Software-Ingenieure als kompetente Teamworker

Claus Lewerentz und Heinrich Rust *
Lehrstuhl Software-Systemtechnik, BTU Cottbus

Für Software-Ingenieure der Zukunft sind professionelle persönliche Arbeitstechniken und Arbeitsweisen der Softwareentwicklung im Team ebenso integrale Bestandteile ihres professionellen Repertoires wie technische Kenntnisse.

Für diese These werden in den folgenden Abschnitten Beobachtungen und Argumente angeführt, die erkennen lassen, daß eine Integration sozial- und arbeitspsychologischer Konzepte in die Software-Technik für ihren zukünftigen Erfolg wesentlich ist.
Im letzten Abschnitt skizzieren wir Konsequenzen dieser Sicht für die Softwaretechnik-Ausbildung.

1. Software-Entwicklung findet immer im Team statt.

Das Potenzial des sozialen Charakters der typischen Softwareentwicklungssituation wird zwar erkannt und von Praktikern anekdotisch beschrieben, aber nicht systematisch analysiert. Ohne eine Systematisierung der bisherigen Erkenntnisse ist auch eine Systematisierung der Suche nach neuen Prozesselementen und Prozesselementkombinationen, die die soziale Entwicklungsbedingung nutzen, unmöglich. Für diese Systematisierung sind sozialpsychologische Theorien nötig.

Zentrale Begriffe der beginnenden Softwaretechnik waren „strukturierte Programmierung" und "top-down software development" [18, 19]. Hier wurden technische Verfahren entwickelt, die die konzeptionelle Überforderung individuell arbeitender Softwareentwickler reduzieren sollten.

Im Nachhinein lässt sich der Erfolg dieser Verfahren kognitionspsychologisch begründen. Dijkstra [9] nutzt ein implizites Argument dieser Art, wenn er für die Nutzung mathematischer Herangehensweisen in der Softwareentwicklung wirbt: Die Mathematik sei die historisch erfolgreichste Vorgehensweise, mit allzu komplexen Problemen umzugehen. Naur [20] betont die Gefahr der Orientierung an einem allzu sehr am Formalismus orientierten Verständnis der Mathematik. Aber auch er bleibt beim Ansatz individueller Softwareentwicklung.

Softwareentwicklung ist aber gewöhnlich ein soziales Geschehen. Im Rahmen der heutigen Softwaretechnik wird dies etwa unter dem Generalthema „Softwareprozesse" untersucht. In explizit gemachten Softwareentwicklungsprozessen wird nicht nur die Reihenfolge von Arbeitsgängen festgelegt, sondern auch die Art und Weise der Zusammenarbeit der Projektbeteiligten miteinander. Solche wohldefinierten Softwareprozesse werden vorgeschlagen, um die Leistungsfäahigkeit von Softwareentwicklungsorganisationen systematisch zu verbessern [22, 23]. Wenn die mit diesen Prozessen verbundenen Versprechungen auch nur annähernd realistisch sind, warum werden sie dann nicht intensiver eingesetzt?

* BTU Cottbus, Postfach 101344, D-03013 Cottbus, Deutschland;
Tel. +49(355)69-3881, Fax.:-3810;(cl, rust)@informatik.tu-cottbus.de

Dass sozialpsychologische Faktoren für den Projekterfolg kritisch sein können, wurde schon früh erkannt: Weinberg [27] etwa sammelte anekdotische Hinweise auf psychologische Erfolgsfaktoren für Softwareentwicklungsprojekte, unter denen sozialpsychologische eine wichtige Rolle spielen. Auch Brooks [3] referiert derartige Erfahrungen, wie sie bei der Entwicklung eines umfangreichen Programmsystems gesammelt wurden. Äußerst populär sind die Analysen von DeMarco und Lister [8]. Neuere Empfehlungen für die soziale Entwicklungspraxis finden sich in der Form von Prozessmustern [4], dem Ansatz des Extreme Programming [1] oder des Team Software Process [16]. Diese wirken allerdings als bloße ad-hoc-Kombinationen empirisch erfolgreicher Prozesselemente und werden nicht aus einer umfassenderen Systematik heraus gerechtfertigt. Ein Beispiel: “Pair programming”, ein Element des Extreme Programming, ist ein anscheinend erfolgreiches Konzept, aber theoretisch ist dies nicht soweit analysiert, dass die Erfolgsbedingungen für diese Entwicklungstechnik klarer abgrenzbar würden.

