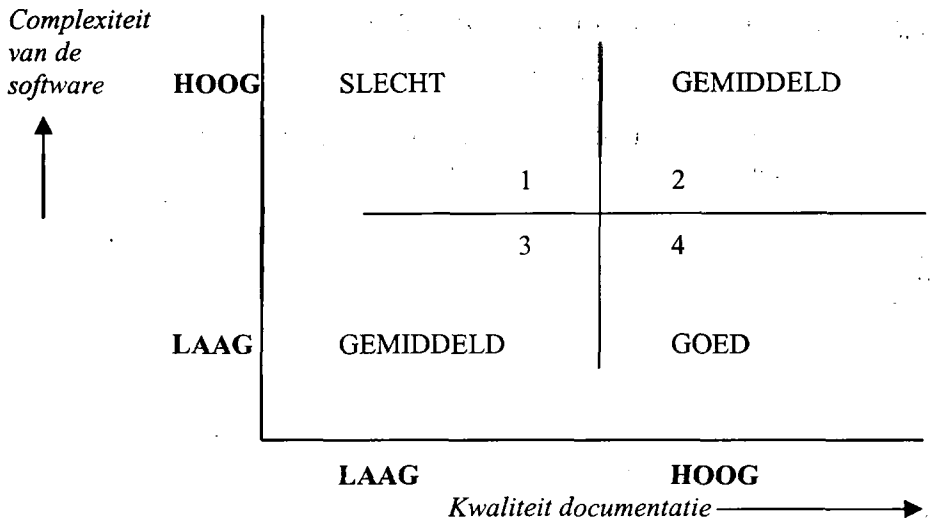
Het effect van opleiden, wordt ondermeer geïllustreerd door Les Hatton (1998). Hij geeft aan dat de vakkennis van een softwareontwikkelaar een bepalende factor is om snel fouten in de software te vinden, te analyseren en te verbeteren. De snelheid van vinden, analyseren en verbeteren is volgens Hatton afhankelijk van onder andere:

- de complexiteit van de software en
- de kwaliteit van de documentatie.

Een van de dilemma's van de software engineering is dat veel ontwikkelaars ervan uitgaan dat de door hen ontwikkelde software goed gedocumenteerd en goed ontworpen is, terwijl bij het optreden van fouten het tegenovergestelde blijkt. De software is veel complexer dan men denkt, is vaak beperkt gedocumenteerd en slecht ontworpen. Goede, dat wil zeggen volledige en eenduidige documentatie dient garant te staan voor bijvoorbeeld begrijpelijke foutmeldingen. Op basis van een goed ontwerp wordt complexe software inzichtelijk en 'leesbaar'. Hierdoor is het mogelijk de oorzaak van een fout snel te vinden. Door complexiteit van de software en kwaliteit van de documentatie aan elkaar te relateren ontstaan vier combinaties zoals weergegeven in figuur 22.3.

De meest beroerde situatie treedt op in het kwadrant 1 'complexiteit hoog, kwaliteit laag'. We hebben dan te maken met onbegrijpelijk foutmeldingen en bovendien met software die nauwelijks toegankelijk is. Hatton noemt het volgende vermakelijke voorbeeld:

> ***"error -23009: there are already more than 64 tcp or udp streams open {tcp:104}"***
> *This appeared on the screen of a new Macintosh g3 desktop running Apple's latest and greatest version of OS 8. Two desolate hours later the person realised that an acceptable substitute might have been: "modem not responding".*



Figuur 22.3: De relatie tussen kwaliteit van documentatie en complexiteit van software

De ideale situatie is er een waarbij de foutmelding rechtstreeks wijst naar de bron van de fout en de software zo goed is gestructureerd en geordend dat moeiteloos de fout gealloceerd kan worden. Een voorbeeld van een situatie, kwadrant 4, met 'complexiteit laag, kwaliteit hoog' is (Hatton 1998):

> ***"dereference pointer contents 0x0 at***
> ***Strlen(...) Called from***
> ***Line 126 of myc_constexpr.c called from***
> ***Line 247 of myc_evalexpr.c called from***
> ***Line 2459 of myc_expr.c"***
> *It points unerringly at the responsible code line and usually takes a matter of moments to fix.*

