Eine kurze Geschichte der Vorgehensmodelle[1]

Ralf Kneuper[2]

Abstract: Dieser Beitrag gibt einen Überblick über die Unterschiede und Gemeinsamkeiten von klassischen und agilen Vorgehensmodellen, um diese besser verstehen und nachvollziehen zu können. Insbesondere wird erläutert, woher die beiden Ansätze kommen und welche externen Einflüsse auf die Entwicklung eingewirkt haben Bei beiden Ansätzen sind die zu Grunde liegenden Ideen mittlerweile fast 60 Jahre alt, auch wenn die systematische Auseinandersetzung mit klassischen Vorgehensmodellen erst etwa Mitte der achtziger Jahre begann, mit agilen Vorgehensmodellen etwa zehn Jahre später.

1 Einleitung

Softwareprozesse und ihre Repräsentationen in Vorgehensmodellen haben im Laufe der Zeit erhebliche Veränderungen erfahren. Dieser Beitrag gibt einen Überblick über diese Entwicklung, mit besonderem Augenmerk auf den Ansätzen der „klassischen“ oder Plan-getriebenen Entwicklung einerseits und der agilen Entwicklung andererseits.

Eine andere Sichtweise auf zumindest einen Teil dieser Entwicklung hat der Autor in [Kn13] beschrieben.

2 Die „Frühzeit“ der Vorgehensmodelle

Erste Überlegungen zum Vorgehen bei der Softwareentwicklung entstanden von Beginn an, wobei die Softwareentwicklung sich in dieser Frühzeit weitgehend auf die Programmierung beschränkte.

Beispielsweise hat schon Alan Turing in [Tu51, Abschnitt „Programming Principles“] folgendes aus heutiger Sicht sehr einfache Vorgehen beschrieben:

- Make a plan
- Break the problem down
- Do the programming of the new subroutines

[1] Dieser Beitrag ist eine ergänzte und leicht überarbeitete Fassung des Beitrags „Klassische und agile Vorgehensmodelle – Ein historischer Überblick“ in: M. Engstler et al. (Hrsg.): Projektmanagement und Vorgehensmodelle 2015. Lecture Notes in Informatics (LNI), Köllen Verlag, 2015.

[2] Beratung für Softwarequalitätsmanagement und Prozessverbesserung,
Philipp-Röth-Weg 14, 64295 Darmstadt, ralf@kneuper.de

- Programme the main routine

Einige Jahre später, als die Entwicklungsaufgaben allmählich umfangreicher wurden, stellte Benington bei einer Konferenz 1956 ein erstes Phasenmodell der Entwicklung vor, in dem bereits Begriffe wie „Spezifikation“, „Design“ und „Test“ auftauchen (siehe Abbildung 1). Breiter veröffentlicht wurde dieses Phasenmodell 1983 in [Be83].

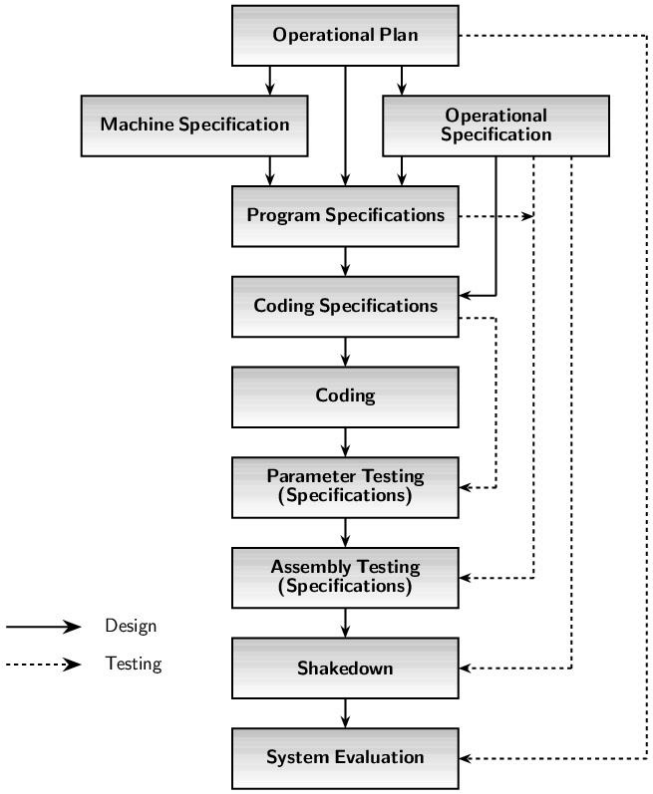

Abbildung 1: Beningtons Program Production Lifecycle [Be83]

Eingesetzt wurde dieses Phasenmodell im Projekt „Semi-Automatic Ground Environment (SAGE)“, einem sehr anspruchsvollen Projekt zur Entwicklung eine militärischen Echtzeitsystems.

