Regulierung durch Technik

Arbeitsverteilung und Arbeitsorganisation in Projekten transnational verteilter Softwareentwicklung

Ingo Schulz-Schaeffer und Matthias Bottel

Beitrag zur Ad-Hoc-Gruppe »Problembearbeitung und Regulierung im Kontext grenzüberschreitender Arbeit«

Einleitung

Arbeit wird zunehmend mobil. Wir beobachten zum einen eine zunehmende Mobilität von Arbeitskräften, zum anderen aber auch eine zunehmende Mobilität von Arbeitstätigkeiten. Die Lohnbuchhaltung eines Unternehmens oder die in der Verwaltung einer Organisation anfallenden Dateneingabe, aber beispielsweise auch Verlagsarbeiten, wie das Lektorieren und Korrekturlesen von Texten, werden immer öfter nicht mehr vor Ort von Beschäftigten der betreffenden Organisation durchgeführt, sondern organisational ausgelagert („outsourcing“) und räumlich über Ländergrenzen hinweg verlagert („offshoring“). Wir interessieren uns in diesem Beitrag für die Mobilität von Arbeit in Gestalt der Mobilität von Arbeitstätigkeiten. Ein besonders hohes Mobilitätspotenzial besitzen alle informationsverarbeitenden Tätigkeiten, deren Inputs und Ergebnisse sich in Form digitaler Zeichenketten kodieren und übermitteln lassen. Denn sie lassen sich zumindest im Prinzip von jedem Ort aus durchführen, an dem die erforderliche Computer- und Telekommunikationsinfrastruktur vorhanden ist (vgl. Huws 2008: 16). Diese Tätigkeiten werden als Telearbeit (Baruch 2001), eWork (Huws 2008) oder auch als medienvermittelte Arbeit (Kleemann, Matuschek 2000) bezeichnet.

Die informationstechnische Infrastruktur ist eine notwendige Bedingung für Telearbeit, weil sie es ermöglicht, sowohl die erforderlichen Arbeitsmaterialien und Vorprodukte problemlos digital über räumliche Entfernungen zu übermitteln als auch die aus der Arbeitstätigkeit resultierenden Ergebnisse. Aber sie ist keine hinreichende Bedingung. Ebenfalls ist erforderlich, dass die betreffenden Arbeitstätigkeiten sich hinreichend entkoppelt von den Arbeits- und Produktionszusammenhängen durchführen lassen, zu denen sie beitragen. Sonst ließen sie sich nicht aus- und verlagern. Umgekehrt muss sichergestellt sein, dass die dergestalt entkoppelten Tätigkeiten nichtsdestotrotz ihre jeweilige Aufgabe als Teilschritte innerhalb jener umfassenderen Arbeits- und Produktionszusammenhänge erfüllen.

Standards sind hilfreich, um dieser doppelten Anforderung der gleichzeitigen Entkopplung und Integration mobiler Arbeitstätigkeiten aus ihren und in ihre jeweiligen Arbeitszusammenhänge Rechnung zu tragen. Dies kann (a) die Standardisierung ganzer Arbeitsabläufe sein, bei denen für bestimmte Arbeitsprozesse die einzelnen Arbeitsschritte und die Art und Weise ihres Ineinandergreifens vor-

gängig festgelegt sind; oder (b) die Standardisierung zumindest einzelner Arbeitsschritte; oder (c) die Standardisierung von Arbeitsmitteln, etwa in Gestalt von Standardsoftware, und (d) die damit häufig verbundene Angleichung von Arbeitspraktiken; (e) die Standardisierung von Schnittstellen zwischen Arbeitsschritten, beispielsweise durch Etablierung bestimmter Daten- und Dateiformate (ein Prozess, der ebenfalls durch Standardsoftware sehr vorangetrieben wird) sowie (f) die Entwicklung von Qualitätsstandards. Die Entwicklung von Standards wird auch von Ursula Huws als eine wesentliche Bedingung der Mobilität medienvermittelt durchführbarer Arbeitstätigkeiten genannt:

> „The spread of information and communications technologies, together with a global convergence in standards which has eased interoperability, combined with the near monopoly use of a relatively small number of software products and the growing dominance of English as the world business language, have created a situation where the standardisation of a very large range of business processes has become possible [...] The standardisation of many business processes combined with the digitisation of information and the development of high-capacity telecommunications networks has made it possible for telemediated work to be outsourced and/or relocated" (Huws 2006: 20).

Standardisierung ist ein wesentliches Mittel, um bestimmte Arbeitsschritte so aus umfassenderen Arbeitszusammenhängen herauslösen und getrennt von ihnen bearbeiten zu können, dass ihre Ergebnisse anschließend wieder in den umfassenderen Arbeitszusammenhang integrierbar sind. In dieser Weise eingesetzt, ist Standardisierung Bestandteil der klassischen Lösung für das Problem der Auskopplung von Arbeitsschritten oder Arbeitspaketen aus umfassenderen Arbeits- und Produktionszusammenhängen, nämlich der Strategie der vorgeplant-arbeitsteiligen Organisation von Arbeitsabläufen auf der Grundlage einer mehr oder weniger strikten Trennung von Planung und Ausführung. Hier sorgt im Erfolgsfall der vordefinierte Plan dafür, dass einzelne Arbeitspakete getrennt voneinander bearbeitet werden können und dennoch wie gewünscht ineinandergreifen.