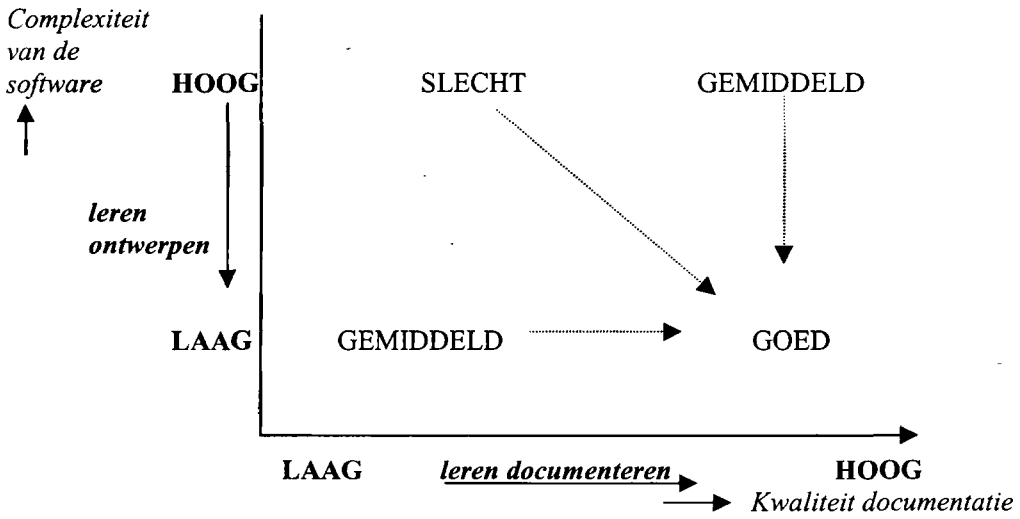
Hatton geeft talloze voorbeelden van de gevolgen van matige documentatie in combinatie met matig ontworpen software (kwadrant 3). Een bijzonder vermakelijk voorbeeld is het volgende:

> ***"system stressed ..."***
> *This appeared on the cash registers of the author's local pub. An hour of exciting discussion about communication protocols, deadlocks and such later, one realised an acceptable substitute might have been: "printer out of paper".*

Om dit soort problemen te voorkomen pleit Hatton voor maatregelen om:
- de documentatie van software te verbeteren
- de complexiteit van software te verminderen.

Een van de belangrijkste maatregelen om het eerste doel te bereiken is het beter opleiden van ontwikkelaars zodat zij in staat zijn documentatie te schrijven die anderen kunnen begrijpen. Foutmeldingen dienen ondubbelzinnig te zijn en precies aan te geven om welke fout het gaat en welke actie nodig is om de fout te herstellen. Om de complexiteit te verminderen pleit Hatton voor maatregelen die leiden tot een beter ontwerp. Door ongestructureerd ontwerpen is de kans bijzonder groot dat in de kluwen van code niet meer is na te gaan waar wat staat en waar een foutmelding naar verwijst. Ook hiervoor geldt dat door middel van opleiding de

ontwikkelaar geleerd moet worden gestructureerd te ontwerpen. Het effect van beide opleidingsmaatregelen wordt in figuur 22.4 weergegeven.



Figuur 22.4: Op weg naar eenvoudige en goed gedocumenteerde software

De illustraties van Les Hatton benadrukken het belang van opleiding, kennis en kunde. In de volgende paragraaf willen we aangeven op welke wijze opleiden ingevuld kan worden. Hiervoor kunnen legio mogelijkheden worden aangedragen. We zullen ons beperken tot het opleidingsprogramma dat door het Personal Software Process-model wordt voorgeschreven. De keuze voor dit model is gemaakt omdat PSP een duidelijke relatie heeft met het uitvoerig behandelde CMM en het nog te behandelen Team Software Process (TSP) model. Daarbij komt dat door Humphrey, de ontwikkelaar van PSP, gemeten is naar het effect van de toepassing van PSP op softwarekwaliteit. Kortom, er is empirisch evaluatiemateriaal beschikbaar.