Aus heutiger Zeit missverständlich sind die Phasen „Coding Specification“ und „Coding“, da sich die Bedeutung dieser Begriffe seither deutlich geändert hat. Die „Coding Specification“ entspricht etwa der heutigen Programmierung in einer der üblichen Programmiersprachen, während „Coding“ die Übersetzung der Coding Specification in Maschinencode umfasst, also eine Aktivität, die heute normalerweise von einem Compiler durchgeführt wird und keine eigene Entwicklungsphase darstellt.

Fast zur gleichen Zeit, um 1957, wurde aber auch schon mit einer iterativen, XP-ähnlichen Vorgehensweise gearbeitet, wie Gerald M. Weinberg in [LB03] berichtet.

Beide Arbeiten hatten aber zu dieser Zeit noch wenig Einfluss auf die weitere Entwicklung, wie sich schon an den erst wesentlich später erschienen Veröffentlichungen erkennen lässt. Trotzdem wurden diese Ideen in den Folgejahren auch von anderen Autoren allmählich weiter ausgebaut.

Beispielsweise beschrieb Hosier 1961 in seinem Artikel [Ho61] zur Entwicklung von Echtzeitsystemen die relevanten Aktivitäten in einem Flussdiagramm, allerdings auf einer relativ detaillierten Ebene und ohne eine Struktur entsprechend dem Modell von Benington.

1967 war die Entwicklung von Informationssystemen etwas weiter ausgereift und Rosove veröffentlichte das erste dem Autor dieses Beitrags bekannte Buch [Ro67] zu diesem Thema, in dem er ebenfalls ein Wasserfall-artiges Vorgehen beschrieb. Im Gegensatz zu Beningtons Modell enthält dieses Vorgehen explizite Feedback-Schleifen:

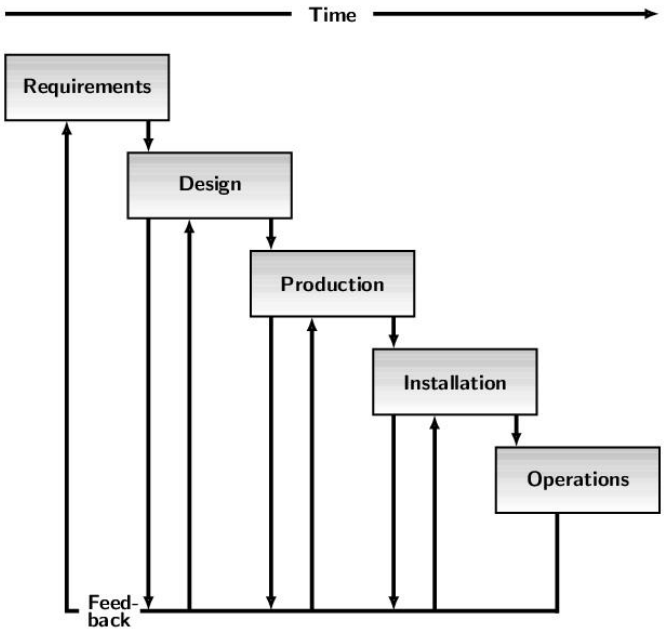

Abbildung 2 Wasserfallmodell nach Rosove (Ro67]

Als „die“ Referenz für Wasserfallmodelle wird heute allerdings meist das Vorgehensmodell von Royce, siehe [Ro70], bezeichnet. Royce beschreibt in der Tat einen sequentiellen Ablauf der Entwicklungsphasen, geht dabei allerdings explizit auch auf die Notwendigkeit von Rücksprüngen zu vorherigen Phasen, gelegentlich auch

weiter zurück, ein, d.h. es handelt sich nicht um ein streng sequentielles Vorgehen, auch wenn das später häufig so interpretiert wurde.

