Xpert.press

Die Reihe **Xpert.press** vermittelt Professionals in den Bereichen Softwareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Mehr Informationen zu dieser Reihe auf http://www.springer.com/series/4393

Matthias Book • Volker Gruhn •
Rüdiger Striemer

Erfolgreiche agile Projekte

Pragmatische Kooperation und faires Contracting

Springer Vieweg

Matthias Book
Dept. of Computer Science
University of Iceland
Reykjavik
Island

Rüdiger Striemer
adesso AG
Berlin
Deutschland

Volker Gruhn
paluno – The Ruhr Institute for Software Technology
Universität Duisburg-Essen
Essen
Deutschland

Die Darstellung von manchen Formeln und Strukturelementen war in einigen elektronischen Ausgaben nicht korrekt, dies ist nun korrigiert. Wir bitten damit verbundene Unannehmlichkeiten zu entschuldigen und danken den Lesern für Hinweise.

ISSN 1439-5428
Xpert.press
ISBN 978-3-662-53329-1 ISBN 978-3-662-53330-7 (eBook)
DOI 10.1007/978-3-662-53330-7

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Springer Vieweg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Vieweg ist Teil von Springer Nature
Die eingetragene Gesellschaft ist Springer-Verlag GmbH Deutschland
Die Anschrift der Gesellschaft ist: Heidelberger Platz 3, 14197 Berlin, Germany

Vorwort

Industrielle Softwareentwicklung gehört zu den großen Erfolgsgeschichten des 20. Jahrhunderts. Anders hätte es nicht zur Durchdringung weiter Lebens- und Geschäftsbereiche mit Software kommen können, anders hätten die etablierten Geschäftsmodelle ganzer Branchen nicht durch Digitalisierung hinweggefegt werden können, anders wäre der weltweite Erfolg von Apple, Amazon, Google, Facebook und eBay nicht möglich.

Software Engineering, also die Konstruktion immer größerer Softwaresysteme auf der Grundlage ingenieursmäßiger Prinzipien, hat Softwaresysteme realisierbar gemacht, die jeweils ein paar Jahre zuvor unerreichbar schienen. Deshalb ist jede Art des grundsätzlichen Abstreitens dieser Erfolgsgeschichte geradezu absurd (Osterweil et al. 2008). Daran ändern auch zahlreiche, teilweise unter fadenscheinigen Umständen zustande gekommene Studien über den angeblichen Zustand der Softwareindustrie nichts, die z. B. von Eveleens und Verhoef (2010), Glass (2006) oder Jørgensen und Moløkken-Østvold (2006) entsprechend enttarnt werden.

Dennoch kommt es immer wieder zu Projekten, die in Schieflage geraten – manchmal, weil die bewährten Praktiken der Softwareentwicklung außer Acht gelassen wurden, manchmal, weil die handelnden Personen allzu optimistisch mit Ankündigungen und Versprechungen umgehen, und immer mal wieder, weil die vielen an einer Softwareentwicklung beteiligten Personen kein einheitliches Bild davon haben, worauf es im konkreten Projekt tatsächlich ankommt.

Dass das immer wieder geschieht und keine exotische Ausnahme ist, ist das eigentlich Verblüffende. Klar, Havarien passieren auch in anderen Projekten als in der Softwareentwicklung – Flughäfen werden nicht oder ganz arg verspätet fertig, öffentliche Baumaßnahmen werden teurer als geplant und Züge können nicht an allen Bahnsteigen halten. Echte Havarien – im Sinne von Vervielfachungen der Projektdauer oder -kosten, oder im Sinne von abgebrochenen oder rückabgewickelten Projekten – scheinen in der Softwareentwicklung jedoch häufiger als in anderen Branchen vorzukommen.

Vielleicht liegt das daran, dass der immaterielle Charakter von Software die Einschätzbarkeit des Projektzustandes erschwert und die Verschwendung durch einen Abbruch weniger greifbar macht. Vielleicht liegt es auch daran, dass Softwareentwicklungsprojekte (in denen die relevante Investitionsgröße ja „nur“ die Personalkosten sind) oft überambitioniert sind und sich nicht im nötigen Maße auf schlanke Lösungen konzentrieren.

