Pair Programming

Ausarbeitung im Rahmen des Seminars

Agile Softwareprozesse

Prof. Lutz Prechelt, Stephan Salinger

Freie Universität Berlin
Wintersemester 2004/05

von
Tilman Walther
walther@inf.fu-berlin.de

Abstract

Pair Programming ist durch die steigende Popularität von agilen Softwareprozessen verstärkt ins Blickfeld wissenschaftlicher Untersuchungen geraten. Dieser Artikel bietet einen Überblick über die Technik des Pair Programming. Die Vor- und Nachteile werden anhand von zwei Studien erörtert. Des Weiteren wird der Stand der Entwicklung bei der Sonderform des Distributed Pair Programming beschrieben. Abschließend wird an der Methodik geäußerte Kritik dargestellt und in den Kontext der Studien eingeordnet.

Basis-URL: http://www.tilman.de/uni/PairProgramming.pdf

Inhaltsverzeichnis

1 Einleitung

Pair Programming ist eine Methodik, die im Umfeld der agilen Softwareprozesse eine große Rolle spielt und in den letzten Jahren immer größere Verbreitung gefunden hat. Obwohl verschiedene Arten des Pair Programming schon immer in der Softwareentwicklung zum Einsatz kamen und episodenhaft von Fällen berichtet wurde, in denen zum Teil immense Verbesserungen der Entwicklungsleistungen beobachtet wurden, waren wissenschaftliche Untersuchungen zum Thema selten. Erst mit dem Durchbruch der Extreme Programming Methodik als dem prominentesten agilen Softwareprozess rückte auch die dort wichtige und stark reglementierte Technik des Pair Programming ab Mitte der neunziger Jahre verstärkt ins Blickfeld empirischer Bewertung. Einen Überblick über die verfügbaren Forschungsergebnisse liefern Müller e.a. in (1). Dieser Artikel erörtert die Konzepte der Paarprogrammierung und die dabei auftretenden Effekte anhand der Studien von Nosek (2) und Williams (3). Es sollte allerdings erwähnt werden, dass andere Studien zum Teil stark abweichende oder sogar gegenteilige Ergebnisse bezüglich der Vorteile von Pair Programming ergaben. In Abschnitt 5 wird darauf näher eingegangen. Des Weiteren wird die Sonderform des *Distributed Pair Programming* beschrieben und der Abeit in lokalen Paaren gegenübergestellt.

2 Konzept

Pair Programming ist eine Technik, die in der Softwareentwicklung, insbesondere im Umfeld agiler Softwareprozesse wie Extreme Programming eingesetzt wird, um die Softwarequalität zu erhöhen und den Entwicklungsprozess zu beschleunigen. Beim Pair Programming bearbeiten zwei Programmierer gemeinsam dasselbe Problem. Dabei teilen sie sich einen Computer, den sie abwechselnd bedienen. Die beiden zu besetzenden Rollen werden in der Terminologie des Extreme Programming mit Driver und Navigator bezeichnet. Der Driver bedient den Computer und programmiert aktiv. Dabei erläutert er seine Vorgehensweise dem Navigator, der die Arbeit des Drivers fortwährend kontrolliert. Da der Navigator mehr Abstand zur aktuellen Arbeit hat, kann er die angewendete Strategie zur Problemlösung überdenken, nachprüfen und verbessern.

Stellt der Navigator einen Fehler fest, oder ist ihm die Vorgehensweise des Drivers unverständlich, werden Probleme unmittelbar im Gespräch gelöst. Auf diese Weise werden insbesondere kleinere Fehler so früh wie nur möglich beseitigt, da der geschriebene Co-

de fortwährend begutachtet wird. Im Vergleich mit nachträglichen automatisierten Tests beschleunigt dies auf der einen Seite den Entwicklungsprozess, da Implementierung und Codedurchsicht in einem Arbeitsschritt stattfinden. Auf der anderen Seite erhöht dieser *Permanent Review* die Softwarequalität, da auch Fehler gefunden werden können, die in Tests eventuell nicht berücksichtigt würden. Allgemein lässt sich sagen, dass Fehler im Nachhinein wesentlich schwerer zu finden sind, was die Gefahr erhöht, dass sie erst im „freien Feld“ zum Tragen kommen, wo sie schnell exorbitante Kosten durch falsche Ergebnisse oder Ausfall der Software verursachen können.

