Institut für Informatik
Freie Universität Berlin

Der Personal Software Process

... in einer didaktischen Betrachtung

für das Seminar *Ausgewählte Beiträge zum Software Engineering*' im WS 05/06 von
Prof. Dr. Lutz Prechelt

Frank@Schlesinger.com

Kurzfassung *Der „Personal Software Process" ist nicht nur eine Methode zur Verbesserung von Softwareprodukten und Softwareprojekten, sondern ebenfalls eine didaktische Methode, ein Hilfsmittel zum Lernen. Bei der Softwareentwicklung mit PSP-Methoden lernt der Programmierer seine Arbeit kennen, er lernt sie zu verbessern, er lernt softwaretechnisch besser zu arbeiten und disziplinierter und strukturierter vorzugehen. Anhand der Einordnung des PSPs in die „Berliner Didaktik" erläutere ich welche Lernziele bei der Benutzung des Prozesses erreicht werden können und beschreibe, wie sich die Automatisierung der PSP-Techniken durch PSP-Tools und das Hackystat System auf den Lerneffekt auswirken.*

Abstract *The „Personal Software Process" is not only a methodology to improve software products and software projects, but also a didactical method, a learning tool. When a programmer develops software with PSP methods, he learns about his own work, he learns to improve his work, he learns to work better from a software engineering point of view, and he learns to work more disciplined and structured. By bringing the PSP into terms of the „Berliner Didaktik", I explain what can be learned when using the Personal Software Process, and I describe the impact of automating the PSP techniques to this learning effect.*

1 Einleitung

Der *Personal Software Process* (PSP) ist eine ebenso bekannte wie nachgewiesen hilfreiche Sammlung von Methoden und Techniken zur Steigerung der Qualität von Softwareprodukten und der Genauigkeit von Softwareprojektplanungen. Auch aufgrund des relativ großen Aufwands bei dessen Benutzung durch den Softwareentwickler hat der PSP in der Softwareindustrie nahezu keine Bedeutung. Die Anhänger des PSPs unternahmen verschiedene Versuche, den Aufwand für den Programmierer zu reduzieren und so die Akzeptanz dieses Prozessrahmenwerks zu steigern. Angefangen bei einfachen Werkzeugen, die grafische Oberflächen zur Erfassung der PSP-Daten bereitstellen, bis hin zu Systemen wie der *Hackystat* Software, in der die Datenerfassung und -analyse voll automatisiert wurde, reicht die Palette an Unterstützung für den PSP.

Der Personal Software Process, wirkt sich aber nicht nur auf Projekte und Produkte aus, sondern auch auf den Programmierer, der mit ihm arbeitet. In der Literatur wird ein solcher Lerneffekt durch die Benutzung von PSP zumindest impliziert. Der Fokus liegt bei den bisherigen Veröffentlichungen aber immer auf das Lernen des PSPs in Kursen oder im Selbststudium. In dieser Arbeit möchte ich das Augenmerk hingegen auf das Lernen richten, das bei der Benutzung des Personal Software Process' stattfindet. Gerade unerfahrene Softwareentwickler können meiner Meinung nach viel durch die Arbeit mit dem PSP lernen. Dabei sehe ich jedoch die bisher nicht benannte Problematik, dass der Trend zur Automatisierung der Arbeitsabläufe des PSPs, der einen wichtigen Beitrag zur Steigerung seiner Akzeptanz leistet, sich negativ auf den Lerneffekt auswirkt. Um dies zu verdeutlichen werde ich den Personal Software Process zunächst in Hinblick auf die mit ihm zu erreichenden Lernziele untersuchen und diese dann in die Taxonomie der *Berliner Didaktik* einordnen. Anschließend werde ich betrachten, wie sich die benannten Lernziele verändern, wenn die Vorgänge des PSPs automatisiert werden. Meine Argumentation baut dabei auf die Erkenntnisse der konstruktivistischen Lerntheorie, die ich ebenfalls kurz vorstellen werde.

Mit dieser Arbeit möchte ich nicht die Automatisierung des PSPs durch Tools oder gar das Hackystat Projekt in Frage stellen, an dem ich selbst partizipiere. Ich möchte aber zeigen, dass automatisierte PSP-Verfahren nicht die beste Wahl sind, wenn es um das Erlernen guter softwaretechnischer Tugenden und projektplanerischer Fähigkeiten geht.