Vielleicht liegt es auch daran, dass die Frage nach der Natur des Softwareprozesses immer noch nicht einheitlich beantwortet ist. Ist er im Wesentlichen ein Fertigungsprozess? Dann kann er Gegenstand tayloristischer Strukturierung und detaillierter Vorgaben sein, so wie die Erstellung von Autos am Fließband. Oder ist er ein rein kreativer Prozess, der einzig vom gestalterischen Talent des Entwicklers lebt? Dann machen prozessuale Vorgaben wenig Sinn, genauso wenig wie es einen präzisen Prozess zur Erstellung eines Gemäldes geben kann. Softwareentwicklung scheint sich zwischen diesen beiden Polen zu bewegen. Es gibt Teile, die klar reglementiert und variantenfrei sein sollten, zum Beispiel bestimmte Testaktivitäten oder das Konfigurationsmanagement. Andere sind nicht algorithmisch beschreibbar und kaum heuristisch unterstützbar, wie zum Beispiel das Vorgehen zur Identifikation möglichst früh zu entwickelnder Features.

Und dann ist da das Phänomen der Ungewissheit. Meir M. Lehman (1989) argumentierte bereits überzeugend, dass es in Softwareprojekten zu Ungewissheiten kommt, d. h. dass man im Laufe der Entwicklung auf Anwendungssituationen stößt, von denen bis dahin unbekannt oder zumindest ungewiss war, dass sie auftreten können und wie sie angemessen unterstützt werden sollen. Lehman erkannte auch, dass diese Stellen kaum im Vorhinein identifiziert werden können. Eine Beobachtung, die auch andere Autoren schon früh klar formulierten:

- „Ungewissheit ist im Softwareentwicklungsprozess und in Softwareprodukten inhärent und unvermeidlich." – Ziv et al.'s Ungewissheitsprinzip des Software Engineerings (1996)
- „In einem neuen Softwaresystem werden die Anforderungen solange nicht vollständig bekannt sein, bis man ein funktionierendes Produkt hat." – Humphreys Prinzip der Anforderungsungewissheit (1995)
- „Es ist unmöglich, ein interaktives System vollständig zu spezifizieren oder zu testen." – Wegners Lemma (1997)

Angesichts dieser Erkenntnis, die sich in praktisch jedem Softwareprojekt bestätigt, erscheinen Begriffe wie die „Software Factory" (Cusumano 1989) und Titel wissenschaftlicher Artikel wie „Software Processes are Software Too" (Osterweil 1987) irreführend, oder mindestens missverständlich. Softwareprozesse (zumindest für die Entwicklung soziotechnischer Systeme) sind erkenntnisgetriebene Prozesse, sie verfügen über mehr kreative als algorithmische Anteile, und es kann als sicher gelten, dass sie nicht präzise vorhersehbar sind (Gruhn und Urbainczyk 1998).

Dies soll in keiner Weise in Abrede stellen, dass es Arten von Software gibt, die sehr wohl vollständig beschreibbar sind. Eingebettete Systeme ohne Schnittstelle zum Menschen sind beispielsweise sehr wohl vollständig spezifizierbar und können gemäß des Fertigungsparadigmas erstellt werden.

Genau das gilt aber für soziotechnische Systeme nicht – ganz einfach deshalb, weil solche Systeme nicht am Bildschirm enden, sondern sich bis zur hinteren Schädeldecke der Anwender erstrecken. Das bedeutet nicht nur, dass die Software auf unvorhergesehenes

Anwenderverhalten vorbereitet sein muss. Viel wichtiger: In soziotechnischen Systemen ist die Software nur ein kleiner Teil eines Systems menschlicher und maschineller Akteure, die zusammen komplexe Prozesse abwickeln. Dieses Zusammenspiel, in das Software sich nahtlos integrieren muss, ist jedoch kaum vollständig beschreibbar und zudem ständiger Änderung unterworfen. Und gerade wenn es um Innovation, um die Etablierung neuer Geschäftsprozesse, neuer Dienstleistungen und die Realisierung neuartiger Automatisierungen geht, ist die Konzeption, Implementierung und Adaption von Software ein kreativer Prozess, dessen Zielsetzung zudem kontinuierliche Kalibrierung erfordert. Die Entwicklung solcher Softwarelösungen ist eben kein Fertigungsprozess, sondern ein Erkenntnisprozess, der genau dann beste Chancen auf Erfolg hat, wenn alle Beteiligten das Ziel im Auge behalten und auf schlanke Lösungen achten.

Auch wenn diese Lösungen technischer Natur sind, ist das Ziel, das sie unterstützen sollen, jedoch nicht in der Informationstechnik (IT), sondern in der Anwendungsdomäne verankert. In Unternehmen, die Software entwickeln, ist enge Kommunikation von Enterprise IT[1] und Fachbereichen daher unvermeidbar und unverzichtbar für den Erfolg. Ganz oft ist sie aber auch schwierig, von unterschiedlicher Terminologie und vor allem von unterschiedlichen Arten der Abstraktion (und Abstraktionsfähigkeit) geprägt.