Diese Form der Strukturierung von Arbeitszusammenhängen begrenzt den Umfang der aus- und verlagerbaren Telearbeit auf rein ausführende Arbeitstätigkeiten. Tatsächlich werden aber in einigem Umfang durchaus auch Arbeitsschritte medienvermittelt verlagert, die eigenständige Planungs- und Entscheidungskompetenzen erfordern. In diesem Zusammenhang wird häufig auf die Verlagerung von Arbeitstätigkeiten der Softwareentwicklung verwiesen. Hier hatte man es zwar anfangs überwiegend mit der Verlagerung ausführender Programmierung zu tun. Inzwischen ist zunehmend aber auch eine stärkere Verlagerung komplexerer nicht rein planausführender Tätigkeiten des Programm- und Systementwurfs etwa zu beobachten (vgl. Moczadlo 2002: 5; Huws et al. 2009: 58f.; Aspray et al. 2006: 55, 95).

Unsere Forschungen über transnational verteilte Projekte der Softwareentwicklung, die wir im Rahmen des DFG-Projekts „Techniken und Praktiken der Zusammenarbeit in transnationalen Projekten der Softwareentwicklung (TransSoft)" durchgeführt haben,[1] verweisen auf die zentrale Bedeutung von Verfahrensmodellen der Softwareentwicklung für die Mobilität der gut definierten, wie auch der weniger gut definierten Arbeitstätigkeiten in diesem Bereich.

[1] Unsere empirischen Befunde beruhen auf vier Fallstudien und knapp 20 Experteninterviews mit Wissenschaftlern und Praktikern aus dem Bereich der kollaborativen Softwareentwicklung. Die Fallstudien haben transnationale Entwicklungsprojekte zum Gegenstand, die jeweils von einem westeuropäischen Unternehmen mit einem Nearshore-Partner (nearshoring = Verlagerung ins nahegelegene Ausland, hier: Ost- und Südeuropa) durchgeführt wurden.

Softwareentwicklung als strukturierte Aufgabe

Strategien, die darauf zielen, Arbeitsaufgaben in Teilaufgaben zu zerlegen, deren Bearbeitung zu koordinieren und die Teilergebnisse in ein Gesamtergebnis zu integrieren, treten nicht erst mit dem Aufkommen der Option der räumlichen Verlagerung von Arbeitstätigkeiten auf den Plan. Sie sind eine Begleiterscheinung der funktionalen Differenzierung und arbeitsteiligen Organisation moderner Gesellschaften. Seit der industriellen Revolution werden entsprechende Strategien der Strukturierung von Arbeits- und Produktionsabläufen vor allem aus zwei Gründen gezielt entwickelt und eingesetzt: (a) um Effizienzgewinne arbeitsteiliger Spezialisierung realisieren zu können, und (b) um Aufgaben bewältigen zu können, die zu umfangreich und intern zu komplex sind, um als Ganzes überschaut und im Griff behalten werden zu können. Im Fall der Softwareentwicklung war vor allem der zweite Grund für die Entwicklung entsprechender Strategien strukturierter Programmierung verantwortlich. Mit ihnen reagiert die entstehende Profession der Softwareentwickler auf die so genannte Software-Krise der 1960er-Jahre. Die Bastler und Tüftler der ersten Stunde waren angesichts immer umfangreicherer Programme zunehmend weniger in der Lage, Software kosten-, zeit- und anforderungsgerecht zu produzieren. Dies führte zu der Forderung, Softwareentwicklung als Ingenieurtätigkeit – als Software-Engineering – neu aufzustellen. Die daraufhin entwickelten Prinzipien strukturierter Programmierung, die sich in der Anfangszeit vor allem in den so genannten Phasenmodellen der Softwareentwicklung niedergeschlagen haben, bilden den Grundstein dieser damaligen Neuorientierung der Softwareentwicklung (vgl. Bauer 1973; Bauer 1993; Schulz-Schaeffer 1996: 120f.).

Projekte der Softwareentwicklung sind heute in der Regel an einem Vorgehensmodell orientiert. Je größer ein Softwareprojekt ist und je mehr Entwickler/innen an ihm arbeiten, desto formaler und verbindlicher wird die Orientierung an Vorgehensmodellen typischerweise. Die Softwareentwicklungsmethode, die gegenwärtig besonders häufig in kleinen bis mittelgroßen Entwicklungsteams eingesetzt wird, heißt „Scrum“ und gehört in die Familie der so genannten agilen Verfahren. Sie bilden eine Alternative zu den Phasenmodellen der Softwareentwicklung, die lange Zeit vorgeherrscht hatten. Alle von uns untersuchten Softwareentwicklungsprojekte orientieren sich am Scrum-Vorgehensmodell. In allen unseren Fällen sind aber auch Aspekte des Vorgehens der Phasenmodelle relevant.