Es fehlen Theorien, die die vielversprechenden Partialerfahrungen integrieren und eine systematischere Suche nach praktisch erfolgversprechenden Prozesselementen und ihren Kombinationen erlauben. Ansätze aus soziologischer Sicht finden sich etwa in [29], solche aus psychologischer Perspektive in [2]. Solche Ansätze müssen in die Softwaretechnik aufgenommen werden.

2. Software-Entwicklungsprozesse werden als gemeinsame Lernprozesse verstanden

Softwareentwicklung findet gewöhnlich in Teams und in rasch veränderlichen Umgebungen statt. Entwicklungsteams müssen sich also auf wechselnde Anforderungen einstellen, oder mit anderen Worten: Softwareentwicklung sollte als gemeinsamer Lernprozess aller Beteiligten organisiert werden.

Dass Softwareentwicklungsprozesse als Lernprozesse organisiert werden sollten, spiegelt sich schon in der ersten Beschreibung des klassischen Wasserfall-Prozesses [24]: Zyklen beim Durchlauf der Prozessschritte sind unvermeidbar und sollten daher von vornherein eingeplant werden. Die Iteration von Arbeitsphasen ist ein wichtiges Element von Lernprozessen im allgemeinen. Um zu klären, welche anderen Merkmale möglicher Arbeitsweisen das Lernen fördern, und wie insbesondere die soziale Situation für den Lernprozess genutzt werden kann, ist aber ein differenziertes begriffliches Instrumentarium nötig. Pasch etwa betont die Bedeutung von Gesprächen und des wechselseitigen Eingehens der Gruppenmitglieder aufeinander bei der Suche nach angemessenen Entwürfen für eine Problemstellung [21]; er nutzt dafür soziologische Begriffe.

Floyd polemisiert seit den achtziger Jahren gegen ein Bild der Softwareentwicklung, das sich insofern allzu eng an der Algorithmusmetapher orientiert, als in einer ersten Phase mit der Anforderungsspezifikation ein “input” verlangt wird, aus dem die gewünschte Applikation als “output” bestimmt wird, ohne dass unterdessen mit Auftraggebern oder anderen Betroffenen Zwischenergebnisse geprüft und validiert werden müssten [12, 13, 11, 14]. Floyd weist auf die Praxisferne eines solchen Ansatzes hin. Im typischen Fall lassen sich die Anforderungen im Vorhinein nicht vollständig erfassen, schon weil diese sich ändern, weil verschiedene Betroffene widersprüchliche Anforderungen haben, und weil sich die Betroffenen die künftige Situation nicht im hinreichenden Detail vorstellen können. Floyd benutzt den Begriff der Software-Evolution. Die Betonung des Evolutionsbegriffs stellt die Erfahrung in den Mittelpunkt, dass Projekte ein Eigenleben entwickeln, so dass eine quasi-algorithmische Vorplanung des Projekt-Ablaufes unmöglich wird. Floyd thematisiert zudem die Akzeptanz der informationstechnischen Lösungen; diese soll durch Partizipation der Betroffenen bei der Systementwicklung erreicht werden. Es existieren also verschiedene Ansätze, Softwareentwicklung als Lernprozesse zu organisieren.

Eine organisationspsychologische Analyse von Arbeitsgruppen ist nötig, um besser die Bedingungen für erfolgreiches Lernen im Entwicklungsteam zu beleuchten.

3. Techniken und Verfahren des Projektmanagements gewinnen an Akzeptanz

Anscheinend vielversprechende Techniken und Verfahren, die für das Projektmanagement und die Qualitätssicherung in der Softwaretechnik entwickelt wurden, stoßen in der Praxis auf teils massive Akzeptanzprobleme. Beispiele dafür sind etwa strikt definierte Softwareprozesse und auf Reflexion beruhende Prozesselemente wie Prozess- oder Produktmetriken. Für die Analyse solcher Akzeptanzprobleme sind arbeitspsychologische Untersuchungen nötig

Da Softwareentwicklung gewöhnlich in sozialen Situationen stattfindet, liegt es nahe, deren spezifische Vorteile zu nutzen. Solche Versuche gibt es schon lange. Gewisse Leistungen einer spezifischen sozialen Situation hat Mills versucht, bei der Konzeption der Chief Programmer Teams zu nutzen [17], und Fagan bei der Gestaltung von Inspektionen für die Qualitätssicherung [10]. Diese Ansätze zur Nutzung der sozialen Entwicklungssituation sind teils fast völlig erfolglos geblieben (im Falle der Chief programmer teams), teils lange Zeit auf erhebliche Widerstände von Entwicklerseite gestoßen, so dass das Potenzial der Methode nicht ausgeschöpft wird (bei Fagans Inspektionen).