Da Entscheidungen bezüglich der angewandten Strategie und des Programmdesign stets doppelt durchdacht sind, werden auch konzeptuelle Fehler reduziert. Durch das unterschiedliche Fachwissen und die verschiedenen Erfahrungen, die beide Partner in den Entwicklungsprozess einbringen, soll das Design immer auf der bestmöglichen Lösung beruhen. Um diesen Effekt zu unterstützen, werden in der Regel die Paare immer wieder neu zusammengesetzt; in der Extreme Programming Methodik werden beispielweise die Partner spätestens nach Abarbeiten einer *Story Card* gewechselt, so dass normalerweise nicht länger als acht Stunden mit demselben Partner gearbeitet wird. Auf diese Weise haben sämtliche Teammitglieder im Lauf der Zeit Kontakt mit allen Teilen der entwickelten Software, so dass das Projekt weniger abhängig von einzelnen Personen wird. Verlässt ein Mitarbeiter das Projekt, ist der von ihm geschriebene Programmcode immer noch anderen Mitarbeitern vertraut, die ihn pflegen und erweitern können. Diese Form der Absicherung stellt unzweifelhaft für viele Softwareprojekte einen nicht zu unterschätzenden Vorteil dar.

3 Empirische Untersuchungen

Nosek 1998: The Case for Collaborative Programming

John T. Nosek veröffentlichte 1998 die Ergebnisse einer Untersuchung, in der er die genannten Vorteile von Pair Programming (resp. *Collaborative Programming*) einer näheren Betrachtung unterzog (2). Er hatte aus 15 Systemprogrammierern einer Firma zwei Gruppen gebildet. Die Teilnehmer erstellten ein Programm zur Konsistenz-Prüfung einer Datenbank, dabei arbeitete zwei Drittel in Paaren, der Rest allein. Bis dahin war diese Aufgabe von externen Sachverständigen durchgeführt worden, da sie als sehr schwierig und essentiell für die Arbeit des Unternehmens angesehen wurde. Für die Arbeit hatten die Programmierer maximal 45 Minuten Zeit, im Anschluss wurden sie zu ihrer Lösung

und dem Arbeitsgefühl befragt.

Nosek evaluierte Lesbarkeit der Quellen, Funktionalität gemäß der Spezifikation und Bearbeitungszeit. Aufgrund der Befragung der Beteiligten wurden außerdem Werte für Vertrauen in das eigene Programm und Arbeitsgefühl erstellt.

Lesbarkeit und Funktionalität der Programme, die in Paaren erstellt wurden, waren deutlich besser als die der einzeln programmierten. Auch die Bearbeitungszeit betrug bei den Paaren im Mittel nur 71% im Vergleich zur Kontrollgruppe, allerdings bemerkt Nosek, dass dieser Wert aufgrund hoher Abweichung keine statistische Eindeutigkeit aufweist. Die Paare zeigten in der Einzelbefragung ein deutlich höheres Vertrauen in die erstellten Lösungen (was ja auch der gemessenen Funktionalität entsprach) und bewerteten das Arbeitsgefühl deutlich positiver als ihre allein programmierenden Kollegen. Nosek beurteilte die Ergebnisse der Studie als Beleg für die Vorteile von Pair Programming in Bezug auf Code-Qualität und Entwicklungsgeschwindigkeit.

Williams 2000: The Collaborative Software Process

Ein wesentliches Problem bei der Anwendung von Pair Programming ist die vermeintliche Verdoppelung der Arbeitszeit - schließlich muss jeder Arbeitsschritt von zwei Personen gemeinsam durchgeführt werden. Selbst die (statistisch nicht relevante) Verbesserung der Arbeitszeit in Noseks Untersuchung würde immer noch eine Erhöhung der Gesamtarbeitszeit um über 40% bedeuten. Und im Regelfall müsste man sogar mit einer Verdoppelung rechnen.