Die Phasenmodelle (auch Wasserfallmodelle genannt) beruhen darauf, dass Softwareentwicklung sich arbeitsteilig als ein Prozess organisieren lässt, der verschiedene Phasen durchläuft, in denen unterschiedlich spezialisierte Arbeitstätigkeiten im Vordergrund stehen (vgl. zum Beispiel Boehm 1981: 35ff.; Balzert 1992: 15ff., 468f.; Chroust 1992: 42ff.). Am Anfang steht demnach die Anforderungsermittlung, während derer die Systemspezifikation erarbeitet und festgelegt wird, also bestimmt wird, welche Funktions- und Leistungsmerkmale die zu entwickelnde Software besitzen soll. Es folgt die Ausarbeitung der Struktur des Programms, die als Software-Architektur bezeichnet wird. Die eigentliche Softwareentwicklung besteht in der Programmierung, also darin, den Software-Code zu schreiben. Schließlich müssen die programmierten Komponenten getestet und integriert werden. Phasenmodelle organisieren Softwareentwicklung als einen Prozess, in dem diese Arbeitsschritte in einer mehr oder weniger strikten Weise sequenziell abgearbeitet werden. Diese Form der arbeitsteiligen Produktion schlägt sich in korrespondierenden sozialen Positionen und Rollen nieder: der Position der Anforderungsanalytiker/-in, der Software-Architekt/-in, der Software-Entwickler/-in und der Tester/-in.

Die Strukturierungsleistung der Phasenmodelle wird mit einem hohen Maß an Festgelegtheit des Entwicklungsprozesses, langen Entwicklungszeiten bis zur Erstellung lauffähiger Software und geringer Flexibilität bei der Berücksichtigung veränderter Anforderungen erkauft. In allen diesen Hinsichten verschreibt sich die agile Herangehensweise den entgegengesetzten Zielen: der schnellen Erstellung

funktionierender Software; der Möglichkeit, Veränderungen zu jedem Zeitpunkt berücksichtigen zu können; der direkten Kooperation und Interaktion im Team statt formal festgelegter Arbeitsteilung (vgl. Beck et al. 2001).

Scrum ist das am weitesten verbreitete und am häufigsten eingesetzte agile Prozessmodell. Es ist zum einen darauf gerichtet, Zusammenarbeit im Team zu befördern und unterstützen; zum anderen unterstützt es ein Vorgehen, das darauf gerichtet ist, Entwicklungsprojekte gleichsam portionsweise zu verwirklichen und nicht in einem im Ganzen vorgeplanten Gesamtzusammenhang. Das heißt es wird in jedem Arbeitspaket jeweils nur ein abgrenzbarer Teil des späteren Produkts entwickelt; dieser Teil dann aber soweit, dass er funktionsfähig ist und getestet und evaluiert werden kann. Auf diese Weise soll erreicht werden, dass Fehler oder Holzwege schnell entdeckt und leicht und kostengünstig beseitigt bzw. korrigiert werden können.

Während Phasenmodelle den Arbeitszusammenhang der Software-Entwicklung auf der Grundlage arbeitsteilig bestimmter Positionen strukturieren, verkörpert Scrum eine Art Stakeholder-Modell. Ein Scrum-Team besteht aus einem Product-Owner, einem Entwicklungsteam und einem Scrum-Master. Der Product-Owner vertritt die Interessen des Produkts (und damit auch des Auftraggebers). Er/sie ist für die Qualität des Produkts verantwortlich. Das Entwicklungsteam ist für die Organisation und Durchführung der Entwicklungsarbeit verantwortlich. Es kann dabei arbeitsteilig vorgehen, ist aber als Team für die interne Strukturierung des Arbeitsprozesses und das Arbeitsergebnis verantwortlich. Der Scrum-Master schließlich ist dafür verantwortlich, dass die Regeln und Verfahren des Scrum-Prozesses eingehalten werden.

Wie ein an Scrum orientiertes Vorgehen im Idealfall aussieht, haben die Erfinder dieses Vorgehensmodells, Ken Schwaber und Jeff Sutherland in ihren Scrum-Leitfaden (vgl. Schwaber, Sutherland 2013) beschrieben: Im Zentrum des Scrum-Prozesses stehen die so genannten Sprints. Der Projektablauf ist als eine Abfolge aneinander anschließender Sprints strukturiert. Ein Sprint ist ein Arbeitspaket von maximal einem Monat Länge, „innerhalb dessen ein fertiges […], nutzbares und potenziell auslieferbares Produkt-Inkrement hergestellt wird." (Schwaber, Sutherland 2013) Ziel eines Sprints ist es, eine ausgewählte Anzahl von Funktionalitäten zu entwickeln, fertigzustellen und in den Entwicklungsstand des betreffenden Produkts zu integrieren, der am Ende des vorangegangenen Sprints erreicht worden war. Im Fall von Software heißt das: bestimmte Komponenten der zu entwickelnden Software so weit zu entwickeln, dass sie stabil lauffähig sind und die gewünschte Funktionalität besitzen, und diese neuen Komponenten in das Software-System, soweit es zuvor gediehen war, zu integrieren. Als Produkt-Inkrement wird der jeweils nach Abschluss eines Sprints erreichte neue Entwicklungsstand des betreffenden Produkts in Gestalt eines lauffähigen Programms bezeichnet. (Schwaber, Sutherland 2013)