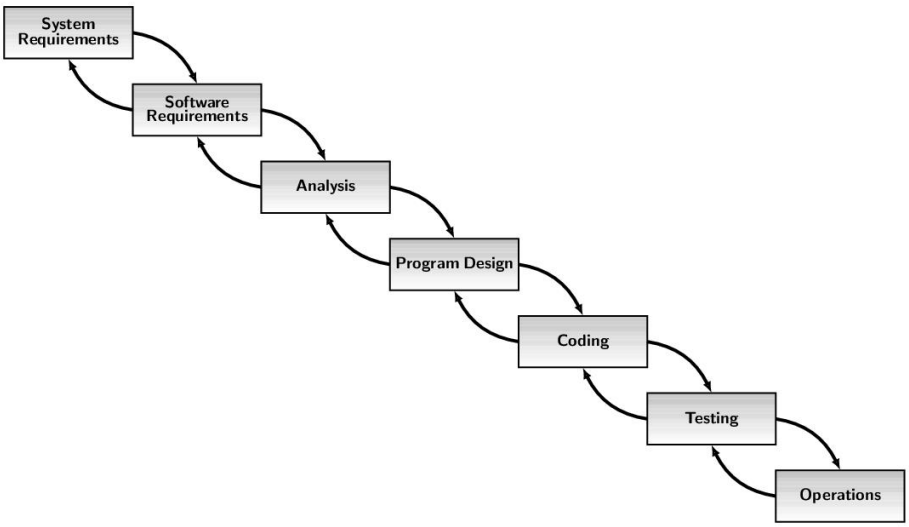

Abbildung 3 Wasserfallmodell nach Royce (Ro70]

Der Begriff des „Wasserfallmodells“ stammt ebenfalls nicht von Royce, sondern wurde etwas später geprägt von Bell und Thayer bei der Beschreibung dieses Vorgehens in [BT76]. Bekannt wurde dieser Begriff dann durch das Buch [Bo81] von Boehm.

Auch das iterative Vorgehen wurde danach immer wieder in der Literatur beschrieben und in der Praxis eingesetzt, siehe [LB03]. Alleine aus Kostengründen konnte dies aber nicht im gleichen Umfang geschehen wie heute, da die Rechenzeit wie auch Speicherplatz teure Ressourcen waren. Die erste *Beschreibung* eines iterativen Vorgehens in der Literatur stammt von Zurcher und Randell, siehe [ZR68]. Das von ihnen beschriebene Vorgehen begann mit einem abstrakten Systemmodell, dem dann schrittweise Details zugefügt wurden, bis das System schließlich in Form von Programmcode beschrieben war. Dies ist allerdings noch eine deutlich andere Interpretation des Begriffs der Iteration als heute in der agilen Entwicklung, bei der ein ausführbares System am Ende (fast) jeder Iteration ausgeliefert wird.

Bis etwa Mitte der achtziger Jahre allerdings standen Softwareprozesse und Vorgehensmodelle noch nicht selbst im Fokus der Betrachtung, sondern es mussten erst einmal die darin enthaltenen Aufgaben, Techniken und Methoden entwickelt werden. Insbesondere die aus der Systemanalyse entstandenen strukturierten Methoden entstanden in dieser Zeit, beispielsweise die „Structured Analysis and Design Technique“ (SADT), „Structured Analysis / Structured Design“ (SA/SD) und „Structured Systems Analysis and Design Method“ (SSADM). Grundlegende Bausteine dieser strukturierten Methoden

waren die Modellierung der Funktionalität, beispielsweise durch Funktionsbäume und hierarchische Prozessbeschreibungen mit Input, Verarbeitung und Output, oder durch Flussdiagramme; die Modellierung der Daten durch Entity-Relationship-Diagramme; und die Modellierung des Programmablaufs durch Kontroll- und Datenfluss-Diagramme oder Nassi-Shneidermann-Diagramme.

3 Mitte der achtziger Jahre: Beginnende systematische Betrachtung der Softwareprozesse

Ab etwa Mitte der achtziger Jahre wurden, nicht nur in der Softwareentwicklung, die Prozesse zur Erstellung von Produkten zunehmend systematischer betrachtet und als Hebel zur Verbesserung der zu erstellenden Produkte gesehen.