22.5 Personal Software Process (PSP)

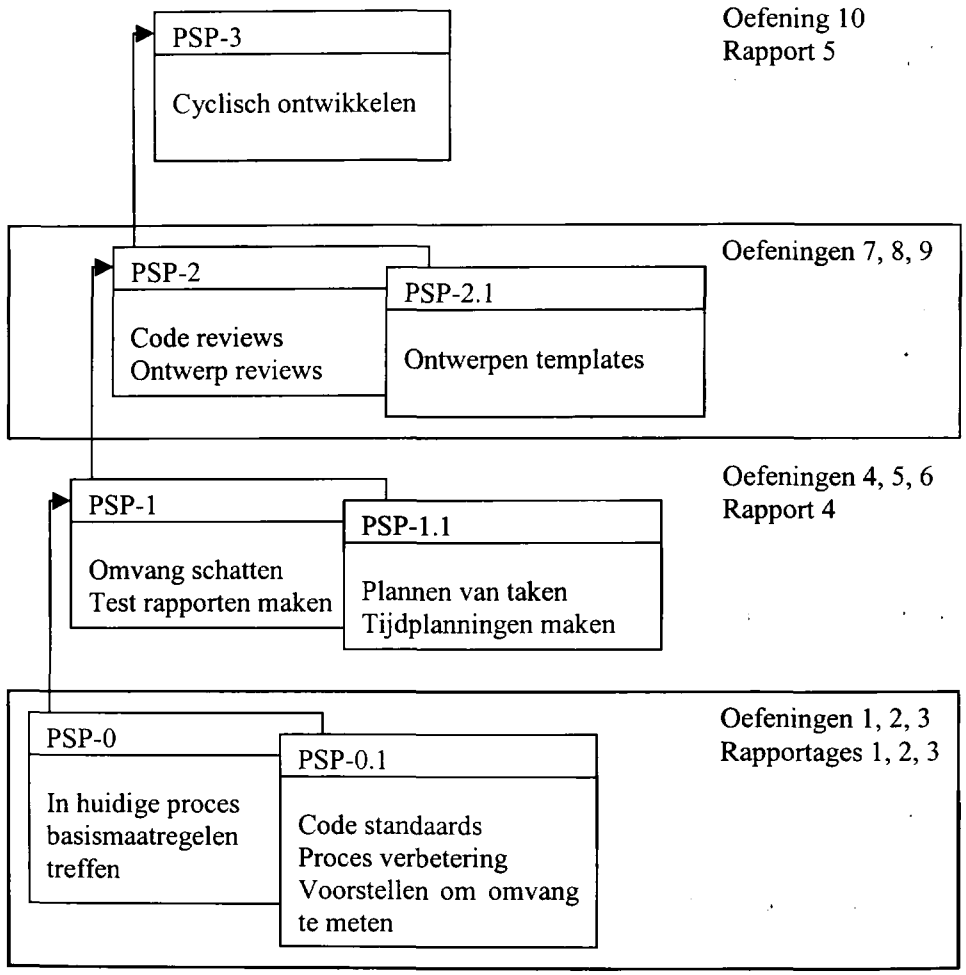
Watts Humphrey (1996) geeft een korte en duidelijke omschrijving van zijn PSP-model:

> *"The Personal Software Process' is a framework of techniques to help engineers improve their performance – and that of their organizations – through a step-by-step, disciplined approach to measuring and analyzing their work".*

Volgens Humphrey (2000) zijn de vruchten van het gebruik van PSP bijzonder zoet: minder fouten in de code, betere schattingen en planningen en hogere productiviteit. Bovendien is PSP vanaf CMM level 3 een noodzakelijk vehikel om procesverbeteringen mogelijk te maken. Een andere reden voor Humphrey om PSP te ontwikkelen was de constatering dat het CMM vooral binnen grote organisaties weerklank vond en dat het CMM voor kleine softwareorganisaties iets was als 'schieten met een kanon op een mug'.

Zoals uit figuur 22.5 blijkt biedt PSP een vastomlijnd leertraject dat een softwareontwikkelaar moet doorlopen. Door in totaal tien leeropdrachten uit te voeren en vijf rapporten te schrijven doorloopt de ontwikkelaar de verschillende PSP levels (van PSP-0 naar PSP-3). Stap voor

stap wordt zijn inzicht vergroot en krijgt hij handvatten om betere software op een betere wijze te maken.. Uiteindelijk moet een ontwikkelaar dan in staat zijn zijn eigen werk op een gedisciplineerd wijze uit te voeren. Een centraal thema binnen PSP is het meten van de eigen prestatie. Na iedere stap c.q. leeropdracht dient de ontwikkelaar het effect van het geleerde te meten en te evalueren. In figuur 22.5 wordt de structuur van PSP weergegeven.



Figuur 22.5: De structuur van PSP

Zonder al te gedetailleerd in te gaan op het PSP-leertraject, kunnen de verschillende PSP levels met daarin opgesloten de PSP-concepten als volgt globaal worden omschreven:

Personal Management
In deze eerste stap leert de ontwikkelaar hoe hij bepaalde PSP-formulieren en -werkwijzen in zijn eigen werk moet toepassen. Hij doet dit door ontwikkeltijd en aantal fouten te meten. De ontwikkelaar moet dus gegevens verzamelen. Met deze gegevens kan hij nagaan of hij in de loop van de tijd vorderingen maakt (zijn kwaliteit beter wordt). De stap Personal Management bestaat uit drie fasen: planning, ontwikkeling (ontwerp, codeer, compileer en test) en postmortem. In de eerste fase krijgt de ontwikkelaar a) een standaard die hem voorschrijft hoe te coderen, b) een metriek om de omvang te meten en c) een zogenoemd 'Process

Improvement Proposal' formulier. Met behulp van dit formulier houdt de ontwikkelaar bij welke problemen optreden en legt hij ideeën vast hoe hij in het vervolg zijn eigen proces kan verbeteren. De ontwikkelaar leert bovendien wat het nut is van het verzamelen en vastleggen van gegevens.