Ein Sprint umfasst „das Sprint Planning, die Daily Scrums, die Entwicklungsarbeit, das Sprint Review und die Sprint Retrospektive". (Schwaber, Sutherland 2013) Das Sprint-Planning ist ein zeitlich begrenztes Treffen des gesamten Scrum-Teams und dient der Planung des jeweils anstehenden Sprints. Grundlage für die Planung eines Sprints ist das Product-Backlog. Das Product-Backlog ist eine Liste mit der Beschreibung aller Merkmale und Eigenschaften, die das Produkt besitzen soll. Abhängig von der Prioritätensetzung des Product Owners und den Aufwandseinschätzungen des Entwicklungsteams werden die im Sprint zu bearbeitenden Aufgaben ausgewählt und in einer zweiten Liste, dem sogenannten Sprint-Backlog, festgehalten. Die Entwicklungsarbeit orientiert sich am Sprint-Backlog. Dadurch ist der Entwicklungsfortschritt während eines Sprints zu jedem Zeitpunkt sichtbar.

Die Entwicklungsarbeit wird begleitet durch tägliche kurze Treffen, die Daily Scrums, an denen nur die Mitglieder des Entwicklungsteams aktiv teilnehmen, und die dem gegenseitigen Austausch über

Fortschritte und Probleme dienen. Am Ende des Sprints steht das Sprint-Review, ein Treffen des gesamten Scrum-Teams gemeinsam mit wichtigen Stakeholdern (zumeist: Vertretern des Auftraggebers). Es dient dazu, den erreichten Stand des Produkt-Inkrements vorzustellen, zu diskutieren, was gut und was schlecht gelaufen ist, wo das Projekt steht, welche Arbeiten sinnvollerweise im nächsten Sprint angegangen werden sollten und ob es Ergänzungen oder Änderungen des Projektziels gibt, die dann im Product-Backlog eingetragen werden. Die Sprint-Retrospektive ist ein Treffen des Scrum-Teams am Ende eines Sprints, in dem mögliche Verbesserungen der Arbeitsweise des Scrum-Teams identifiziert werden sollen.

Die Entwicklungsarbeit wird begleitet durch tägliche kurze Treffen der Mitglieder des Entwicklungsteams, den Daily Scrums, die dem Austausch über die aktuellen Fortschritte und Probleme dienen. Am Ende des Sprints steht das Sprint-Review, ein Treffen des gesamten Scrum-Teams gemeinsam mit wichtigen Stakeholdern (zumeist: Vertreter/innen des Auftraggebers). Es dient dazu, den erreichten Stand des Produkt-Inkrements vorzustellen und zu diskutieren und daraus abgeleitet den nächsten Sprint vorzuplanen. Die Sprint-Retrospektive ist ein Treffen des Scrum-Teams am Ende eines Sprints, in dem mögliche Verbesserungen der Arbeitsweise des Scrum-Teams identifiziert werden sollen.

Zusammengenommen soll dieses Verfahren der Projektorganisation und Aufgabenstrukturierung dafür sorgen, dass Entwicklungsprojekte transparent und flexibel arbeiten. Dadurch, dass jeder Sprint ein in sich abgeschlossenes Projekt im Gesamtprojekt bildet, soll einerseits Komplexität reduziert werden, andererseits sichergestellt werden, dass die Kosten für Fehlentwicklungen begrenzt bleiben, nämlich auf die Kosten eines Sprints. Teure Software-Ruinen, wie sie an Phasenmodellen orientierte Entwicklungsprojekte immer wieder zeitigen, sollen auch dadurch vermieden werden, dass am Ende jedes Sprints eine fertige und nutzbare Produktversion steht. Außerdem soll die Sprint-Struktur es ermöglichen, sich wandelnde Anforderungen an das zu entwickelnde Produkt zu jedem Entwicklungszeitpunkt erkennen und berücksichtigen zu können.