Nosek sieht deshalb Vorteile von Pair Programming hauptsächlich für Projekte, bei denen die Personalkosten weniger stark ins Gewicht fallen, weil eine stärkere Fokussierung auf Entwicklungszeitraum und/oder Fehlerfreiheit vorliegt. Allerdings würde dies das Spektrum von sinnvollen Einsatzmöglichkeiten für Pair Programming stark einschränken. Deswegen war in der von Williams 1999 durchgeführte Studie (3) eine der wesentlichen Fragen, um welchen Faktor sich die Arbeitszeit bei der Anwendung von Pair Programming tatsächlich erhöht.

Die Untersuchung wurde an der University of Utah mit 41 Studenten durchgeführt. Diese mussten innerhalb von sechs Wochen vier verschiedene Programme schreiben, dabei arbeiteten 28 Teilnehmer in Paaren, während die restlichen 13 die Kontrollgruppe bildeten. Die Funktionalität der Programme wurde mittels automatisierter Tests bewertet, die Arbeitszeit dokumentierten die Teilnehmer selbst über eine Webseite.

Die Studie bestätigte die Ergebnisse von Noseks Untersuchung in Bezug auf die verbesserte Funktionalität der entwickelten Software: Die Programme der Paare bestanden im Schnitt 14% mehr Tests (Abb. 1)[1], auch die Varianz war geringer als bei der Kontrollgruppe. Die Qualität der entwickelten Software war durch die Programmierung in Paaren also besser vorhersagbar.

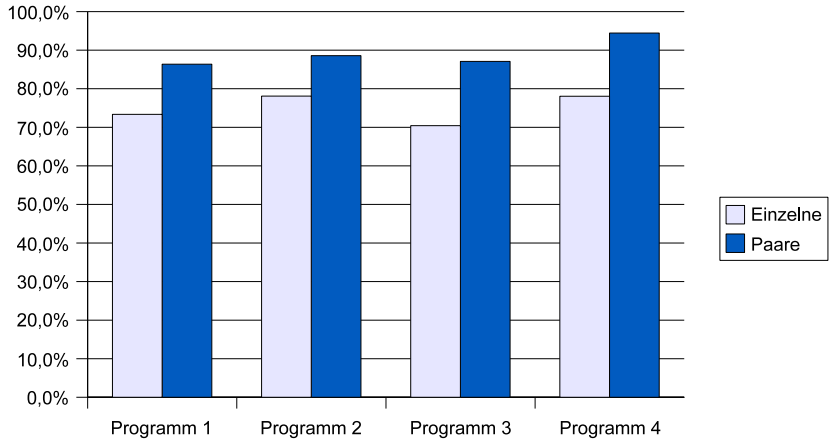

Abb. 1: ø bestandene Tests

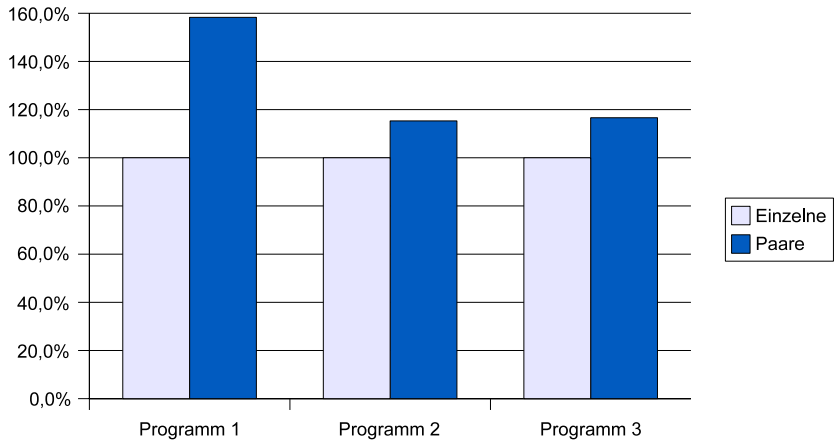

Abb. 2: ø Arbeitszeit pro Person

Der Vergleich der Arbeitszeit deutet sogar darauf hin, dass Pair Programming die Gesamtarbeitszeit wesentlich weniger erhöht als angenommen: Während die Paare zur Bearbeitung der ersten Aufgabe im Durchschnitt noch um 58% länger brauchten, sank die