2 Der Personal Software Process (PSP)

Watts S. Humphrey, Professor am Software Engineering Institute (SEI) der Carnegie Mellon University, begann 1989 damit, über mehrere Jahre hinweg seinen eigenen Softwareentwicklungsprozess zu dokumentieren und zu verbessern. Bei der Arbeit an über einhundert Programmen in dieser Zeit passte er das vom SEI zuvor entwickelte *Code Maturity Model* so an, dass er im Ergebnis seinen persönlichen, optimierten Softwareprozess erhielt. Die Ergebnisse dieses „Selbstversuchs" veröffentlichte er 1995 in [Humphrey 1995] unter dem Namen Personal Software Process. Der Leitgedanke hinter dem PSP ist es, die Qualität von Softwareprodukten und die Abschätzungen über Entwicklungszeiten und -aufwand zu verbessern, indem der Softwareentwicklungsprozess verbessert wird. PSP ist „*a roadmap for a disciplined approach to software development*" [Hirmanpour 2000]. Der Personal Software Process ist ein Prozessrahmenwerk, das erst durch den Softwareentwickler ausgestaltet wird. Die vom PSP bereitgestellten Techniken und Standards helfen dem Programmierer seine Projekte besser zu planen und in Bezug auf Zeit, Ressourcen und Größen besser abzuschätzen. Der PSP bietet auch Methoden an, um

die Produktqualität zu steigern.

Das Rückrat des PSPs ist eine Sammlung von Formularen zur Erfassung von Daten des Programmierprozesses und Metriken der Softwareprodukte. Arbeitszeiten und Unterbrechungen des Programmierers, aber vor allem auch gefundene Defekte werden mit diesen Formularen durch den Programmierer erfasst. Aus diesen Daten werden nach vorgegebenen Verfahren Analysen erstellt, deren Ergebnisse sich wieder auf den Prozess auswirken. Die dokumentierten Arbeitszeiten etwa werden benutzt um zukünftig genauere Abschätzungen über die zu investierende Arbeitszeit vorzunehmen. Aus den dokumentierten Defekten werden Prüflisten erstellt, die bei vorgeschriebenen Codereviews benutzt werden, um individuell typische Defekte zu finden.

2.1 PSP hilft den Produkten und den Projekten

Um zu ermitteln wie sich die Benutzung des Personal Software Process' auf Softwareentwicklungen auswirkt, wurden verschiedene Studien erstellt. Zunächst sind da diejenigen zu nennen die vom SEI selbst durchgeführt wurden. Diese machen teils deskriptive Aussagen [Humphrey 1996] und [Ferguson 1997] oder sind statistisch ausgewertet [Hayes 1997]. Mit [Weslén 2000] existiert aber auch eine statistisch ausgewertete Studie außerhalb des SEIs. Alle diesen Studien, ob sie aus PSP-Kursen, Fallstudien in der Softwareindustrie oder aus Seminaren an Universitäten stammen, zeigen eine Abnahme der Defektdichte mit zunehmenden Einsatz der Methoden des PSPs. Außerdem werden die Abschätzungen der Programmierer, die PSP-Techniken einsetzen, in Bezug auf den Entwicklungsaufwand und die Entwicklungszeit genauer.

2.2 Das Akzeptanzproblem

Es ist erstaunlich, dass es keine Anzeichen dafür gibt, dass der Personal Software Process in der Praxis, in der Softwareindustrie, tatsächlich eingesetzt wird. Philip Johnson [Johnson 2002] schildert seine Erfahrung, nach der Teilnehmer eines PSP-Kurses das gelernte nicht weiter verwenden würden, wenn der Kurs abgeschlossen ist. Johnson verweist auch auf [Borstler 2002]. Dort wird von einem PSP Seminar mit 78 Teilnehmern berichtet. Nach Abschluss des Kurses gaben 72 Teilnehmer an, den Personal Software Process nicht weiter benutzen zu wollen. Nach dem Grund dafür gefragt, nannten sie dessen strikte und einengende Natur. Von den übrigen sechs Teilnehmern des Kurses ist nicht bekannt, ob sie den PSP weiter einsetzen. Johnson sieht hauptsächlich zwei Gründe für die mangelnde Akzeptanz des PSPs. Zum einen sei der „*overhead*" zu groß. Der Verwaltungsaufwand des PSPs mit der kontinuierlichen Erfassung von Daten und der Auswertung und Analyse ist tatsächlich hoch. Als zweiten Grund hat Johnson den im PSP nötigen „*context-switch*" erkannt. Die Notwendigkeit die eigentliche Arbeit, das Programmieren, immer wieder zu unterbrechen, um auf der Metaebene zu arbeiten und Daten aufzuschreiben, sei schlicht lästig. Prechelt nennt in [Prechelt 1997] die Dinge bei einem etwas anderen Namen und spricht von einem Mangel an Selbstdisziplin vieler Entwickler. Manche seien einfach nicht in der Lage einen so disziplinierten bzw. disziplinierenden Prozess wie den PSP zu benutzen. In diesem Zusammenhang ist auch die Studie von Manfred Prenzel [Prenzel 1993] interessant. In drei Untersuchungen führte er Befragungen von Erwachsenen durch, die entweder an auto-didaktischen Lernprogrammen teilgenommen haben oder die Studenten der Medizin bzw. der Erziehungswissenschaften waren. Ziel seiner Studie war es den Zusammenhang zwischen „*Autonomie und Motivation im Lernen Erwachsener*" zu untersuchen, und er kam zu folgendem Ergebnis: „*Maßgeblich ist die von der Person erlebte Au-*