Personal Planning
In deze stap wordt een methode geïntroduceerd waarmee de ontwikkelaar op basis van zijn eigen verzamelde gegevens de omvang en de ontwikkeltijd van zijn programma's kan inschatten. Verder krijgt hij geleerd hoe hij plannen moet opstellen en hoe plannen worden opgedeeld in taken en activiteiten. Door vroeg in het ontwikkeltraject te plannen, leert de ontwikkelaar het belang om al snel gegevens te verzamelen. Op deze wijze ervaart hij het nut van statistische schattings- en planningsmethoden.

Personal Quality
In deze stap komt het omgaan met fouten aan de orde. Aan de hand van de opgetreden fouten in zijn eigen programma's leert de ontwikkelaar checklists te maken en te gebruiken om ontwerpen en code te reviewen. De ontwikkelaar leert waarom het belangrijk is meteen vanaf het begin te letten op kwaliteit en leert hoe hij op efficiënte wijze reviews kan uitvoeren. Hij ervaart ook dat hij de checklists sneller kan aanpassen naarmate zijn vaardigheden en ervaringen toenemen. In deze stap worden ontwerpspecificatie en analysetechnieken geïntroduceerd, samen met fout preventie technieken, procesanalyse en benchmarktechnieken. Door te meten hoe lang hij bezig is met een taak en het aantal fouten dat hij veroorzaakt en oplost, is de ontwikkelaar in staat om zijn eigen prestaties te evalueren en te verbeteren

Scaling Up
Dit is de laatste stap in de PSP-aanpak. De ontwikkelaar wordt nu geconfronteerd met steeds lastiger en omvangrijke opdrachten. Hij krijgt nu geleerd wat onder andere ontwerp verificatie methoden zijn.

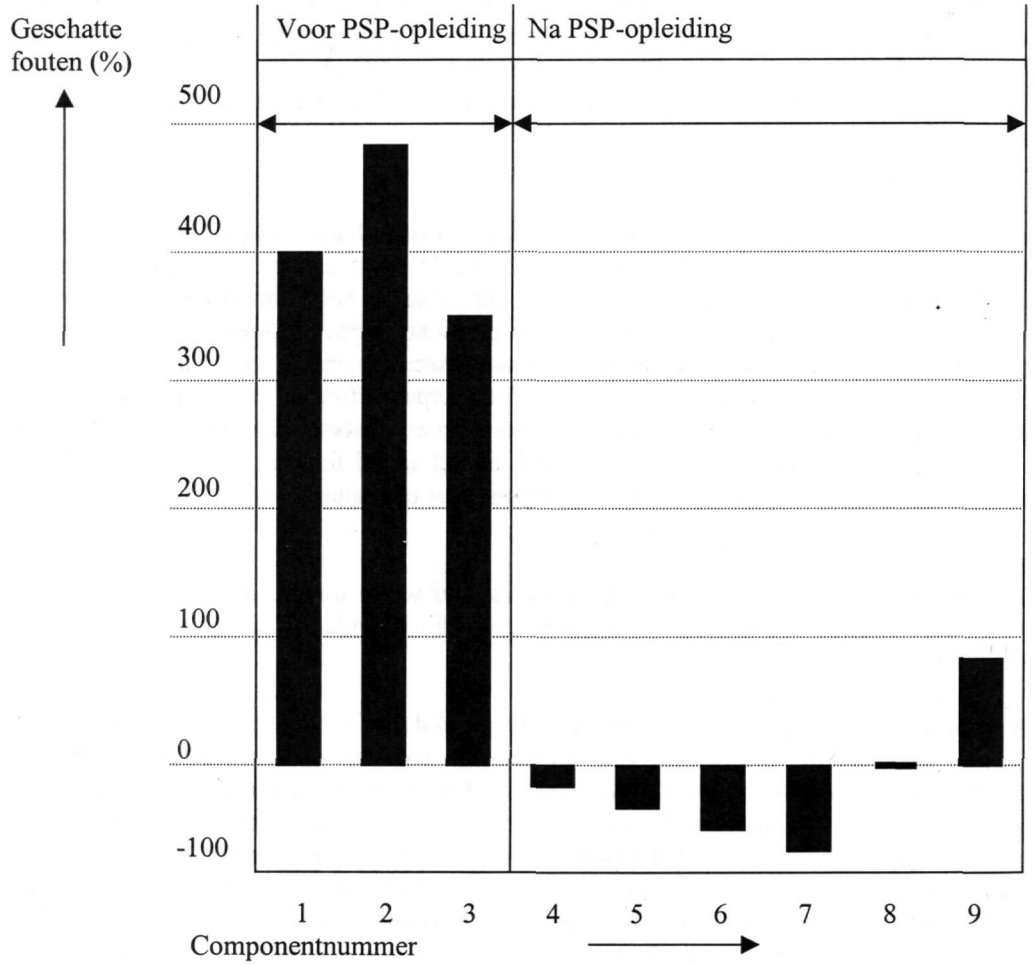
We hebben in het vorige deel kunnen constateren dat het CMM handvatten biedt c.q. activiteiten voorschrijft die uitgevoerd dienen te worden om procesverbetering op organisatieniveau te realiseren. Het CMM verschaft dus mogelijkheden om het werk goed uit te voeren. Het CMM garandeert dat evenwel niet. 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 level twee bevinden.