Erkauft wird dieses Maß an Flexibilität, Transparenz und Schnelligkeit allerdings mit einem Verzicht auf die Vorteile einer längerfristigen Vorausplanung (etwa für die Ressourcenplanung oder für die Möglichkeit einer System-Architektur aus einem Guss) und arbeitsteiliger Spezialisierung. Auch in Entwicklungsprojekten, die sich an Scrum orientieren, sind Anteile des phasenorientierten Vorgehens deshalb zumeist ebenfalls zu finden. Die Auffassung, dass in der Praxis zwangsläufig Elemente aus beiden Vorgehensmodellen zum Tragen kommen, bringt einer unserer Gesprächspartner wie folgt auf den Punkt:

> „Vielleicht noch so, dass man das Agile und, ich sag mal so, die alte Schule der Softwareentwicklung, dass man da weder schwarz noch weiß machen kann. Es gibt das, was alle immer sagen, früher haben wir Wasserfall gemacht, wenn man heute mit Leuten spricht, also [...] wir machen heute Agil, früher haben wir Wasserfall gemacht. [...]. Also ich glaube, weder das eine noch das andere stimmt. Ich glaube, die meisten haben nie Wasserfall gemacht, weil, dass da zwei Jahre keine Änderungen gaben, glaube ich niemandem. Dass am Anfang wirklich allen alles klar war, glaube ich auch niemandem. Umgekehrt ist so dieser rein agile Ansatz, so, wir bauen jetzt einfach nur ein Feature und noch ein Feature und noch ein Feature, das funktioniert halt auch nicht. Man muss sich halt schon so einmal zurücklehnen, so die Gesamtheit der Anforderungen oder zumindest die, die man kennt, angucken, und dann überlegen, was dafür auch eine valide Architektur ist, statt so Feature-getrieben monatsweise festzustellen, ach, so was wollt ihr auch, dann müssen wir da noch mal umbauen. Das ist halt auch nicht zielführend, weder das eine noch das andere, der Anspruch, alles vorher zu wissen, das funktioniert nicht, das weiß einfach niemand, und der andere Ansatz, ich muss eigentlich nur wissen, was ihr in den nächsten vier Wochen wollt, ist auch nicht hilfreich,

> weil eigentlich das Wissen schon im Haus ist, was auch noch kommen muss, und dann sollte man sich damit auch beschäftigen und die Entscheidungen zumindest in dem Licht dann auch treffen“ (FallB-BerMa-D-P2: 7).

Organisationstechnische und sachtechnische Strukturierungen von Arbeit

Die Vorgehensmodelle der Softwareentwicklung sind im Kern Organisationstechniken. Es sind Modelle, die aus einem Zusammenhang von Regeln für die Strukturierung des Arbeitsprozesses der Herstellung von Softwareprogrammen bestehen. Ihr Ziel ist es, bessere Softwareprodukte einfacher herstellen zu können als es ohne diese Verfahren möglich wäre. Im Sinne der Definition von Technik als „künstlich erzeugte und in der einen oder anderen Weise festgelegte Wirkungszusammenhänge, die genutzt werden können, um hinreichend zuverlässig und wiederholbar bestimmte erwünschte Effekte hervorzubringen“ (Schulz-Schaeffer 2008: 445), handelt es sich bei ihnen dementsprechend um Techniken. Die Vorgehensmodelle der Softwareentwicklung sind im Kern Organisationstechniken, weil der technische Effekt wesentlich durch Regeln für das handelnde Zusammenwirken der Beteiligten erzeugt wird (vgl. Schulz-Schaeffer 2008: 447).

Die organisationstechnische Strukturierung des Prozesses der Softwareentwicklung durch die Vorgehensmodelle ist eng verbunden mit sachtechnischen Werkzeugen der Softwareentwicklung, das heißt mit den sogenannten Software-Werkzeugen, die Softwareentwickler als Hilfsmittel im Arbeitsprozess benutzen. Beispiele für solche Software-Werkzeuge sind Compiler, die den in der Programmiersprache verfassten Text in ein ausführbares Programm übersetzen, Codeanalyse-Programme, die die Qualität des Codes überprüfen, oder Versionsverwaltungsprogramme, die das Dateimanagement bei der Programmierarbeit unterstützen. Die häufig enge Verbindung von Organisationstechniken und sachtechnischen Werkzeugen ist dadurch bedingt, dass die Strukturierung des Arbeitsprozesses sich auf die Strukturierung der Arbeitsaufgabe auswirkt (und umgekehrt). Dementsprechend müssen die softwaretechnischen Werkzeuge, die der Unterstützung der Durchführung der Arbeitsaufgabe dienen, Anforderungen gerecht werden, die sich aus der Strukturierung des Arbeitsprozesses ergeben.

Ein Software-Werkzeug namens „Continuous Integration Server“ eignet sich gut, um diesen Zusammenhang zu veranschaulichen: Dieses Werkzeug dient dazu, kontinuierlich neue Stücke Software in den erreichten Stand zu integrieren, daraus ein lauffähiges Programm zu generieren und es auf Integrationsprobleme hin zu testen. Kontinuierliche Integration neuer Programmteile ist eine durch die agilen Vorgehensmodelle hervorgerufene Anforderung. Denn hier ist das Ziel ja, jeden Sprint mit einer lauffähigen Version des Programms zu beenden, in die der erreichte Entwicklungsfortschritt enthalten ist.