Die stetige Neujustierung der Projektidee, die kontinuierliche Abstimmung zwischen Enterprise IT und Fachbereich, und die Abkehr von der Idee der Software Factory (die ja vollständig planbare Softwareproduktion suggeriert) bringt jedoch einige unerfreuliche Erkenntnisse mit sich. Zum Beispiel die, dass es vorab keine vollständige Spezifikation geben kann (und dass das Streben danach zum Scheitern verurteilt ist), dass es späte Anforderungen gibt (also solche, die erst während der Entwicklung oder gar danach auftauchen), dass Budgetallokationen und Kostenschätzungen vorläufig sind, und dass man zu Beginn eines Projekts nicht genau weiß, was man wann und zu welchem Preis bekommt.

Und das alles muss wirklich noch sein? Fast 50 Jahre, nachdem der Begriff „Software Engineering“ geprägt wurde? Nach fast 50 Jahren, in denen das „Engineering“ in „Software Engineering“ einen Anspruch ausdrückt – nämlich den Anspruch von Reproduzierbarkeit, Verlässlichkeit und Kalkulierbarkeit? Es scheint so zu sein, denn Softwareentwicklung ist immer noch riskant, Projekte gehen immer noch schief, und wenn man nach den Ursachen sucht, stößt man immer wieder auf die gleichen Gründe: Fehlendes Verständnis der Anwendungsdomäne, falsche Prioritätensetzung, und mangelnde Kommunikation zwischen den Stakeholdern (Curtis et al. 1988). Softwareprozesse sind und bleiben Erkenntnisprozesse, an die aber die Ansprüche von Produktionsprozessen gestellt werden.

[1]Unter „Enterprise IT“ verstehen wir im Folgenden die IT-Abteilung eines Unternehmens oder externe Dienstleister, die diese Funktion wahrnehmen.

Aufbau und Zielgruppe dieses Buchs

Genau hier setzt dieses Buch an – an der Erkenntnisnatur der Softwareentwicklung, der Notwendigkeit einer einheitlichen Zielsetzung, der Konzentration auf schlanke Software, der Fokussierung auf Wertschöpfung, dem Weglassen von Unwichtigem. Es beschreibt konkrete Instrumente und Methoden dafür, wie alle Projektbeteiligten ein einheitliches Verständnis der zu erstellenden Software entwickeln können, wie sie ihre wirklich essenziellen Anforderungen ermitteln und mit Änderungen an diesem Verständnis und den Anforderungen umgehen können.

Der in Teil II beschriebene **Interaction Room** bringt dazu alle Beteiligten zusammen – nicht an einem Tisch, sondern in einem Raum, in dem gemeinsam Digitalisierungs- und Mobilisierungsstrategien entwickelt, Technologiepotenziale evaluiert, Softwareprojekte geplant und begleitet werden. Warum all das einen dedizierten Raum braucht? Weil die Stakeholder sich dort gegenüberstehen, statt sich Mails zu schreiben. Weil sie komplexe Zusammenhänge dort allgemeinverständlich skizzieren können, statt sie umständlich ausformulieren zu müssen. Weil nur Platz für das Wichtigste ist. Und weil Erkenntnisse weder im Kurzzeitgedächtnis noch in Dokumenten verschwinden, sondern direkt im Raum prägnant notiert werden und so präsent bleiben. Kurz: Weil Projekte dort sicht- und greifbar werden.

Das in Teil III beschriebene Vertragsmodell **adVANTAGE** stellt unterdessen sicher, dass ein so erkenntnisgetriebener und damit unscharfer Prozess wie Softwareentwicklung auch im kommerziellen Umfeld – d. h. in einer Beziehung zwischen Kunde und Dienstleister – nicht nur funktionieren, sondern florieren kann. Änderungen im Projektverlauf sind hier kein Grund für Stress, sondern normale Projektereignisse. Das Vertragsmodell sorgt dafür, dass die Beteiligten sich trotz (bzw. gerade mithilfe) aller Änderungen darauf konzentrieren, maximalen Nutzen zu erzeugen, schlanke Software zu erstellen und Risiken fair zu verteilen.

Wie das im Projektalltag funktionieren kann, zeigt in Teil IV das **Praxisbeispiel** der Entwicklung eines Bestandsführungssystems für eine private Krankenversicherung – ein komplexes System mit einer auf den ersten Blick nahezu unüberschaubaren Menge an fachlichen Anforderungen, gesetzlichen Rahmenbedingungen, Akteuren und Prozessen für Regel- und Spezialfälle, eingebettet in die historisch gewachsene IT-Landschaft eines Versicherungsunternehmens. Das Beispiel des Projektstarts und ersten Sprints zeigt, wie Mitarbeiter des Unternehmens und eines IT-Dienstleisters sich mit den Mitteln des Interaction Rooms einen Überblick über das Projekt verschafft haben, wie die Konzeption und Entwicklung begleitet und wie die Aufwände abgerechnet wurden.