[1] Alle Diagramme nach (3), (4)

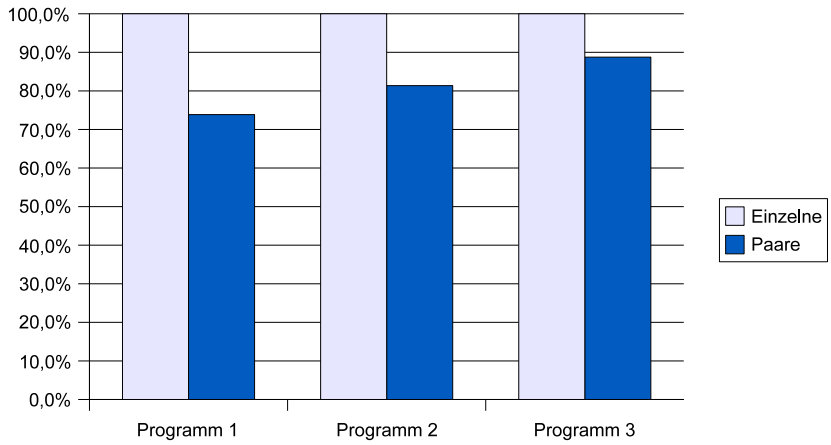

Abb. 3: ø Programmlänge

Arbeitszeit nach der Einarbeitungsphase (*Jelling Time*) rapide - ab der zweiten Aufgabe lag sie nur noch etwa 16% höher (Abb. 2).[2]

Auch war die Länge der in Paaren geschriebenen Programme im Schnitt 19% kürzer, was (zusammen mit den besseren Ergebnissen bei den Unit-Tests) auf eine gute Strukturierung hindeutet (Abb. 3).

Eine von Williams durchgeführte Befragung der Studenten, ergänzt durch eine Online-Umfrage unter professionellen Programmierern die Pair Programming betrieben zeigte, dass das Arbeitsgefühl sich durch die gemeinsame Arbeit verbesserte. Auch gaben die Teilnehmer ähnlich wie schon zuvor in Noseks Untersuchung an, dass sie ein höheres Vertrauen in das Ergebnis ihrer Arbeit hatten, wenn sie in Paaren programmierten.

In der Bewertung der Studie nennt Williams verschiedene positive Effekte, die im Pair Programming zu beobachten sind:

- *Pair-Pressure* - Durch die Anwesenheit des Partners wird die Konzentration auf die anliegende Aufgabe fokussiert, ein gedankliches Abschweifen ist wesentlich unwahrscheinlicher. Außerdem gaben die Programmierer an, dass sie durch den Partner motivierter waren Leistung zu zeigen. Durch die indirekte gegenseitige Kontrolle werden Vorgehensweisen und Standards während des Entwicklungsprozesses auch stärker eingehalten, die von einzeln Agierenden in Phasen erhöhten zeitlichen Drucks eher vernachlässigt werden.

[2] Allerdings ist hierbei zu bemerken, dass die Datenbasis für eine Aussage über die langfristigen Entwicklung der Arbeitszeit wohl kaum ausreicht, zumal von den vier Programmen auf Grund von Messungenauigkeiten nur die Arbeitszeit der ersten drei Eingang in die Studie fand.

- *Pair-Think* - Unterschiedliche Herangehensweisen und Erfahrungen der Partner führen zu unterschiedlichen Designvorschlägen, die gegeneinander abgewogen werden, was zu einem besseren Endergebnis führt.
- *Pair-Reviews* - Da der geschriebene Code unablässig durch eine zweite Instanz geprüft wird, werden Programmier- und Designfehler stark dezimiert.
- *Pair-Learning* - Die gemeinsame Arbeit unterstützt die Fortbildung der beteiligten Programmierer, da von beiden Seiten Lösungen eingebracht werden, die dem jeweiligen Parter eventuell noch unbekannt sind. Da der Driver sein Handeln durchgängig erklären muss, wird sowohl das Wissen um die Hintergründe des Designs als auch das Wissen um die verwendeten Techniken weitergegeben.