Illustrierende Beispiele

Für das Verständnis der Strukturierungen und Regulierungen des Arbeitsprozesses, die es ermöglichen, Softwareentwicklung in transnational verteilten Entwicklungsteams durchzuführen, scheinen uns – als Ergebnis der Überlegungen der vorangegangenen beiden Abschnitte – zwei Formen der Verbindung von Verfahren zentral zu sein: die Verbindung von Aspekten der Phasenmodelle mit Aspekten der agilen Vorgehensweise, und die Verbindung von organisationstechnischer und sachtechnischer

Strukturierung. Dies wollen wir im letzten Abschnitt dieses Beitrages exemplarisch am Beispiel einiger konkreter Verfahren und Techniken der Softwareentwicklung und der Praktiken ihrer Nutzung in den von uns beforschten Entwicklungsprojekten veranschaulichen: den User-Stories, dem Sprint-Backlog und dem Product-Backlog, den Scrum-Boards und den Burn-Down-Charts.

User-Stories sind eine Form der Spezifikation von Systemanforderungen an das zu entwickelnde Softwareprogramm. Es ist eine Verfahrenstechnik, die es erlaubt, die Anforderungen an das System so zu erfassen und zu spezifizieren, dass sie dann als Funktionalitäten in separaten Sprints realisiert werden können. Eine User-Story ist eine technische Funktionalität so wie sie sich aus der Perspektive der Nutzung des zu entwickelnden Softwareprogramms darstellt. Beispielsweise: „Ich möchte das Ergebnis einer Suche bei Bedarf anschließend ausdrucken können." Die Formulierung von Anforderungen als User-Stories ist eine Konkretisierung der Richtlinie des Scrum-Leitfadens, dass die für einen Sprint „ausgewählten Product-Backlog-Einträge […] eine zusammenhängende Funktionalität [bilden]" (Schwaber, Sutherland 2013). Diese Anforderung ergibt sich direkt aus der Vorgehensweise, mit jedem Sprint weitere lauffähige Funktionalitäten zu dem entstehenden Produkt hinzuzufügen. Denn, um dies zu erreichen, müssen die in einem Sprint entwickelten Stücke Software entsprechende funktionale Einheiten bilden. User-Stories sind genau dies: Bündel von Anforderungen, die – wie im eben angeführten Beispiel – auf die Realisierung einer zusammenhängenden Funktionalität zielen. Liegen alle Anforderungen an das Produkt im Product-Backlog in Form von User-Stories vor, dann kann für den jeweiligen Sprint eine entsprechende Anzahl von User-Stories ausgewählt und als Arbeitspaket für den betreffenden Sprint in das Sprint-Backlog verschoben werden, in Teilaufgaben klein gearbeitet und realisiert werden. So wird sichergestellt, dass jedes Produkt-Inkrement für sich genommen nutzbare Funktionalitäten aufweist.

Scrum-Boards sind die softwaretechnische Realisierung des Product-Backlogs und des Sprint-Backlogs in Form von Datenbanken, in die die Aufgaben für das ganze Projekt (Product-Backlog) bzw. für den jeweiligen Sprint (Sprint-Backlog) eingetragen werden können. Im Scrum-Board kann die Reihenfolge der Aufgaben vermerkt werden, die Aufteilung von Aufgaben in Unteraufgaben, der geschätzte Zeitaufwand für die einzelnen Aufgaben sowie Abhängigkeiten zwischen Aufgaben. Außerdem ist jede Aufgabe mit einem Status versehen, der ihren Bearbeitungszustand vermerkt (zum Beispiel: „to do", „in progress", „done"). Scrum-Boards werden als Bestandteil unterschiedlicher Software-Pakete angeboten, etwa als Bestandteil der Projektmanagement-Software „JIRA" oder der Plattform für kollaborative Softwareprojekte „Team Foundation Server".

Ein wichtiges Merkmal des Scrum-Boards sind die von diesem Werkzeug bereitgestellten Visualisierungen des Projektfortganges, die einen schnellen Überblick darüber erlauben, welche Aufgaben weniger oder mehr Aufwand erfordern als erwartet und ob ein Sprints sich insgesamt im Plan befindet. Eine besonders markante Visualisierung ist der Burn-Down-Chart, eine Grafik, die über den Zeitverlauf eines Sprints hinweg die Abnahme des Arbeitsaufwandes dokumentiert, der zur Erfüllung des Sprint-Ziels erforderlich ist, und der im Idealfall linear abnehmen und am Ende des Sprints bei null angekommen sein sollte. Den engen Zusammenhang zwischen der regelhaften Strukturierung und der sachtechnischen Verfestigung von Verfahrenselementen der Softwareentwicklung zeigt das folgende Problem mit der Generierung von Burn-Down-Charts, das uns aus einem der von uns untersuchten Entwicklungsprojekte berichtet wurde.