tonomie, die sich auf die Bestimmung der Lernziele, auf die Lernkoordination und auf die Lernorganisation bezieht. Das subjektive Erleben von Autonomie ist eng verknüpft mit der Motivation, sowohl mit der sachbezogenen intrinsischen Motivation als auch mit Formen der extrinsischen Motivation. Beide Varianten von Motivation sind relevant für selbstbestimmtes, intentionales Lernen. Inwieweit sich eine Person als autonom erlebt, hängt von den Bedingungen der sozialen Umgebung ab: vom Ausmaß, in dem sich die Person als sozial eingebunden fühlt und in ihrer Autonomie und Kompetenz unterstützt sieht. [...] Unter einer massiven Einschränkung und Kontrolle, unter Bedingungen eines ständigen Vorschreibens und fehlender sozialer Akzeptanz kann sich eine Person nicht als autonom, kompetent und sozial eingebunden erleben." Die gefühlte Enge bei der Benutzung des Personal Software Process' und die kontrollierenden Elemente müssten sich demgemäß negativ auf die Motivation des Softwareentwicklers auswirken.

2.3 Unterstützung für den PSP

Als überzeugter PSP-Lehrer und -Anwender wollte Johnson wie andere auch die Benutzbarkeit des PSPs vereinfachen und seine Verbreitung fördern. Mittlerweile existiert eine große Anzahl von PSP-Werkzeugen, welche die Datenerfassung vereinfachen und die Auswertungen automatisieren. Sie bieten dem Programmierer meist kleine grafische Benutzungsoberflächen an, in der er durch Knopfdruck Arbeitszeiten erfassen oder Defekte protokolliert kann. Mit der Software Leap [Johnson 1999], die eine ebensolches PSP-Werkzeug ist, machte Johnson allerdings auch keine zufrieden stellenden Erfahrungen. Die Akzeptanz des PSPs bei der Benutzung von Leap sei zwar etwas größer. Allerdings neigen Softwareentwickler dazu gerade in stressigen Situationen und unter Zeitdruck auch diese Werkzeuge nicht mehr zu benutzen. „*You can't even ask them to push a button*" ist gleichzeitig Titel wie auch Fazit der Arbeit [Johnson 2001]. Für das weitere Vorgehen zog er daher den Schluss, dass die Datenerfassung vollständig automatisiert werden müsse. Humphrey schreibt dazu in [Humphrey 1995]: „It would be nice to have a tool to automatically gather the PSP data. Because judgment is involved in most personal process data, however, no such tool exists or is likely in the near future." Mit dem Hackystat System [Johnson 2002] entwickelten Johnson und seine Mitarbeiter nichtsdestotrotz eine solche Software, deren wichtigste Eigenschaften es sind „*non-disruptive*" zu sein und die Daten „*in-process*" zu erfassen. Das Hackystat System besteht aus Sensoren, die in den Softwareentwicklungs-Werkzeugen des Programmierers automatisch Daten erfassen. Die gesammelten Daten werden von den Sensoren dann an den zentralen Hackystat Server gesendet, wo sie ausgewertet und grafisch aufbereitet werden. Der Programmierer hat die Möglichkeit sich die Auswertungsergebnisse seiner Daten auf dem Hackystat Server anzusehen.

Das Hackystat System löst die Probleme des „overhead" und des „context-switch", bringt dafür aber neue mit. Die wichtigste Einschränkung gegenüber dem klassischen PSP ist es, dass die Natur der erfassten Daten eine andere ist als jene, die im PSP erfasst werden sollen. Besonders deutlich wird dies bei den Defekten. Für den Personal Software Process sind die vom Programmierer gefundenen Defekte zu protokollieren. Sie werden klassifiziert und die Zeit, die zu ihrer Beseitigung benötigt worden ist, wird vermerkt. Defekte findet der Programmierer noch vor dem Übersetzen durch Reviews. Diese Defekte können von den Hackystat Sensoren nicht erfasst werden. Sie können natürlich keine Reviews für den Programmierer durchführen und wissen auch nichts von Defektklassen oder dem Aufwand zu Defektbehebung. Im Hackystat System werden Defektdaten daher lediglich mittels eines JUnit-Sensors aus fehlgeschlagenen Testfällen und mittels eines ANT-Sensors aus fehlgeschlagenen Bild-Vorgängen ermittelt. Für Philip Johnson ist dieser