Dies führte beispielsweise 1987 zur Herausgabe der ersten Version der ISO 9000er-Normenreihe (siehe [PG12]) und etwas später zu IT-spezifischen Ansätzen wie CMM [Pa93], V-Modell [Hu90] und ITIL.

Auslöser für die systematische Beschäftigung mit Prozessen in der Softwareentwicklung, neben dem Anstoß aus dem allgemeinen Qualitätsmanagement heraus, waren u. a. die zu dieser Zeit erneut laufenden Diskussionen über die „Softwarekrise“ (siehe z.B. [We92]), sowie ab etwa Anfang der neunziger Jahre die auch in Unternehmen aufkommende objektorientierte Entwicklung (über die Nutzung objektorientierter Programmiersprachen hinaus).

Gemeinsam führten diese verschiedenen Auslöser zu einer intensiven Beschäftigung mit Vorgehensmodellen und Softwareprozessen in den neunziger Jahren. Schon 1984 begann die Tagungsreihe International Software Process Workshop (ISPW, siehe [Po84]), 1991 dann auch die Tagungsreihe der European Workshops on Software Process Technology (EWSPT), wenn auch zuerst noch unter etwas anderem Namen (siehe [FCA91]). In Deutschland erschien 1992 mit [Ch92] eines der ersten Bücher zu diesem Thema, 1993 wurde die Fachgruppe „Vorgehensmodelle für die betriebliche Anwendungsentwicklung“ der Gesellschaft für Informatik gegründet und organisierte 1994 ihren ersten Workshop (siehe [Kn13]). 1995 erschien erstmals die Zeitschrift „Software Process: Improvement and Practice“ bei Wiley.

Schwerpunkte der Arbeit in dieser Zeit waren die Modellierung und Anpassung (Tailoring) von Softwareprozessen sowie deren Unterstützung durch Werkzeuge, beispielsweise durch Process-Centered Software Engineering Environments (PCSEE, siehe z.B. [Gr02]), Werkzeuge für Computer-Aided Software Engineering (CASE) oder Integrated Project Support Environments (IPSE). An einem dieser Projekte, IPSE 2.5 (siehe [Wa90]), und dort speziell dem Teilprojekt mural (siehe [Jo91]) hat auch der Autor dieses Beitrags mitgearbeitet.

Die veröffentlichten Vorgehensmodelle in dieser Zeit kamen meist aus dem Umfeld

großer, komplexer Systeme, oft bestehend aus Hard- und Software, häufig auch im militärischen Umfeld.

In der Forschung und Entwicklung von Vorgehensmodellen wurde meist selbstverständlich von eher umfangreichen (später auch als „schwergewichtig“ bezeichneten) Vorgehensmodellen ausgegangen, die die Vorgehensweise bei der Entwicklung relativ detailliert vorgeben und diese Vorgaben ggf. auch durch entsprechende Werkzeuge durchsetzen. Osterweils bekannter Artikel [Os87] beschreibt diese Sichtweise recht gut als den Ansatz, Softwareprozesse zu „programmieren“.

Auch in der Anfang der neunziger Jahre aufkommenden objektorientierten Entwicklung war diese Sichtweise anfangs sehr verbreitet. Zwar änderten sich die verwendeten Entwicklungsmethoden, aus strukturierter Analyse und Design wurden objektorientierte Analyse und Design und schließlich die UML, aber auch hier versuchte man, die Vorgehensweise relativ detailliert festzulegen, beispielsweise im (Rational) Unified Process (siehe [Kr98], [JBR99]).

Bei der Umsetzung dieser Vorgehensweise gab es allerdings, neben den technischen Schwierigkeiten, auch das Problem der Akzeptanz durch die Entwickler. Überspitzt gesagt: Die Entwickler wollten keine „Computer“ sein, die Prozess-Programme ausführen. Die Frage nach den sozialen Aspekten, der Akzeptanz und der passenden Vorgehensweise zur Einführung von Softwareprozessen in Entwicklungsorganisationen war daher ein weiteres wesentliches Thema der Forschung und der Diskussion.