Scrum-Boards unterstützen die Vorgehensweise, zusammenhängende Funktionalitäten zu entwickeln. Das Scrum-Board von JIRA, das im betreffenden Fall benutzt wurde, bezeichnet die Ebene der User-Stories als „issues" und die Teilaufgaben, aus denen sie bestehen, als „sub-tasks". Die Anzahl der für eine User-Story veranschlagten und bereits eingesetzten Arbeitsstunden ermittelt JIRA durch Auf-

summieren der entsprechenden Angaben für die Teilaufgaben, aus denen diese User-Story besteht. Für unterschiedliche technische Unterstützungen wird dann nur noch dieser aufsummierte Wert verwendet. So auch bei der Erstellung der Burn-Down-Charts.

In dem betrachteten Fall haben die Entwickler Sprints definiert, die unvollständige User-Stories enthielten, bei denen also bestimmte Teilaufgaben ausgelassen und für spätere Sprints aufgehoben wurden. Weil nun aber das Scrum-Board die Arbeitsstatistik stets auf der Aggregationsebene der Issues berechnet, tauchen dort die Arbeitsstände aller Teilaufgaben der im Sprint einbezogenen Issues auf, also auch die Aufwandsschätzungen der Teilaufgaben, die nicht im Sprint aufgenommen worden waren. Folglich funktionierte der Burn-Down-Chart nicht. Denn Aufgaben, die man nicht bearbeitet, verringern sich logischerweise nicht. Einer unserer Gesprächspartner beschreibt das Problem so:

> „Und warum wir keine Burn-Down-Charts haben, das liegt an dem Tool, weil irgendwie ist es so: Wenn ich diese User-Story nicht schließe, also, wenn ich Sub-Tasks schließe, kommt der Burn-Down-Chart nicht vorwärts, also nicht nach unten. Erst wenn ich die User-Story komplett schließe, kommt es nach unten. Aber bei uns ist es ja anders. Also wir arbeiten nicht gegen User-Stories, wir arbeiten gegen Arbeitspakete [d.h. Teilaufgaben, Anm. d. Verf.]. Und auch wenn wir hier zum Beispiel fünf Arbeitspakete abschließen, kann es sein, dass die User-Story immer noch nicht geschlossen ist, dass wir vielleicht in Sprint 5 noch mal dran arbeiten, aber an einer anderen Ecke. Und deswegen funktioniert das mit den Burn-Down-Charts nicht“ (FallD-GesFü-TR-P1: 65).

Die Störung des Zusammenwirkens von organisationstechnischen und sachtechnischen Verfahren der Strukturierung des Arbeits- und Produktionsprozesses kollaborativer Softwareentwicklung zeigt umso deutlicher, wie sehr in die softwaretechnischen Werkzeuge Annahmen über die organisationstechnische Strukturierung des Verfahrens eingeschrieben sind – in diesem Falle also die Erwartung, dass Softwareentwickler sich an der Verfahrensregel orientieren, stets nur vollständige User-Stories (zusammenhängende Funktionalitäten) in ein Sprint-Backlog einzustellen. In vergleichbarer Weise leiten sich umgekehrt aus der organisationstechnischen Strukturierung Erwartungen an die sachtechnische Unterstützung ab. Der zuvor erwähnte Continuous Integration Server ist dafür ein Beispiel.

Auf den ersten Blick mag es so scheinen, dass das Vorgehen der agilen Verfahren und das der Phasenmodelle einander ausschließen. Auf den zweiten Blick wird erkennbar, dass die beiden Vorgehensweisen unterschiedliche Probleme der Strukturierung von Softwareentwicklung adressieren. Und der Blick in die empirische Wirklichkeit zeigt, dass erfolgreiche Projekte der Softwareentwicklung stets beide Problemkomplexe bearbeiten, also vorgeplante Arbeitsteilung und flexible Verständigung im Team miteinander kombinieren. Es dürfte ein Verdienst des Scrum-Vorgehensmodells sein, die Kombinierbarkeit dieser beiden Vorgehensweisen weit über das hinaus ermöglicht zu haben, was dessen Erfindern vorschwebte. Es versteht sich von selbst, dass diese Kombination nicht spannungsfrei erfolgt. Entscheidend aber ist, dass sie grundsätzlich möglich ist und tatsächlich genutzt wird.