Characteristic	Generation 1 (manual PSP)	Generation 2 (Leap, PSP Studio, PSP Dashboard)	Generation 3 (Hackystat)
Collection overhead	High	Medium	None
Analysis overhead	High	Low	None
Context switching	Yes	Yes	No
Metrics changes	Simple	Software edits	Tool dependent
Adoption barriers	Overhead, Context-switching	Context-switching	Privacy, Sensor availability

Figure 1. Three generations of approaches to metrics for individuals

Abbildung 1: Entnommen aus [Johnson 2002]

Unterschied aber kein Nachteil. Bereits im Titel seiner Arbeit [Johnson 2002] „*Beyond the Personal Software Process*", suggeriert er, dass Hackystat eine Weiterentwicklung des PSPs sei. Neben der vollständigen Automatisierung der Datenerfassung und -auswertung sei Hackystat auch zuverlässiger als die manuelle Datenerfassung durch den Programmierer. Automatische Sensoren seien nicht faul und auch nicht beeinflussbar. Die erfassten Daten seien somit wirklich objektiv. Darüber hinaus steigt durch die Benutzung des Hackystat Systems die Qualität der erfassten Daten. Johnson hat in einer Untersuchung herausgefunden, dass manuell erfasste PSP Daten zum Teil ungenau oder falsch aufgeschrieben werden.

3 PSP als didaktische Methode

In den vorhandenen Veröffentlichungen finden sich Hinweise auf einen Lerneffekt, der dem PSP zugestanden wird. In [Prechelt 2001] wird der Prozess als „*One successful approach to improving individual learning*" beschrieben. Prechelt schreibt, dass Programmierer zwar immer aus ihren Erfahrungen und vor allem aus ihren Fehlern lernen würden, der Personal Software Process könne diesen Lernprozess aber beschleunigen. Philip Johnson meint in [Johnson 2002]: „*... studying the PSP is similar to studying Latin: a task that advocates suggest you learn for its indirect benefits, rather than because you'll actually use it in your daily life.*"

Ich möchte im Folgenden den Personal Software Process einer didaktischen Betrachtung unterziehen. Dabei geht es mir nicht darum das Erlernen des PSPs zu untersuchen. Vielmehr möchte ich beschreiben, was durch die Benutzung der Methoden aus dem PSP gelernt werden kann. Der Begriff Didaktik bedarf dabei zunächst einer Erläuterung. Im engeren Sinne bezeichnet er das konkrete, methodische Vorgehen des Lehrenden in einer Unterrichtssituation und wird in dieser Weise oft mit dem Begriff der Unterrichtsmethodik synonym verwendet. Im weiteren Sinne bezeichnet der Begriff Didaktik ein Teilgebiet der Erziehungswissenschaften, welches sich mit dem Lernen und Lehren auseinandersetzt und als solches auf eine lange Geschichte bis in die griechische Antike zurückblicken kann. Im letzten Jahrhundert war die Didaktik geprägt durch die Konkurrenz verschiedener wissenschaftstheoretischer Strömungen. So gründeten einige Didaktiker ihre Theorien auf den Positivismus und lieferten sich

heftige Kontroversen mit den Anhängern der kritischen Theorien. Solche Grabenkämpfe sind in der Didaktik heute überwunden und haben unter Anderem eine große Zahl von Taxonomien, Methoden und Planungsrastern hinterlassen, die heute von den Lehrenden vor allem nach pragmatischen Gesichtspunkten ausgewählt und für ihren Unterricht eingesetzt werden.

3.1 Die Berliner Didaktik

Eine dieser Hinterlassenschaften ist die so genannte *Berliner Didaktik* oder *Lehr- und Lerntheoretische* Didaktik. Konzipiert wurde sie Anfang der 60ger Jahre von Paul Heimann und seine Kollegen Gunther Otto und Wolfgang Schulz, die als Hochschullehrer und Seminarleiter an der Pädagogischen Hochschule Berlin (West) arbeiteten. Die Berliner Didaktik zeichnet sich durch ihre Ideologiefreiheit aus. So verzichtet sie beispielsweise auf einen expliziten Bildungsbegriff. Sie ist nicht normativ konstruiert, sondern von Heimann aus vielen Unterrichtsbeobachtungen heraus empirisch gewonnen worden. Heimann, Otto und Schulz wollten zunächst nur ein Mittel an die Hand bekommen, um den Unterricht ihrer Studenten systematisch analysieren zu können. Sie wollten die verschiedenen Gesichtspunkte der Lehre klar strukturieren, voneinander abgrenzen und in Beziehung setzen können. Genau diese Analysefähigkeit der Berliner Didaktik möchte ich mir bei der Betrachtung des PSPs zu nutze machen. Erst später erkannten Heimann und seine Kollegen, dass sie mit dem Analyseraster auch ein vorzügliches Mittel zur Unterrichtsplanung geschaffen hatten.