4 Ende der neunziger Jahre: Leichtgewichtige und agile Prozesse

Die genannten Akzeptanz-Probleme wurden von vielen Vertretern der „klassischen“ Softwareprozesse als eine Frage der Motivation der Entwickler betrachtet. Mit der Zeit setzte sich aber immer mehr die Einsicht durch, dass diese Sichtweise zu einfach war und die Kritik an diesen Softwareprozessen durchaus ihre Berechtigung hatte. DeMarco und Lister beispielsweise argumentierten schon 1987 in [DL87] gegen die „Method madness“ und wiesen darauf hin, dass man für Softwareentwickler als Wissensarbeiter andere Management-Methoden benötigt als in der Produktion. Damit ist auch die Frage nach einer anderen Unternehmenskultur verbunden, die weit über die reine Softwareentwicklung hinausgeht, und die gemeinsame Ausrichtung an einem Ziel in den Mittelpunkt stellt, im Gegensatz zur zentral gesteuerten und Plan-getriebenen Arbeitsweise. Als Beispiel für diese in den achtziger Jahren laufende Diskussion sei hier der Bestseller von „Thriving on Chaos“ von Tom Peters [Pe87] genannt.

Wesentliche Anregungen kamen in diesem Zusammenhang auch aus dem gerade entstehenden Thema Wissensmanagement. Beispielsweise entwickelten Nonaka und Takeuchi ihr Modell des Wissensmanagements und führten dabei bereits 1986 in [TN86] Scrum als Arbeitsweise für wissensintensive Arbeiten ein. Ken Schwaber und Jeff Sutherland übernahmen diese Arbeitsweise für die Softwareentwicklung, passten sie an

und veröffentlichten sie in [Sc97].

Darüber hinaus wurde immer stärker in Frage gestellt, ob eine Klärung der Anforderungen sowie eine Definition der zu verwendenden Architektur vorab wirklich sinnvoll und realistisch möglich sind, da beide im Laufe der Entwicklung immer wieder mehr oder weniger stark überarbeitet oder korrigiert werden müssen.

Ein weiterer Kritikpunkt war, dass die Möglichkeit zu Rücksprüngen zu früheren Phasen in einem Wasserfall-artigen Vorgehen nicht ausreicht und Iterationen und evolutionäres Vorgehen explizit im Vorgehensmodell berücksichtigt werden sollten. Dies führte beispielsweise zu Ansätzen wie Prototyping [Fl84], Boehms Spiralmodell [Bo86] und etwas später zum Rapid Application Development [Ma91].

Im Lauf der neunziger Jahre wurden diese Ideen weiter ausgebaut und setzten sich nun auch in der Breite durch. Dies führte neben dem produktiven Einsatz der genannten Methoden zu einer Vielzahl von neuen Methoden wie Extreme Programming (XP, siehe [Be99]), Dynamic Systems Development Method (DSDM), die Crystal-Methodenfamilie, Feature Driven Develoment (FDD) etc. In unterschiedlichen Ausprägungen und Schwerpunktsetzungen erweiterten diese den Ansatz der iterativen und inkrementellen Entwicklung und legten Wert auf die heute als „agile Methoden“ bezeichneten Prinzipien wie intensive Kommunikation im Projekt, häufige Iterationen, schnelle Rückmeldungen, wenig externe Festlegungen des Projektablaufs etc.

Eine ähnliche, wenn auch nicht ganz so ausgeprägte, Gegenbewegung gegen eine stark Plan- und Prozess-getriebene Arbeitsweise gab es auch in der Industrieproduktion, wo Anfang der neunziger Jahre das Konzept der „Lean Production“ aufkam, angestoßen in erster Linie durch [WJR91]. Viele der Grundideen der Lean Production wurden dann auch in den agilen Methoden in der Softwareentwicklung aufgegriffen.

Unterstützt wurden diese neuen Ansätze durch die ab Mitte der neunziger Jahre stark wachsende Bedeutung des Internets, genauer gesagt des WWWs, und die damit einher gehende wachsende Bedeutung kleiner, zeitkritischer Projekte mit unklaren und schnell wechselnden Anforderungen.

Anfangs wurden diese Vorgehensweisen meist als „leichtgewichtig“ bezeichnet, im Gegensatz zu den „schwergewichtigen“ Modellen, die die Entwicklungsprozesse relativ genau vorschreiben. Im Februar 2001 gab es dann ein Treffen der Vertreter der verschiedenen leichtgewichtigen Prozesse, bei dem das Agile Manifesto entstand (siehe [Hi01], [Be01]). Damit wurde dann auch die Bezeichnung der „agilen“ Methoden eingeführt, der sich sofort durchgesetzt hat, da viele der Vertreter mit der Bezeichnung als „leichtgewichtige“ Methoden eher unglücklich waren.