Im Scrum-Vorgehensmodell hat das Sprint-Backlog – und dementsprechend dessen softwaretechnische Realisierung im Scrum-Board – eine verständigungsorientierte Funktion. „Das Sprint Backlog ist ein hochgradig sichtbares Echtzeit-Bild der Arbeit, die das Entwicklungsteam plant, während des Sprints zu erreichen.“ (Schwaber, Sutherland 2013) Es soll „die Transparenz der wesentlichen Informationen maximieren, um für alle ein gleiches Verständnis über das Artefakt zu schaffen“ (Schwaber, Sutherland 2013). Der so herstellbaren Transparenz wegen, lässt sich das Scrum-Board zugleich aber bestens nutzen, um zu kontrollieren, ob sich die Sprintdurchführung im Plan befindet. So lässt sich dieses Werkzeug für beide Abstimmungsmodi, Abstimmung im Team und Abstimmung von Plan und Durchführung, einsetzen. In einem unserer Fälle (Fall C) wird die Nutzung des Scrum-Boards sogar soweit in Richtung der Abstimmung von Planung und Durchführung verschoben, dass die kollabor-

ative Funktion dieses Werkzeugs zumindest für das ungarische Mitglied des verteilten Entwicklungsteams nicht mehr erkennbar ist:

> „In JIRA we mainly tracked our workload, so on which project, how many… how much effort was put on a task or some project. […] It was easy to use and it helped me a lot to see, what kind of tasks I have. But we did not use it on a very collaborative mode, on a very collaborative way. It was just helpful for me to see what kind of tasks I have." (FallC-Entw-U-P2: 95)

Zusammengefasst sprechend die hier nur beispielhaft illustrierten Befunde unserer Untersuchung dafür, dass vor allem zwei Formen der Strukturierung die räumliche und transnationale Mobilität der Arbeitstätigkeiten begünstigen: (1) die Kombination von Verfahrenselementen eines vorgeplant-arbeitsteiligen Vorgehens mit Elementen eines teamförmig-aushandlungsorientierten Vorgehens und (2) die regelhafte Verfestigung von Verfahrenselementen beider Vorgehensweisen. Durch Einbettung des vorgeplant-arbeitsteiligen Vorgehens in iterative Aushandlungszyklen ermöglicht die Verfahrenskombination es, deren Vorteile für verteiltes Arbeiten zu nutzen, zugleich aber ihre Risiken kontrollierbar zu machen. Die aufeinander bezogenen organisationstechnischen und sachtechnischen Verfahren machen das zur Durchführung der Arbeitsaufgaben erforderliche Wissen mobiler, weil technische Verregelung eine Form der Standardisierung ist.

Literatur

Aspray, W., Mayadas, F., Vardi, M. Y. 2006: Globalization and Offshoring of Software. A Report of the ACM Job Migration Task Force.

Balzert, H. 1992: Die Entwicklung von Software-Systemen. Prinzipien, Sprachen, Werkzeuge, Mannheim u.a.: BI-Wissenschaftsverlag.

Baruch, Y. 2001: The status of research on teleworking and an agenda for future research. International Journal of Management Reviews, 3. Jg. Heft 2, 113–129.

Bauer, F. L. 1973: Software engineering. In F. L. Bauer (Hg.), Software engineering. An advanced course. New York u.a.: Springer, 522–545.

Bauer, F. L. 1993: Software Engineering - wie es begann. Informatik-Spektrum, 16. Jg., 259–260.

Beck, K. et al. 2001: Agile Manifesto, http://www.agilemanifesto.org (letzter Aufruf 5. Januar 2017).

Boehm, B. W. 1981: Software engineering economics. Englewood Cliffs, New Jersey: Prentice-Hall.

Chroust, G. 1992: Modelle der Software-Entwicklung, München u.a.: Oldenbourg.

Huws, U. 2006: The transformation of work in a global knowledge economy: Towards a conceptual framework. Leuven: Katholieke Universiteit Leuven.

Huws, U. (Hg.) 2008: The globalisation glossary. A researcher's guide to understanding work organisation restructuring in a knowledge-based society. Leuven: Katholieke Universiteit Leuven.

Huws, U. et al. 2009: Value chain restructuring in Europe in a global economy. Leuven: Katholieke Universiteit Leuven.

Kleemann, F., Matuschek, I. 2000: Arbeits- und Technikstile in medienvermittelter Arbeit. In Tagungsreader zur Tagung „Neue Medien im Arbeitsalltag", TU Chemnitz, 169–192.

Moczadlo, R. 2002: Chancen und Risiken des Offshore-Development. Empirische Analyse der Erfahrungen deutscher Unternehmen, http://www.competence-site.de/chancen-und-risiken-des-offshore-development/ (letzter Aufruf 5. Januar 2017).

Schulz-Schaeffer, I. 1996: Software-Entwicklung zwischen Ingenieur- und Designwissenschaft. Überzeugungskraft und nützliche Widersprüchlichkeit von Software-Engineering und Software-Gestaltung. In H. D. Hellige (Hg.), Technikleitbilder auf dem Prüfstand. Leitbild-Assessment aus Sicht der Informatik- und Computergeschichte. Berlin: Edition Sigma, 115–140.

Schulz-Schaeffer, I. 2008: Technik. In N. Baur et al. (Hg.), Handbuch Soziologie. Wiesbaden: VS Verlag, 445–463.

Schwaber, K., Sutherland, J. 2013: Der Scrum Guide. Der gültige Leitfaden für Scrum: Die Spielregeln, http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-DE.pdf (letzter Aufruf 5. Januar 2017).