Auch auf die Situation des einzelnen Entwicklers abzielende anscheinend vielversprechende Techniken treffen auf Akzeptanzprobleme. Ein Beispiel aus neuerer Zeit sind Schwierigkeiten mit dem PSP [15]: Es gibt erhebliche Probleme, Entwicklern diesen ausgefeilten individuellen Entwicklungsprozess schmackhaft zu machen. Softwaretechnikern kann man, zumindest beim aktuellen und kurzfristig erwartbaren Arbeitsmarkt, ihre Arbeitsweise nur in kleinem Maße vorschreiben. Prozess-Schemata, die die Akzeptanz nicht in Anschlag bringen, werden also unter diesen Bedingungen nicht erfolgreich sein.

Es ist zu erwarten, dass Softwareprozesse noch wichtiger werden, aber nicht so sehr ihre bloß normative Definition, sondern ihre empirische Praktikabilität. In diesem Zusammenhang wird es zu einer wichtigen Frage, wie die Akzeptanz technischer und organisatorischer Mittel in der Entwicklungspraxis erhöht werden kann. Dies wird eine zunehmende Empirisierung der Softwaretechnik-Forschung bedeuten, da die bloß normative Beschreibung einer Vorgehensweise als der angemessenen nicht begründet. Damit verbunden ist die Chance einer zunehmenden Praxisbezogenheit. Was für Merkmale von Softwareprozessen erleichtern oder erschweren die Akzeptanz bei heutigen Entwicklern? Das kann allenfalls eine arbeitspsychologische Analyse erhellen.

Auch bei eher technischen Fragen wird es in zunehmendem Maße darum gehen, wie sie arbeitsorganisatorisch auf akzeptable Weise in vorhandene Arbeitsweisen eingebunden werden können. Am Lehrstuhl für Software-Systemtechnik der BTU Cottbus wurde ein Werkzeug zur Vermessung objektorientierter Softwaresysteme auf Entwurfsebene entwickelt [25]. Bei der praktischen Anwendung trat immer wieder dasselbe Problem auf: Die Projektleiter und Manager waren sehr angetan von der Aussicht auf ein einfach anzuwendendes Werkzeug zur Qualitätsmessung auf Designebene, während die Entwickler bei der Diskussion umgehend eine defensive Haltung einnahmen und das Werkzeug ablehnten. Wie ist die Akzeptanz eines solchen Werkzeuges zu erhöhen?

4. Das professionelle Selbstbild der Software-Ingenieure verändert sich

Die Softwaretechnik ist, als Teildisziplin der Informatik, wie diese von mathematischen und elektrotechnischen Anfängen geprägt. Struktur- und ingenieurwissenschaftliche Begriffe und Methoden werden zur Beschreibung und Lösung von Problemen genutzt. Damit dies funktioniert, werden auftretende Probleme als technische oder formale aufgefasst.

Die mathematisch-technischen Anfänge prägen die Softwaretechnik bis heute. Die Beschäftigung mit Formalismen, normierten Notationen und computergestützten Werkzeugen zur Verwendung bei der Softwareentwicklung passen in den von den Ursprungsdisziplinen aufgespannten Rahmen. Begriffe der Geistes- und Sozialwissenschaften gehören nicht zum akzeptierten Werkzeuginventar der Softwaretechnik. Diese methodische Eingrenzung erschwert es, softwaretechnische Probleme überhaupt als sozial- und arbeitspsychologische wahrzunehmen und zu beschreiben, was zur Konsequenz hat, dass eine Lösung dieser Probleme mit den adäquaten Mitteln gewöhnlich unterbleibt. So konnte Weinberg sein Buch von 1971 auch darum 1998, nur um einige Kommentare ergänzt, wieder veröffentlichen [28], weil seine Beobachtungen zum allergrößten Teil nach wie vor gültig waren, und weil eine systematisierende Vertiefung im verstrichenen Vierteljahrhundert nicht stattgefunden hatte. Die Hoffnung, mit seinem Buch eine neue Disziplin, nämlich die „Psychologie der Softwareentwicklung" anzuregen, hatte sich nicht erfüllt.