Het CMM en PSP bevruchten elkaar wederzijds. Het CMM biedt die gestructureerde en geordende omgeving die ontwikkelaars nodig hebben om hun werk zo goed mogelijk te doen. PSP levert ontwikkelaars die in staat zijn topkwaliteit te leveren mits zij kunnen werken in een daarvoor aangemeten omgeving.

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 en Terdyne (Humphrey, 1998). Evaluatiegegevens van gebruikers laten

duidelijk de voordelen van het PSP-leerprogramma zien (Humphrey, 1996). Naast beter schatten en plannen en een hogere productiviteit wordt door PSP geschoolde ontwikkelaars betere software geleverd.

In figuur 22.6 worden in gecomprimeerde vorm de evaluatiegegevens van een softwareontwikkelteam bij het bedrijf Advanced Information Services gegeven (Ferguson e.a., 1997).



Figuur 22.6: Evaluatiegegevens van toepassing van PSP

Halverwege het ontwikkeltraject kreeg het team een PSP-opleiding. In de figuur worden de relatieve schattingsfouten van voor en na de opleiding in beeld gebracht. Voor component 1 bijvoorbeeld, werd de ontwikkeltijd oorspronkelijke geschat op 4 weken. In werkelijkheid bleek de benodigde tijd 20 weken te zijn. Een onderschatting van dus 394 %. Na de PSP-opleiding bleken de ontwikkelaars veel realistischer schattingen te kunnen afgeven. In tabel 22.5 worden de testtijden van een ontwikkelteam vergeleken voor en na het ondergaan van een PSP-leertraject (Furgeson e.a., 1997).

Tabel 22.5: Besparingen bij het testen door gebruikmaking van PSP

Testtijden voordat de PSP training was gegeven		
Project	**Omvang**	**Testtijden**
C	19 requirements	3 testcycli
D	30 requirements	2 maanden
H	30 requirements	2 maanden
Testtijden voordat de PSP training was gegeven		
Project	**Omvang**	**Testtijden**
B	24 requirements	5 dagen
E	2.300 regels code	2 dagen
F	1.400 regels code	4 dagen
G	6.200 regels code	4 dagen
I	13.300 regels code	2 dagen

De eerste resultaten van het gebruik van PSP zijn hoopgevend (zie ook Kamatar en Hayes, 2000) en Zong e.a. (2000). Enkele kritische opmerkingen zijn echter op zijn plaats. Allereerst zijn de evaluatiegegevens gepubliceerd door de ontwikkelaars van PSP. Het wachten is op de resultaten van andere, wellicht objectievere evaluatoren. De eerste resultaten uit een andere dan de SEI-omgeving laten zien dat de gemaakte investeringen de baten ver overtreffen. Bovendien leidde het gebruik van PSP niet tot minder fouten die door de gebruiker werden gemeld (Morisio, 2000) .Een tweede opmerking is principiëler van aard. Hoewel er positieve elementen te vinden zijn in PSP, gaat het enigszins voorbij aan het gegeven dat ontwikkelwerk voor een deel een creatief en intellectueel proces is. Het rigide, enigszins mechanistische mensbeeld van een ontwikkelaar dat schuil gaat achter de filosofie van PSP is hiermee strijdig.