Allerdings bemerkt auch Williams, dass diese positiven Effekte nur auftreten, wenn die beteiligten Programmierer sich auf das Konzept des Pair Programming einlassen, die Entwicklung als Gemeinschaftwerk betrachten und bereit sind, die notwendige permanente Kommunikation zu unterhalten. In einem 2000 veröffentlichten Artikel legen Williams und Kessler eigens die Eigenschaften und Verhaltensweisen dar, die für gutes Pair Programming benötigt werden (5).

4 Distributed Pair Programming

Eine spezielle Form des Pair Programming ist das *Distributed Pair Programming* (DPP), bei dem die Paare von räumlich verteilten Entwicklern gebildet werden. Hierfür werden Werkzeuge benötigt, die eine virtuelle gemeinsame Arbeitsumgebung schaffen und die unmittelbare Kommunikation und gemeinsame Arbeit am Quelltext ermöglichen. In der Tat sind spezielle Werkzeuge für DPP noch selten, ein Anfang wurde mit dem Eclipse-Plugin Sangam[3] gemacht, das den Wechsel zwischen Driver und Navigator innerhalb des Java Development Tooling der Eclipse-IDE ermöglicht und sämtliche Eingaben über eine Datenverbindung synchron an zwei Computern vornimmt. Auf diese Weise werden alle Änderungen, die der Driver in der IDE vornimmt, unmittelbar für den Navigator sichtbar.

Unerlässlich für jede Form von Pair Programming ist die unmittelbare sprachliche Kommunikation zwischen den Teilnehmern. Deshalb werden eine dauerhafte Telefon- oder VoIP-

[3] http://sangam.sourceforge.net

Verbindung benötigt, die es ermöglicht, ohne Verlassen der Tastatur miteinander zu sprechen.

Der Erfolg von DPP steht und fällt mit der erfolgreichen technischen und geistigen Synchronisation der Partner. Da nonverbale Kommunikation durch die räumliche Trennung nicht mehr möglich ist, ist die unablässige Erläuterung der Arbeitsschritte durch den Driver noch wichtiger. In der Tat kommen auch Stotts, Williams e.a. in ihrer Studie zu DPP (6) zu dem Schluss, dass „Programmierer, die nicht gewillt sind, praktisch ununterbrochen zu sprechen“ mit DPP wohl keine zufrieden stellenden Ergebnisse erreichen werden. Auch stellten sie eine Vielzahl von Problemen mit den von ihnen eingesetzten Werkzeugen zur Sprach- und Datenkommunikation fest, die für sich genommen zwar nicht gravierend waren, zusammen genommen aber wohl durchaus den Unterschied zwischen Pair Programming und „häufigem Besprechen“ ausmachen können, was sich deutlich in der erreichten Produktivität niederschlägt. Insofern werden für einen breiteren Einsatz von DPP ausgereiftere Spezialwerkzeuge nötig sein, die

- das Fehlen der nonverbalen Kommunikation durch verschiedene Hilfmittel zur Kommunikation ausgleichen, bspw. „shared Whiteboards“
- weitgehend unabhängig von der verwendeten Programmiersprache sind
- auch Entwicklungsschritte unterstützen, die sich außerhalb der Entwicklungsumgebung abspielen

Des Weiteren werden wohl auch zusätzliche Eingabemittel wie Grafiktabletts oder Lichtgriffel nötig sein, die Zeichnungen und Skizzen zur Erläuterung von Ideen und Problemen ermöglichen. Durch die speziellen Bedürfnisse von Softwareentwicklern ist eine einfache Kombination von verfügbaren Werkzeugen wie Messengern und Remote-Desktop-Systemen auf Dauer wohl nicht ausreichend, da diese Hilfsmittel in der Regel auf *gleichzeitige* (Messenger) oder *einseitige* Aktion (Remote-Desktop-Systeme) ausgelegt sind. Die Eigenheiten des Driver-Navigator-Modells müssen auch durch die Arbeitsmittel unterstützt werden, dann können sie die positiven Effekte des Pair Programming eventuell sogar verstärken.