Durch diese Einigung auf gemeinsame Werte und Prinzipien erreichten die agilen Methoden nochmals eine stark wachsende Bedeutung, wobei es in der Anfangszeit sehr viele Grundsatzdiskussionen und „Religionskriege“ über die Angemessenheit klassischer, Plan-getriebener Entwicklung einerseits und agiler Entwicklung andererseits

gab. Verschärft wurde dies durch die auch heute noch gelegentlich anzutreffende pseudo-agile Denkweise, gemäß der agiles Vorgehen im Wesentlichen daraus besteht, wenig oder nichts zu dokumentieren.

Erst allmählich setzte sich die Einsicht in der Breite durch, dass beide Ansätze ihre Stärken und Schwächen haben, dadurch für unterschiedliche Typen von Projekten geeignet sind, und sich auch nicht notwendig widersprechen, sondern in vielen Fällen eine Kombination von Aspekten beider Ansätze sinnvoll ist. Zu dieser Sichtweise haben insbesondere Boehm und Turner mit ihrem Buch [BT04] beigetragen, in dem sie Kriterien für die Auswahl und Kombination herausarbeiten.

5 Softwareprozesse heute

Erfreulicherweise setzt sich die Sichtweise von agiler und Plan-getriebener Entwicklung als Enden eines Spektrums mit vielen Zwischentönen und Kombinationsmöglichkeiten immer mehr durch. Es gibt zwar immer noch gelegentlich Vertreter der einen oder anderen Seite, die nur „ihren“ Ansatz als praktikabel und nützlich gelten lassen, aber diese werden weniger. Immer mehr Projekte versuchen, von beiden Ansätzen das für ihre Aufgabenstellung am besten geeignete zu übernehmen (hybrides Vorgehen).

Das zeigt sich beispielsweise darin, dass auch bei großen, komplexen Projekten mit erheblichem Hardwareanteil, die ohne gründliche Planung kaum Aussicht auf Erfolg hätten, die Projektdauer bzw. der Zeitraum zwischen den Releases kürzer und nur noch in Ausnahmefällen nach Jahren gemessen wird.

Bis vor kurzem gab es allerdings kaum Untersuchungen, die sich kritisch-offen mit den verschiedenen agilen Ansätzen und deren Leistungsfähigkeit, aber auch ihren Einschränkungen, auseinander setzten. Das hat sich erfreulicherweise mit dem 2014 erschienenen Buch [Me14] von Bertrand Meyer geändert.

Bei den in Unternehmen genutzten Vorgehensmodellen gibt es eine erhebliche Bandbreite, wobei zumindest in Deutschland Scrum sowie V-Modell, meist in organisations- oder projektspezifisch angepasster Form, überwiegen [KL15].

Bei den neuen Entwicklungen seien hier beispielhaft Kanban und DevOps genannt. Kanban ist dem agilen Vorgehen verwandt und kommt aus dem Konzept der Lean Production in der industriellen Produktionssteuerung, hat mittlerweile aber auch in der Softwareentwicklung erheblich an Bedeutung gewonnen. Hauptziel von Kanban ist es, die Anzahl der parallel bearbeiteten Aufgaben zu reduzieren und dadurch die Durchlaufzeit bei der Umsetzung von Aufgaben bzw. Anforderungen zu reduzieren.

DevOps adressiert dagegen die verbreiteten Schwierigkeiten bei der Kommunikation zwischen den für die Entwicklung und den Betrieb von Software verantwortlichen Gruppen, insbesondere bei der agilen Entwicklung, wo sehr häufig neue Versionen von Software ausgeliefert werden, die dann möglichst kurzfristig in Betrieb gehen sollen.

Ziel von DevOps ist es daher, eine durchgängige und weitgehend automatisierte Lieferkette („Pipeline“) aufzubauen, die von der Klärung der Anforderungen bis zur Inbetriebnahme neuer Funktionalität reicht und dabei die Qualität der gelieferten Funktionalität sicherstellt.