Het voordeel van het PSP ten opzichte van bijvoorbeeld acties die moeten leiden tot gedragsveranderingen of hogere motivatie bij softwareontwikkelaars is dat het opleidingsprogramma van PSP uiterst concreet en praktisch is. In plaats van een lange aanlooptijd zoals bij gedragsveranderingen het geval is, kan met het PSP morgen aan de slag gegaan worden en zijn de effecten op korte termijn zichtbaar en meetbaar. Bovendien is opleiden c.q het PSP een opstap naar een professionalisering van het vakgebied software engineering en haar beoefenaars. Maar voor professionalisering is meer nodig dan alleen kennis, kunde en opleiding. Een zekere mate van volwassenheid, een begrip dat al herhaaldelijk in dit boek is genoemd, is een voorwaarde. Voor een jonge discipline als de software engineering is dat een lastig obstakel. In de volgende paragraaf geven we enkele bespiegelingen over dit onderwerp.

22.6 Professionalisering

Dietz (1999) heeft met zijn L_PASO-model een grote stap voorwaarts gezet in het realiseren van professionaliteit van het vakgebied software engineering. Dietz begint met de constatering dat op de weg naar professionaliteit meteen al een lastige hobbel ligt, namelijk de onoverzichtelijkheid van het werkterrein, de 'jungle' aan beroepen en functies. Wie weet wat een system support analist is of een informatie-architect of een network-consultant, laat staan

dat men een notie heeft wat de taken zijn die bij deze functies horen. Het is zondermeer duidelijk dat de jonge discipline software engineering geen heldere, bruikbare en stabiele structuur heeft. Dat maakt de herkenning en ook de erkenning van de software engineer, vergeleken met andere beroepsgebieden, extra lastig. De eerste stap op weg naar professionaliteit in elk vakgebied en dus ook dat van de software engineering is volgens Dietz het ondubbelzinnig en nauwkeurig definiëren van de begrippen die voor het bereiken en bewaken ervan relevant zijn. Het L_PASO-model biedt een structuur om die relevante begrippen in de software engineering te definiëren. Op basis van die begrippen die het werkterrein van de software engineer in kaart brengen, worden competenties beschreven die een software engineer dient te bezitten en de prestaties die met die competenties geleverd moeten kunnen worden. L_PASO geeft verder aan hoe de beroepsbeoefenaar zijn competenties kan verhogen en in het verlengde daarvan zijn professionaliteit kan verbeteren. Aangegeven wordt hoe, welke competenties verworven kunnen worden en hoe deze gecertificeerd kunnen worden.

Het L_PASO-model start met het aangeven van de soorten systemen waarmee de software engineer in zijn werk te maken heeft. Het betreft de volgende drie systemen, waarvan de eerste twee uitvoering ter sprake zijn gekomen in dit boek:
1. bedrijfsystemen
2. informatiesystemen
3. infrastructurele systemen.

Bij het bestuderen van een systeem kan de software engineer twee, tegengestelde, oriëntaties innemen. Deze zijn:
1. *functie-oriëntatie* (gericht op de functionaliteit/extern gedrag van het systeem)
2. *constructie-oriëntatie* (gericht op de constructie/interne werking van het systeem).

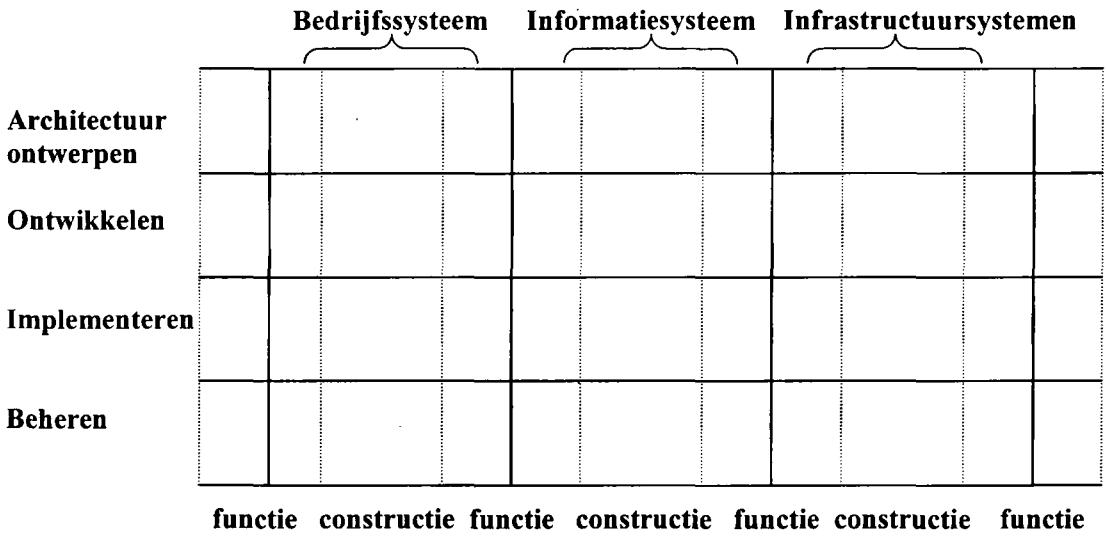
De activiteiten die door de software engineer worden verricht met betrekking tot de hiervoor genoemde soorten systemen, kunnen worden ingedeeld in de volgende vier groepen:
1. *architectuur ontwerpen*
2. *ontwikkelen*
3. *implementeren*
4. *beheren.*