Die Arbeitsumgebungen von immer mehr Menschen sind in immer größerem Maße durch die Produkte der Softwaretechnik geprägt. Damit die Softwareentwickler dafür sorgen können, dass Handlungsspielräume an den Arbeitsplätzen erhalten oder möglichst sogar ausgeweitet werden, müssen die Entwickler für die entsprechenden arbeitswissenschaftlichen Probleme sensibilisiert werden.

Volpert [26] beschreibt Gestaltung vollständiger Arbeitsumgebungen als Ziel: Software-Ergonomie darf nicht auf „Oberflächenphänomene" beschränkt bleiben. Auch Coy betont, dass die Informatik Arbeitsprozesse reorganisiert [5]. Dahme und Raeithel [7] beschreiben, wie ein tätigkeitstheoretischer Ansatz genutzt werden kann, um Brauchbarkeit von Software zu beurteilen.

In mehreren Arbeiten wird ein empathischer Gestaltungsbegriff benutzt, um zu verdeutlichen, dass Softwareentwicklung kein rein technisches Problem ist, etwa von Winograd und Flores [31, 30] und von Mathiassen und Dahlbohm [6]. In diesen Arbeiten wird ausgeführt, dass technische Lösungen gewöhnlich in einen sozialen Kontext eingebettet werden müssen.

5. Konsequenzen für die Ausbildung

Wenn unsere These und die dafür angeführten Argumente zutreffen, dann kommen auf die Ausbildung, und zwar auf die an der Hochschule ebenso wie auf die im Betrieb, neue Aufgaben zu. Den Lernenden muss Gelegenheit geboten werden, sich technische Kenntnisse und sogenannte "soft skills" nicht isoliert anzueignen, sondern sie müsen auch Anwendungsbedingungen der technischen Kenntnisse in der Praxis berücksichtigen können. Damit relevante arbeitspsychologische Kenntnisse wirklich integraler Bestandteil der Ausbilung und in der Konsequenz auch des Selbstbildes der künftigen Software-Ingenieure werden können, müssen sie auch im Zusammenhang mit den notwendigen technischen Kenntnissen vermittelt werden.

Etwa in der universitären Ausbildung eignen sich hierfür insbesondere im Team ausgeführte leidlich umfangreiche Softwarepraktika, in denen die Studierenden technische, planerische, lernerische und soziale Fertigkeiten im wechselseitigen Zusammenhang erproben und es lernen, Erfolge wie Misserfolge von ihren Betreuern unterstützt zu reflektieren und so auszuwerten, dass

die Erfahrungen in früheren zu Konsequenzen für spätere Praktikumsphasen werden [32, 33]. Gut reflektierte Industriepraktika haben die wichtige Funktion, bei der Einschätzung zu helfen, inwieweit die zuvor gewonnenen Einsichten über die universitäre Situation hinausreichen. Die Betreuung der inneruniversitären wie der Industriepraktika muss dabei die Aufmerksamkeit der Studierenden immer wieder von den allzu leicht dominierenden technischen Problemen und Fragestellungen zurück auf die gewöhnlich zugrundeliegenden organisatorischen, sozialen, oder allgemein: arbeitspsychologischen Einflussfaktoren lenken.

Literatur

1. Kent Beck. *Extreme Programming Explained: Embrace Chang*e. Addison Wesley Longman, Reading/Massachusetts, 1999.

2. Felix C. Brodbeck and Michael Frese. *Produktivität und Qualität in Software-Projekte*n. R. Oldenbourg, München, 1994.

3. F. P. Brooks. *The Mythical Man-Mon*th. Addison-Wesley, Reading/Massachusetts, 1975.

4. James O. Coplien. A generative development-process pattern language. In James O. Coplien and Douglas O. Schmidt (Hrsgg.), *Pattern Languages of Program Desig*n, S. 183–237. Addison-Wesley Longman, Reading/Massachusetts, 1995.

5. Wolfgang Coy. Für eine Theorie der Informatik! In Wolfgang Coy, Frieder Nake, Jörg-Martin Pflüger, Arno Rolf, Jürgen Seetzen, Dirk Siefkes, and Reinhard Stransfeld (Hrsgg.), *Sichtweisen der Informati*k, S. 17–32. Vieweg, Braunschweig, Wiesbaden, 1992.

6. B. Dahlbohm and L. Mathiassen. *Computers in Context - The Philosophy and Practice of Systems Desig*n. Blackwell Publishers, Cambridge/Massachusetts, 1993.