3.2 Intentionalitäten

Eines der vielen Systeme der Berliner Didaktik dient der Klassifizierung von Intentionalitäten des Lehrenden. Zu diesen pädagogischen Absichten gehören zunächst die so genannten *Lernziele*. In ihnen ist Beschrieben, was die Lernenden aus dem Unterricht mitnehmen, was sie gelernt haben sollen. Obwohl Heimann die Intentionalitäten bewusst nicht auf die Lernziele festgelegt hat, um auch andere Absichten des Lehrers darin fassen zu können, werde ich mich im Folgenden auf die echten Lernziele beschränken. Nach der Berliner Didaktik gibt es drei Dimensionen auf denen Lernziele angesiedelt sein können. Dabei sind sie in jeder der drei Dimensionen noch in drei Stufen unterteilt. Lernziele niedrigerer Stufen sind leichter zu erreichen als solche höherer Stufen. Die Lernziele höherer Stufen sind aber höherwertig in dem Sinne, dass sie sich tiefer im Lernenden festsetzen. Die erste Dimension der Lernziele ist die

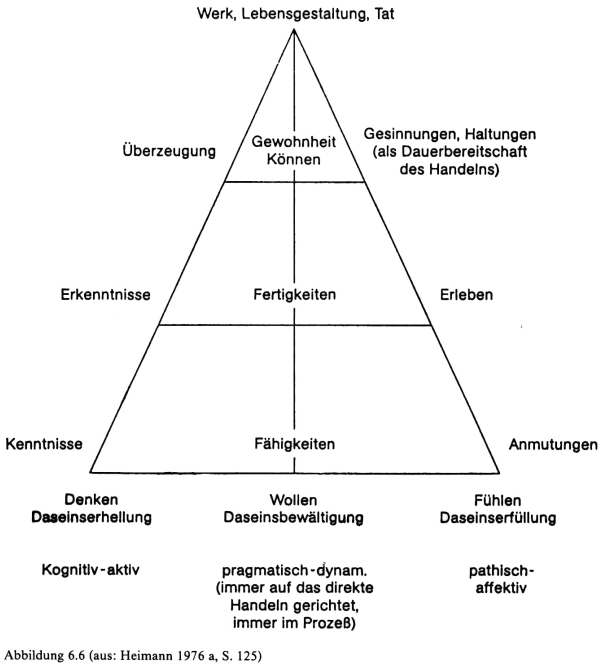

Abbildung 6.6 (aus: Heimann 1976 a, S. 125)

Abbildung 2: Entnommen aus [Meyer 1994]

kognitiv-aktive Dimension. Sie umfasst die auf das Denken bezogenen Absichten des Lehrers. Hier werden Lernziele versammelt, die eine Wissensvermittlung und andere kognitive Prozesse beinhalten. In der ersten Stufe werden hier Kenntnisse vermittelt. Der Lernende hat etwas zur Kennt-

nis genommen. Er kennt es nun. Auf der zweiten Stufe dieser Dimension ist die Erkenntnis über einen Wissenszusammenhang zu finden. Für eine Erkenntnis muss im Gegensatz zur Kenntnis auch eine Reflexion durch den Lernenden stattfinden. Hier kann gelerntes Faktenwissen beurteilt und in sinnvolle Zusammenhänge gebracht werden. Die höchste Stufe der kognitiv-aktiven Dimension ist die Überzeugung im Sinne einer rationalen Prinzipienbildung.

Die zweite Dimension ist die *pragmatisch-dynamische*, die auf das Handeln gerichtet und auf das Können der Schüler bezogen ist. Im einfachsten Fall werden hier Fähigkeiten vermittelt, die in der nächsten Stufe durch ständiges Einüben zu Fertigkeiten ausgebildet werden können. Die höchste Stufe dieser Dimension ist dabei das Können als das in Gewohnheit übergegangene Handeln.

Die dritte und letzte Dimension umfasst alle *pathisch-affektiven* Aspekte, zielt also darauf ab, dem Lernenden Erlebnisse und Eindrücke zu verschaffen und ihn Erfahrungen machen zu lasen. Im einfachsten Fall bekommen die Lernenden eine Anmutung über einen Unterrichtsinhalt. Ein Erlebnis ist dann schon viel beeindruckender und kann sich auch nachhaltig manifestieren. Die höchste pathisch-affektive Stufe ist die so genannte Gesinnung oder Haltung, die eine ständige affektiv-moralische Bereitschaft zum Handeln darstellt.