Door deze drie begrippen (systemen, orientaties en activiteiten) onderling te relateren ontstaat het domein van de software engineering, waarbij elke combinatie van een activiteitsoort, een systeemsoort en een systeemoriëntatie een elementair stukje is van het domein (zie figuur 22.7)

In zijn boek gaat Dietz uitvoerig in op alle domeinen. We volstaan hier met het noemen van de overall structuur en een verwijzing naar het betreffende boek.

Als het gaat om prestaties en competenties maakt het model wat betreft prestaties een onderscheid in de volgende rollen:
- Uitvoering
- Management
- Advisering
- Auditing
- Onderwijs.

Door elk domein te combineren met deze rollen ontstaan zogenoemde werkelementen. Al het werk dat een software engineer doet bestaat uit een aantal van deze werkelementen. Bij competenties wordt een onderscheid gemaakt in competentiesoorten en competentieniveaus, die onderling gecombineerd competentie-elementen opleveren. Het is nu zaak om werkelementen en competentie-elementen goed op elkaar af te stemmen. Het L_PASO-model geeft aan hoe een en ander gebeurt.



Figuur 22.7: Het domein van de software engineer

Binnen de IEEE Computer society and the Association for Computing Machinery wordt eveneens gewerkt aan de ontwikkeling van een professioneel vakgebied (Bourque, 1999). Zo is er een werkgroep actief met het in kaart brengen van de zogenoemde 'Software Engineering Body of Knowledge' (SWEBOK) (http://www.swebok.org). Zonder zo'n gemeenschappelijke visie en omschrijving van Software engineering en wat wel en wat niet tot het vakgebied behoort, is volgens de werkgroep professionalisering niet mogelijk, kunnen er geen eenduidige diploma's worden afgegeven en kunnen er ook geen accreditaties voor opleidingen worden toegekend. In de Body of Knowledge wordt een onderscheid gemaakt tussen de volgende acht belangrijke aandachtsgebieden, waarvan softwarekwaliteit overigens er een is.

Aandachtsgebieden:

- softwareconfiguratiemanagement
- softwareconstructie
- softwareontwerp
- software engineering-infrastructuren
- software engineering-management
- software engineering-proces
- software-evolutie en -onderhoud
- softwarekwaliteit
- software requirements-analyse
- softwaretesten.

Zoals al eerder opgemerkt, is het L_PASO-model een goed stap in de richting van professionalisering van de software engineering. Temeer ook omdat het model op verzoek van

de beroepsvereniging van informatici is ontwikkeld. Het model is een eerste stap, gebruiken en verder uitbouwen van het model is een tweede stap. Voor zo'n tweede stap is de aanwezigheid van een krachtige beroepsvereniging een belangrijke voorwaarde. We zullen enkele woorden wijden aan beroepsverenigingen en aangeven wat hun bijdragen is aan de professionalisering van de software engineering. We zullen ons beperken tot de volgende verenigingen:

- de Vereniging van Registerinformatici
- 'Council of European Professional Informatics Societies'
- 'Professional Development Scheme'.

De VRI (Vereniging van Registerinformatici) heeft als haar missie geformuleerd 'het bevorderen van het kwaliteitsniveau waarop in Nederland de beroepsuitoefening door Registerinformatici plaatsvindt'. De hiervan afgeleide doelen zijn:

- het verhogen van het kwaliteitsniveau waarop in Nederland de beroepsuitoefening door registerinformatici plaatsvindt,
- het verenigen van informatici op grond van opleidings- en ervaringseisen en de gedragscode.