7. Christian Dahme and Arne Raeithel. Ein tätigkeitstheoretischer Ansatz zur Entwicklung von brauchbarer Software. *Informatik Spektru*m, 20(1):5–12, February 1997.

8. Tom DeMarco and Timothy Lister. *Peopleware*. Dorset House, New York, 1987.

9. E. W. Dijkstra. The humble programmer. *Communications of the AC*M, 15(10):859–866, October 1972.

10. M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journa*l, 3:182–211, 1976.

11. C. Floyd, F.-M. Reisin, and G. Schmidt. Steps to software development with users. In C. Ghezzi and J. A. McDermid (Hrsgg.), *ESEC'8*9, LNCS 387, S. 48–64. Springer-Verlag, 1989.

12. Christiane Floyd. On the relevance of formal methods to software development. *Proc. TAPSOFT/Formal Methods and Software Development:Berli*n, 2:1–11, March 1985.

13. Christiane Floyd. Outline of a paradigm change in software engineering. *Software Engineering Note*s, 13(2):25–38, April 1988.

14. Christiane Floyd, Anita Krabbel, Sabine Ratuski, and Ingrid Wetzel. Zur Evolution der evolutionären Systementwicklung: Erfahrungen aus einem Krankenhausprojekt. *Informatik Spektru*m, 20(1):13–20, 1997.

15. Watts S. Humphrey. *A Discipline for Software Engineerin*g. Addison-Wesley, Reading/Massachusetts, 1995.

16. Watts S. Humphrey. *Introduction to the Team Software Process*. Addison Wesley, Reading/Massachusetts., 2000.

17. Harlan D. Mills. Chief programmer teams: Principles and procedures. Technischer Bericht FSC 71-5108, IBM Federal Systems Division, Gaithersburg, Md./USA, 1971.

18. P. Naur and B. Randell (Hrsgg.). *Software Engineerin*g. Scientific Affairs Division NATO, Brussels, 1969.

19. P. Naur, B. Randell, and J. N. Buxton (Hrsgg.). *Software Engineering: Concepts and Techniqu*es. Petrocelli/Charter, New York, 1976.

20. Peter Naur. Formalization in program development. *BI*T, S. 437–453, 1982.

21. Jürgen Pasch. *Softwareentwicklung im Tea*m. Springer-Verlag, Berlin, 1994.

22. Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. Capability Maturity Model for software, version 1.1. Technischer Bericht CMU/SEI-93-TR-024, Software Engineering Insitute, February 1993.

23. Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, and Marilyn Busch. Key practices of the Capability Maturity Model, version 1.1. Technischer Bericht CMU/SEI-93-TR-025, Software Engineering Institute, February 1993.

24. W. W. Royce. Managing the development of large software systems. In *Proc. WESTCO*N, San Francisco, 1970.

25. Frank Simon and Claus Lewerentz. A product metrics tool integrated into a software development environment. In H. Coombes, M. Hooft van Huysduynen, and B. Peeters (Hrsgg.), *Proceedings of the European Software Measurement Conference (FESMA 98*), Antwerp,Belgium, 1998.

26. Walter Volpert. Work design for human development. In Christiane Floyd, Heinz Züllighoven, Reinhard Budde, and Reinhard Keil-Slawik (Hrsgg.), *Software Development and Reality Construcio*n, S. 336–348. Springer-Verlag, Berlin, 1992.

27. Gerald M. Weinberg. *The Psychology of Computer Programmin*g. Van Nostrand Reinhold, 1971.

28. Gerald M. Weinberg. *The Psychology of Computer Programming – Silver Anniversary Editio*n. Van Nostrand Reinhold, 1998.

29. Friedrich Weltz and Rolf G. Ortmann. *Das Softwareprojekt: Projektmanagement in der Praxi*s. Campus, Frankfurt, New York, 1992.

30. Terry Winograd (Hrsg.). *Bringing Design to Softwar*e. ACM Press, New York, 1995.

31. Terry Winograd and Fernando Flores. *Understanding Computers and Cognitio*n. Addison-Wesley, Reading/Massachusetts, 1986.

32. Claus Lewerentz, Heinrich Rust and Frank Simon. *A Model for Analyzing Measurement Based Feedback Loops in Software Development Projects*, Technical Report, BTU 2000/12, 2000

33. Claus Lewerentz and Heinrich Rust. *Die Rolle der Reflexion in Softwarepraktika*, Proceedings SEUH 2001, Teubner-Verlag, 2001