Nach Heimann ist dabei eine höhere Stufe ein Fortschritt gegenüber einer niedrigeren Stufe. Er spricht vom Gesetz der dimensionalen Bereicherung. In [Heimann 1976a] wird dafür folgendes Beispiel gegeben: „*GeschichtslehrerInenn sollen sich z.B. überlegen, ob ihre Schülerinnen und Schüler das Datum des 23. Mai 1949 als Tag der Verkündung des bundesrepublikanischen Grundgesetzes lediglich zur Kenntnis nehmen sollen (kognitiv-aktive Dimension, niedrigste Stufe), ob sie über die Kenntnis dieses Datums und anderer dazugehörender historischer Einzelkenntnisse hinaus auch Ursachen und Folgen verstehen (und nicht nur aufzählen können), in ihren Zusammenhängen begreifen und die zugrundeliegenden Strukturen des historischen Ablaufs erkennen sollen (Erkenntnisse; 2. Stufe), oder ob sie, weiter darüber hinaus, selbständiges Interesse an diesem historischen Geschehen entwickeln können und sich auch über den schulischen Rahmen hinaus damit beschäftigen sollen (Überzeugung; 3. Stufe).*"

3.3 PSP und die Lernziele

Der Personal Software Process besteht aus einer Vielzahl von Methoden und Vorschriften, die alle einzeln und unabhängig voneinander unter didaktischen Gesichtspunkten betrachtet werden könnten. Im Rahmen dieser Arbeit möchte ich mich dem PSP aber im Ganzen nähern. Dabei gehe ich zunächst von einer Benutzung des PSPs aus, bei dem die einzelnen Techniken voll-manuell vom Lernenden ausgeführt werden. Ob der Programmierer seine Daten handschriftlich oder in einer Tabellenkalkulationssoftware erfasst spielt für meine Betrachtung allerdings keine Rolle.

Wenden wir uns zunächst der kognitiv-aktiven Dimension der Lernziele zu. Der Programmierer erfasst im PSP während seiner Arbeit ständig Daten über sein eigenes Vorgehen. Angefangen von den Arbeitszeiten und den Unterbrechungen über die Größe der Produkte bis hin zu den Defekten die er findet und der Zeit die er zu ihrer Beseitigung aufbringen musste, kann der Programmierer *Kenntnisse* erlangen, die ihm ohne den PSP verwehrt bleiben würden. In den meisten Fällen würde er sich einfach nicht die Mühe machen diese Daten zu erheben. In allen PSP-Studien wurde von einer Verbesserung der Abschätzungsfähigkeiten der Programmierer berichtet, wenn diese eine Weile mit dem Personal Software Process arbeiteten. Hier hat also ein *Erkenntnisprozess* stattgefunden. Ein *Erkenntnisgewinn* wird auch bei den Defekten erreicht. Der Entwickler erkennt, welche Defekte er besonders häufig macht und kann den Auf-

wand für deren Beseitigung in ein Verhältnis zu diesen Defekten setzten. Idealerweise *vermeidet* er zukünftig besonders aufwändig zu beseitigende Defekte im *Vorhinein*. In diesem Sinne verhilft der PSP dem Programmierer zu *Selbsterkenntnissen* und bildet die Grundlage für eine Selbstverbesserung.

Wenn die Erfassung der PSP-Daten durch den Programmierer geschieht, so kann er aber auch einen *Eindruck* von seinen Daten bekommen, noch bevor er diese ausgewertet hat. Durch das Aufschreiben der Unterbrechungen bekommt er ein *Gefühl* dafür, zu welchen Zeiten er wodurch unterbrochen wird. Durch das Dokumentieren der gefundenen Defekte bekommt er einen *Eindruck* von der Anzahl und den Typen der Defekte. Im Heimann'schen Sinne kann der PSP *Anmutungen* über die eigene Arbeitsweise erzeugen. Erlebnisse sind von einem so technischen Vorgang wie der Programmierung mit dem PSP nicht zu erwarten. Ich benutze daher hier den Begriff Eindruck, der gegenüber einer Anmutung auch schon eine Nachhaltigkeit besitzt. Der Programmierer kann durch den PSP einen *Eindruck* von seiner Arbeit und von seinen Schwächen und Fehlern bekommen, um sich verbessern zu können. Er kann ebenfalls einen *Eindruck* von der Qualität seiner Software bekommen. Er macht *Erfahrungen* mit sich und mit seinen Produkten. Es kann sich daraus beispielsweise die *grundlegende Haltung* entwickeln ständig mit dem PSP oder überhaupt disziplinierter, strukturierter und qualitätsbewusster zu arbeiten.

Auch im Bereich der dritten Dimension, der pragmatisch-dynamischen, sehe ich durch den PSP erreichbare Lernziele. Denn es werden durch PSP-Methoden auch konkrete Arbeitstechniken *eingeübt*. Ressourcen- und Aufwandabschätzungen, Zeitmanagement und Reviews sind *Fähigkeiten* die der Programmierer bei der Benutzung des PSPs lernen kann. Durch den Personal Software Process kann ein Entwickler zudem ein strukturiertes und diszipliniertes Vorgehen *einüben*. Werden diese Techniken zu *Fertigkeiten* ausgebildet, so können sie auch als nachhaltiger Gewinn aus der Beschäftigung mit dem PSP verbleiben. Im Idealfall werden die Disziplin und die Struktur zur *Gewohnheit*.