Voor het bereiken van deze doelen ontplooit de vereniging een aantal activiteiten, waarvan we enkele belangrijke willen noemen:

- toelating van informatici op grond van door de VRI gestelde criteria
- toetsing van naleving van de gedragscode en afhandeling van klachten over de naleving ervan
- initiëren van onderzoek en ontwikkelingen op het gebied van de kwaliteit binnen het vakgebied.

De gedragscode is een belangrijk instrument dat door de VRI wordt voorgeschreven. De code luidt als volgt:

> *'Bij mijn handelen als informaticus zal ik steeds het belang van de samenleving in al haar facetten positief dienen. Ik heb mij daarom laten registreren in het Register van Informatici. Ik geef daarmee te kennen dat ik als zodanig publiekelijk herkenbaar wil zijn en aangesproken wil worden op de gedragscode.'*

Deze gedragscode wordt vervolgens vertaald in een groot aantal gedragsregels, die hier niet verder ter sprake zullen komen. Hiervoor verwijzen we naar de VRI.

De 'Council of European Professional Informatics Societies' (CEPIS) heeft als missie het bevorderen van de professionalisering van het vakgebied informatie & communicatietechnologie (ICT). Om deze missie te realiseren heeft de CEPIS zich als doelen gesteld om

1. de basis vaardigheden en kennis te definiëren waaraan een IT professional (in Europa) dient te voldoen en
2. te komen tot een internationaal erkend certificaat.

De twee fundamenten van de CEPIS zijn verwoord in:

1. de 'European Informatics Skills Structure', waarin van de werkgebieden binnen de ICT standaards worden gegeven van de benodigde kennis en vaardigheden
2. het 'European Informatics Continuous Learning Programme', dat gebruikt kan worden om via een self-assessment te onderzoeken welk leertraject uitgezet dient te worden.

Zonder verder in details te treden, willen we wijzen op de Amerikaanse tegenhanger van de CEPIS, namelijk de American Society for Quality (ASQ). Voor uitgebreidere informatie verwijzen we naar Hamilton (1998).

De 'Professional Development Scheme' (PDS) is een professionaliseringsprogramma dat door de British Computer Society (BCS) in het leven is geroepen om kwaliteitsbeheersing te bevorderen en via opleiding en scholing carrièremogelijkheden binnen de software engineering te bieden.
Enkele basiselementen van het PDS zijn:

- een 'Industry Structure model' waarin een uitgebreide beschrijving te vinden is van kwaliteitseisen die worden gesteld aan personen die werkzaam zijn binnen de software engineering
- een jaarlijkse inspectie door de BCS om na te gaan of de voorschriften worden nageleefd.

22.7 Conclusies

In dit hoofdstuk hebben we, net als bij behandeling van organisatieafhankelijke factoren, laten zien dat onder de noemer 'mensafhankelijk' talloze factoren te benoemen zijn waarvan aangenomen mag worden dat ze een belangrijke invloed hebben op de kwaliteit van software. Voor veel factoren geldt dat het verband aannemelijk is, maar voor weinige kan dat ook onderbouwd worden aangetoond. Dat neemt niet weg dat investeren in die factoren waarvoor zo'n relatie met softwarekwaliteit aannemelijk is, zinvol is. Vanuit de gedragspsychologie wordt bijvoorbeeld voldoende bewijs aangeleverd om ervan uit te aan dat motivatie en gedrag zondermeer zeer belangrijke beïnvloedende factoren zijn. De vraag 'hoe gedrag te beïnvloeden en hoe motivatie te verhogen?' blijft lastig te beantwoorden. Een eensluidend antwoord kan vooralsnog niet worden gegeven. Vanuit de psychologie is bekend dat zaken als:

- *verscheidenheid aan verlangde deskundigheid*
- *identiteit van de taak*
- *belangrijkheid van de taak*
- *autonomie*
- *terugkoppeling van de taak*

belangrijk zijn voor de motivatie van een professional en in het verlengde daarvan van een softwareontwikkelaar. Hoewel dit soort onderzoeksresultaten vaak worden herkend, blijkt de praktijk van alledag weerbarstiger te zijn en blijken ook zaken als salariëring en carrièrekansen belangrijke motivatieprikkels te zijn.

Anders ligt het bij zaken als opleiden en professionalisering. Hiervoor kunnen concrete acties worden afgesproken en het positieve effect ervan op softwarekwaliteit is duidelijker aantoonbaar. De eerste resultaten met toepassing van het opleidingsprogramma volgens het Personal Software Process-model van W. Humphrey zijn hoopgevend. Initiatieven zoals het L_PASO-model van Dietz en de IEEE (SWEBOK) om te komen tot een 'common Body of Knowledge' moeten het vakgebied software engineering helpen de weg naar volwassenheid te vinden.