Nur wenn die Ausdrucksmöglichkeiten während des Arbeitens an das heranreichen, was den Partnern bei physischer Präsenz möglich ist, kann Distributed Pair Programming die Arbeitsmethoden von Entwicklern sinnvoll ergänzen.

5 Kritik

Pair Programming als Technik und die Studien zum Thema sind allerdings auch immer wieder kritisiert worden. Zunächst einmal ist die Übertragbarkeit der Studienergebnisse auf die Praxis in der Tat alles andere als gesichert: Die Datenbasis ist auch bei relativ aufwändigen Versuchsanordnungen vergleichsweise klein. Außerdem sind die Probanden in der Regel Studenten, die oft über weniger praktische Erfahrungen verfügen und deshalb, so die Kritik, stärker von einem Partner profitieren als ein erfahrener Profi. In der Tat ist Noseks Studie die einzige, die im Umfeld der Software-Industrie durchgeführt wurde.

Ebenfalls nicht abschließend geklärt ist die Einordnung von Spezialistenwissen in den Kontext von Pair Programming: Müssen Programmierer mit hoch spezialisiertem Fachwissen punktuell eingesetzt werden um bestimmte Probleme zu lösen, so lässt sich das Konzept des Pair Programming nicht konsequent anwenden.

Die recht beeindruckenden Ergebnisse aus Williams Untersuchung konnten bisher nicht vollständig bestätigt werden. In einem Versuch zum Pair Programming von Nawrocki und Wojciechowski (7) erzielten die einzeln arbeitenden Mitglieder der Kontrollgruppe sogar bessere Ergebnisse. Allerdings könnte, wie von Müller e.a. angemerkt, die Größe der Aufgabe hierbei eine Rolle spielen: Eventuell kommen die Vorteile der besseren gemeinsamen Übersicht über das gestellte Problem erst ab einer bestimmte Komplexität zum Tragen. Bestätigt wurde in dem Versuch dagegen die höhere Vorhersagbarkeit der gelieferten Qualität: Auch hier wiesen die Ergebnisse der Funktionstests bei der in Paaren entwickelten Software die geringste Varianz auf.

Sollte sich Williams' und Cockburns Ergebnis bestätigen, dass die Vorteile erst nach einer Einarbeitungszeit der Partner vollständig in Erscheinung treten (8), so würde dies bei kleineren Projekten bzw. größeren Teams dem häufigen Wechsel der Partner entgegenstehen, was die Frage nach einer konkreten Bemessung der Einarbeitungszeit aufwirft. Wäre die ständige Neubildung der Paare kontraproduktiv in Bezug auf die Arbeitszeit, so würde man sich die Verteilung der Wissenbasis durch einen Nachteil erkaufen. Ebenfalls nicht geklärt ist, welche Eigenschaften ein produktives Entwicklerpaar auszeichnen.

Ein anderer Kritikpunkt bezieht sich auf die sozialen Effekte des Pair Programming: Verschiedene Quellen (9; 10) berichten vom vereinzelten Auftreten von Problemen, die bestimmte Personen mit Pair Programming haben. Es scheint, als gäbe es einen kleinen aber relativ konstanten Anteil von Programmierern, die sich mit dem Konzept des Pair Programming unwohl fühlen und keinen Nutzen daraus ziehen können. So ist auch einer

der schärfsten Kritikpunkte in Stephens und Rosenbergs Polemik (11) gegenüber der Extreme Programming Methodik, dass sie Pair Programming zwingend vorschreibt. Auch die Teilnehmer von Williams' Studie äußerten zu 74% die Ansicht, das sie getrenntes Arbeiten zumindest für bestimmte Problemstellungen (Prototyping, Analyse schwieriger Probleme, u.a.) durchaus für sinnvoll halten (3, S. 175). Insofern wäre auch hier eine genauere Analyse der Vor- und Nachteile von Pair Programming für bestimmte Aufgaben wünschenswert.