3.4 Auswirkung der Entwicklung auf die Lernziele

Wenn die Datenerfassung und die Auswertung nicht vom Programmierer selbst ausgeführt werden, hat das trivialer weise Auswirkungen auf die Lernziele, die in der pragmatisch-dynamischen Dimension erreicht werden könne. Der Lernende setzt die Techniken des PSPs selbst nicht mehr ein. Er überlässt die Analyse seiner Daten den PSP-Werkzeugen oder er benutzt das Hackystat System und bleibt selbst von der Datenerfassung vollständig unbehelligt. Er übt die Techniken wie Zeiterfassung und -planung selbst nicht mehr ein. Der PSP hilft seinem Benutzer ein strukturiertes Vorgehen bei der Softwareentwicklung zu erlernen. Der Einsatz von PSP-Werkzeugen wirkt weit weniger disziplinierend und strukturierend, der Einsatz des Hackystat Systems hat überhaupt keinen Einfluss mehr auf das Arbeitsverhalten des Programmierers. Die genannten Einschränkungen der handlungsbezogenen Lernziele, die mit der Automatisierung einhergehen, können natürlich nicht als Nachteil angesehen werden, wenn der Softwareentwickler bereits über derartige Fähigkeiten verfügt.

Mit fortschreitender Automatisierung der PSP-Techniken, nimmt die individuelle Auseinandersetzung des Programmierers mit seinen Daten ab. *Eindrücke* und *Erfahrungen* mit seiner eigenen Arbeit und mit seinen Produkten können mit dem Einsatz von PSP-Werkzeugen nur noch sehr eingeschränkt gemacht werden. Der Programmierer kann zwar nach wie vor seine Daten *kennen lernen*. Ein *Gefühl* für diese wird er dann allerdings nicht bekommen.

Es ist außerdem zu bezweifeln, ob der Program-

mierer im gleichen Maße zu *Erkenntnissen* über seinen Prozess und über seine Produkte gelangen kann, wenn die Daten halb oder voll automatisch aufgezeichnet und ausgewertet werden. Der *Erkenntnisgewinn* ist zwar ein Lernziel der aktiv-kognitiven Dimension, nach Heimann sind die drei Dimensionen aber gerade in den höherwertigen Stufen verknüpft. Der Lernende kann demnach nur schwer zu *Erkenntnissen* gelangen, wenn er nicht auch *affektiv* angesprochen wird und vor allem, wenn er nicht auch *handelt*.

Diese grundsätzliche Überzeugung Heimanns gründet sich auf die *konstruktivistische Lerntheorie*, die eine sehr enge Verknüpfung zwischen dem Handeln und dem Lernen sieht. Nach Heimann ist Lernen immer eine Form des Handelns. „*Danach ist für uns auch der Lernende ein Handelnder!*" [Heimann 1976a].

Als Vater der konstruktivistischen Lerntheorie gilt Jean Piaget, der als erster dem Lernprozess mit empirisch-analytischen Mitteln nachging und zu Beeindruckenden und auch heute noch gültigen Ergebnissen über den Lernprozess von Kindern kam. Die Kernaussage dieser Lerntheorie ist es, dass sich jeder sein Wissen selbst zusammenbaut, es konstruiert. Niemand kann also einfach für einen Anderen lernen und quasi fertiges Wissen in einen Anderen eintrichtern. Je intensiver und je ganzheitlicher die individuelle Beschäftigung mit dem Lernstoff ist, desto leichter kommt der Lernende zu kognitiven Lernerfolgen.