Letztlich hängt der Erfolg eines jeden Softwareprojekts – jenseits der Anwendungsdomäne, jenseits der eingesetzten Technologie – an den **Fähigkeiten** der Projektbeteiligten. Nur wenn die Stakeholder bereit sind, miteinander zu reden, sich aufeinander einzulassen, unterschiedliche Wahrnehmungen von Wert- und Aufwandstreibern zu respektieren, Kompromisse zu schließen, innovative Lösungen zu verfolgen und politisches Taktieren beiseite zu lassen, können Instrumente wie der Interaction Room und adVANTAGE ihren

Nutzen entfalten. Teil V beschreibt daher abschließend das Anforderungsprofil, das an die Softwaretechniker, aber auch die Domänenexperten von heute zu stellen ist.

Auch wenn Vertragsgestaltung und Softwareentwicklung zwei verschiedenen akademischen Disziplinen entstammen, sind sie in der Praxis doch untrennbar miteinander verbunden, wo unabhängig von aller Theorie Menschen effektiv zusammenarbeiten müssen, um in einer nachhaltigen Geschäftsbeziehung ein erfolgreiches Produkt zu erstellen.

Dieses Buch wendet sich daher an CIOs, Projektmanager und Softwareentwickler in der Softwareentwicklungspraxis, die lernen wollen, wie sie effektiv mit der unweigerlichen Ungewissheit komplexer Projekte umgehen können, die ein besseres gegenseitiges Verständnis und eine bessere Kooperation mit ihren Kunden und Lieferanten erreichen wollen, und die ihre Projekte trotz der inhärenten Ungewissheit mit geringerem Risiko abwickeln wollen.

Danksagung

Die Autoren danken Simon Grapenthin für das Teilen seiner umfassenden Erfahrungen in der Durchführung von Interaction-Room-Workshops und dem Training von Interaction-Room-Coaches in einem breiten Spektrum von Anwendungsdomänen. Wir danken ebenfalls Sandra Delvos für die unzähligen Stunden, die in die Erstellung und Überarbeitung der Abbildungen in diesem Buch geflossen sind, sowie Alexander Lohberg und Anja Wintermeyer für ihre Hintergrundrecherchen.

Literatur

Curtis B, Krasner H, Iscoe N (1988) A field study of the software design process for large systems. Comm ACM 31(11):1268–1287. doi:10.1145/50087.50089

Cusumano MA (1989) The software factory: a historical interpretation. IEEE Software 6(2):23–30. doi:10.1109/MS.1989.1430446

Eveleens JL, Verhoef C (2010) The rise and fall of the Chaos report figures. IEEE Software 27(1):30–36. doi:10.1109/MS.2009.154

Glass RL (2006) The Standish report: does it really describe a software crisis? Comm ACM 49(8):15–16. doi:10.1145/1145287.1145301

Gruhn V, Urbainczyk J (1998) Software process modeling and enactment: an experience report related to problem tracking in an industrial project. In: Katayama T, Notkin D (Hrsg) ICSE'98: Proc 20th intl conf software engineering, S 13–21. doi:10.1109/ICSE.1998.671098

Humphrey WS (1995) A discipline for software engineering. Addison-Wesley, S 349

Jørgensen M, Moløkken-Østvold K (2006) How large are software cost overruns? A review of the 1994 Chaos report. Inform Software Tech 48(4):297–301. doi:10.1016/j.infsof.2005.07.002

Lehman MM (1989) Uncertainty in computer application and its control through the engineering of software. J Software Maint 1(1):3–27. doi:10.1002/smr.4360010103

Osterweil LJ (1987) Software processes are software too. In: Riddle WE (Hrsg) ICSE'87: Proc 9th intl conf software engineering, IEEE Computer Society Press, S 2–13

Osterweil LJ, Ghezzi C, Kramer J, Wolf AL (2008) Determining the impact of software engineering research on practice. IEEE Computer 41(3):39–49. doi:10.1109/MC.2008.85

Wegner P (1997) Why interaction is more powerful than algorithms. Comm ACM 40(5):80–91. doi:10.1145/253769.253801

Ziv H, Richardson DJ, Klösch R (1996) The uncertainty principle in software engineering. Technical Report UCI-TR-96-33, University of California, Irvine. http://www.ics.uci.edu/~ziv/papers/icse97.ps. Zugegriffen: 23. Febr. 2016

Inhaltsverzeichnis