6 Zusammenfassung

Pair Programming zeigt in einem geeigneten Umfeld deutliche Vorteile gegenüber der Bearbeitung von Problemen durch einzelne Programmierer. Auch wenn die gravierenden Unterschiede zugunsten der Arbeit in Paaren, wie sie von Williams in (3) dargelegt wurden, sich bisher nicht durchgängig bestätigen ließen, so sind bestimmte Vorteile nicht von der Hand zu weisen. Die konkreten Kosten für die Durchführung von Pair Programming lassen sich allerdings nach wie vor nur schwer abschätzen.

Unstrittig ist, dass die Technik des Programmierens in Paaren den Teilnehmenden einiges an Disziplin und Motivation zur gemeinsamen Arbeit abverlangt, damit die spezifischen Vorteile zum Tragen kommen. Nur wenn die Bereitschaft und Fähigkeit zum unablässigen Dialog zwischen den Entwicklern vorhanden sind, kann Pair Programming funktionieren. Dies gilt insbesondere für das noch wenig untersuchte Distributed Pair Programming. Wenn spezialisierte Werkzeuge für DPP vorhanden sind, ergeben sich allerdings auch Chancen für die Evaluierung der Möglichkeiten von Pair Programming insgesamt, da sie eine exaktere Messung von Kommunikationsverhalten und Produktivität ermöglichen.

Nach wie vor behält Noseks Schlussfolgerung Gültigkeit: Wenn ein Softwareprojekt besonders hohe Anforderungen an die Qualität des Codes oder die Geschwindigkeit der Entwicklung stellt und die Zahl der Entwickler als Kostenfaktor eine untergeordnete Rolle spielt, ist Pair Programming zweifelsfrei überlegen.

Literatur

[1] Müller, Matthias M. ; Padberg, Frank ; Tichy, Walter F.: Ist XP etwas für mich? Empirische Studien zur Einschätzung von XP. In: *Tagungsband Software Engineering 2005* Gesellschaft für Informatik

[2] Nosek, John T.: The Case for Collaborative Programming. In: *Communications of the ACM* 41 (1998), Nr. 3, S. 105–108

[3] Williams, Laurie: *The Collaborative Software Process*, University of Utah, Diss., August 2000. http://collaboration.csc.ncsu.edu/laurie/Papers/dissertation.pdf. – Elektronische Ressource

[4] Kessler, Robert: *The Benefits of Pair Programming*. Version: 2002. https://faculty.university.microsoft.com/2002/DesktopModules/ViewDocument.aspx?DocumendID=73. – Online–Ressource, Abruf: 18.3.2005

[5] Williams, Laurie ; Kessler, Robert: All I Ever Needed to Know about Pair Programming I Learned in Kindergarten. In: *Communications of the ACM* 43 (2000), Nr. 5

[6] Stotts, David ; Williams, Laurie ; Nagappan, Nachiappan ; Baheti, Prashant ; Jen, Dennis ; Jackson, Anne: *Virtual Teaming: Experiments and Experiences with Distributed Pair Programming*. 2003

[7] Nawrocki, Jerzy ; Wojciechowski, Adam: Experimental Evaluation of Pair Programming. In: *European Software Control and Metrics (Escom)* (2001). http://www.agilealliance.com/articles/ExperimentalEvaluationOfPP.pdf

[8] Cockburn, Alistair ; Williams, Laurie: *The Costs and Benefits of Pair Programming*. 2001

[9] Sixtus, Mario: Gemeinsam auf die Spitze treiben. In: *Die Zeit* (1/2004)

[10] Sierra, Kathy: *Pair Programming is NOT always a choice*. Version: März 2004. http://weblogs.java.net/blog/kathysierra/archive/2004/03/pair_programmin.html. – Online–Ressource, Abruf: 9.3.2005

[11] Stephens, Matt ; Rosenberg, Doug: *Extreme Programming Refactored: The Case Against XP*. Apress, 2003. – ISBN 1–590–59096–1

[12] Baheti, Prashant ; Gehringer, Edward ; Stotts, David: *Exploring the efficacy of distributed pair programming*. 2002