4 Fazit

Der Personal Software Process ist nicht nur eine bekannte hilfreiche Methode zur Steigerung der Softwarequalität und Verbesserung von Projektplanungen, er ist auch eine didaktische Methode. Als solche hilft er dem Programmierer Erkenntnisse über sein eigenes Arbeitsverhalten zu gewinnen. Erkenntnis ist bekannter maßen der erste Weg zur Besserung. Die Verbesserung der eigenen Arbeitstechniken, das Einüben von Disziplin und das strukturierte Arbeiten sind für mich die wichtigsten Lerneffekte, die sich durch die Benutzung des PSPs ergeben. In diesem Sinne unterstütze ich die Johnson'sche Latein-These: PSP lernen ist wie Latein lernen. Die gelernten Strukturen und Techniken helfen weiter, auch, wenn der Prozess selbst nicht mehr benutzt wird. Durch seine eigene Ganzheitlichkeit, mit welcher der PSP das Lernen auf allen drei Heimann'schen Dimensionen unterstützt, ist er eine sehr effektive handlungsorientierte Lernmethode. Diese ist allerdings mühevoll. Der „overhead" und der „context-switch" und die gefühlte Enge führen zu einem Akzeptanzproblem, dass durch automatische PSP-Tools überwunden werden kann. Der „overhead" im Sinne einer Auseinandersetzung und Beschäftigung mit den Daten und der „context-switch" im Sinne einer Bewusstmachung und Reflexion über die Daten sind für den Lerneffekt durch den PSP aber notwendig. Wenn, wie im Hackystat System gar keine Auseinandersetzung mit Daten, sondern höchstens eine Betrachtung dieser, statt findet, ist der PSP-Lerneffekt nicht mehr vorhanden. Darum erweitere ich die Latein-These: PSP lernen ist wie Latein lernen und es ist genauso mühevoll. Es gilt also abzuwägen und zu entscheiden zu welchem Zweck man sich als Softwareentwickler mit dem Personal Software Process beschäftigen möchte. Geht es dem erfahrenen und professionellen Entwickler darum, Einblicke in seine Projekte und Produkte zu bekommen, die Qualität messen zu können oder Maße aufzuzeichnen und zu analysieren, dann ist ein System wie Hackystat eine gute Wahl. Geht es dem Programmierer aber nicht nur um die Verbesserung seiner Produkte und Projekte, sondern um seine eigene Verbesserung, geht es ihm darum zu lernen softwaretechnisch besser zu arbeiten, dann sollte er die Techniken des PSPs selbst anwenden und erfahren. Die zentrale Aussage formulierte schon Konfuzius: „*Sage es mir und ich werde es vergessen, zeige es mir und ich werde mich erinnern, lass es mich selbst tun und ich werde es verstehen.*"

5 Referenzen

[Borstler 2002] J. Borstler, D. Carrington, G. Hislop, et al. Teaching PSP: Challenges and lessons learned. IEEE Software, 19(5), September 2002.

[Ferguson 1997] P. Ferguson, W. S. Humphrey, S. Khajenoori, et al. Results of applying the personal software process. IEEE Computer 30(5) 1996. S 24f.

[Humphrey 1995] W. S. Humphrey. A Discipline for Software Engineering. Reading, MA. Addison-Wesley 1995

[Hayes 1997] W. Hayes, J. W. Over. The Personal Software Process (PSP): An Empirical Study of the Impact of the PSP on Individual Engineers. Technical Report CMU/SEI-97-TR-001, ESC-TR-97-001. Software Engineering Institute. December 1997.

[Heimann 1976] P. Heimannn. Didaktische Grundbegriffe. In: Reich/Thomas (Hrsg.) 1976. S. 103f.

[Hirmanpour 2000] Iraj Hirmanpour, Soheil Khajenoori. Personal Software Process Technology: An Experiential Report. In Proceedings of ISECON 2000, v 17 (Philadelphia): pp. 115 (http://isedj.org/isecon/2000/115/)

[Humphrey 1996] W. S. Humphrey. Using a defined and measured personal software process. IEEE Software May 1996. S. 77f.

[Humphrey 1999] Humphrey, W.S., 1999, Introduction to Team Software Process, Addison Wesley, Reading, MA.

[Johnson 1999] Johnson. Leap: A „personal information environment" for software engineers. In Proceedings of the 1999 International Conference on Software Engineering, Los Angeles, CA., May 1999.

[Johnson 2001] Johnson. You can't even ask them to push a button: Toward ubiquitous, developer-centric, empirical software engineering. 2001

[Johnson 2002] Johnson, Kou, Agustin et al. Beyond the PSP: Metrics collection and analysis for the differently disciplined. Proceedings of the 2003 International Conference on Software Engineering. Portland, Oregon. May 2003

[Meyer 1994] W. Jank, H. Meyer. Didaktische Modelle. Berlin. Cornelsen Scriptor. 3. Aufl. 1994

[Paulk 1995] Paulk, M.C., C.V. Weber, B. Curtis and M.B. Chrissis, 1995, The Capability maturity Model: Guidelines for Improving the Software Process, Addison-Wesley, Reading, MA.

[Prechelt 1997] Experience Report: Teaching and Using the Personal Software Process (PSP). Submission to ESEC 1997. Online unter URL: http://page.inf.fu-berlin.de/ prechelt/Biblio/psp_experience.pdf

[Prenzel 1993] M. Prenzel. Autonomie und Motivation im Lernen Erwachsener. In: Zeitschrift für Pädagogik 1993, 39. Jg., H. 2, S. 239f.

[Weslén 2000] A. Weslén. A Replicated Empirical Study of the Impact of the Methods in the PSP on Individual Engineers. Empirical Software Engineering May 2000. S. 93f.