Literaturverzeichnis

[Be83] Benington, Herbert D.: *Production of Large Computer Programs*. IEEE Annals of the History of Computing 5 (4): 350–361. 1983. Verfügbar unter http://sunset.usc.edu/csse/TECHRPTS/1983/usccse83-501/usccse83-501.pdf

[Be99] Beck, Kent: *Extreme Programming Explained – Embrace Change*. Addison-Wesley, 1999.

[Be01] Beck, Kent et al.: *Manifesto for Agile Software Development*. http://agilemanifesto.org/. 2001.

[Bo81] Boehm, Barry W.: *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, 1981.

[Bo86] Boehm, Barry W.: *A Spiral Model of Software Development and Enhancement*. ACM SIGSOFT Software Engineering Notes, ACM, 11(4). S. 14-24, August 1986.

[BT76] Bell, T. E.; Thayer, T. A.; *Software requirements: Are they really a problem?* in Proceedings of the 2nd International Conference on Software Engineering, San Francisco, California, USA, October 13-15, 1976. 1976, S. 61-68.

[BT04] Boehm, Barry W.; Turner, Robert: *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison-Wesley, 2004.

[Ch92] Chroust, Gerhard: *Modelle der Software-Entwicklung*. Oldenbourg Verlag, 1992.

[DL87] DeMarco, Tom; Lister, Timothy: *Peopleware. Productive Projects and Teams*. Dorset House Publishing Co., 1987.

[Fl84] Floyd, Christiane: *A systematic look at prototyping*. In (Budde et al., Hrsg.): Approaches to Prototyping; Proc. Namur. S. 1-18. Springer-Verlag, 1984.

[FCA91] Fuggetta, A.; Conradi, R.; Ambriola, V. (Hrsg.): *First European Workshop on Software Process Modeling : CEFRIEL, Milan (Italy), 30-31 May 1991*. Associazione Italiana per l'Informatica ed il Calcolo Automatico, Working Group on Software Engineering, 1991. (cf. http://trove.nla.gov.au/work/22457426)

[Gr02] Gruhn, Volker: *Process-Centered Software Engineering Environments. A Brief History and Future Challenges*. Annals of Software Engineering 14. S. 363–382, 2002.

[Hi01] Highsmith, Jim: History: *The Agile Manifesto*. http://agilemanifesto.org/history.html. 2001.

[Ho61] Hosier, W.A.: *Pitfalls and safeguards in real-time digital systems with emphasis on programming*. IRE Transactions on Engineering Management, Vol. EM-8, S. 99–115, Juni 1961.

[Hu90] Hummel, Helmut: *The Life Cycle Methodology for Software Production and the Related Experience*. In (Ehrenberger, W., Hrsg.): Approving Software Products. Proc. of the IFIP WG 5.4 Working Conference. North-Holland, 1990.

[JBR99] Jacobson, Ivar; Booch, Grady; Rumbaugh, James: *The Unified Software Development*

Process. Addison-Wesley, 1999.

[Jo91] Jones, Cliff .B.; Jones, Kevin D.; Lindsay, Peter A.; Moore, Richard: *mural. A Formal Development Support System*. Springer-Verlag, 1991.

[KL15] Kuhrmann, Marco; Linssen, Oliver: *Vorgehensmodelle in Deutschland. Nutzung von 2006-2013 im Überblick*. WI-MAW-Rundbrief, April 2015.

[Kn13] Kneuper, Ralf: *Zwanzig Jahre GI-Fachgruppe "Vorgehensmodelle für die betriebliche Anwendungsentwicklung"*. In Vorgehensmodelle 2013, Lecture Notes in Informatics, Vol. P-224, Gesellschaft für Informatik, Bonn. S. 17-30, 2013.

[Kr98] Kruchten, Philippe: *The Rational Unified Process: An Introduction*. Addison-Wesley, 1998.

[LB03] Larmann, Craig; Basili, Victor R.: *Iterative and Incremental Development: A Brief History*. IEEE Computer, Juni 2003.

[Ma91] Martin, James: *Rapid Application Development*. Macmillan, 1991.

[Me14] Meyer, Bertrand: *Agile! The Good, the Hype and the Ugly*. Springer, 2014.

[Os87] Osterweil, Leon: *Software Processes are Software Too*. In Proceedings of the Ninth International Conference on Software Engineering, 1987.

[Pa93] Paulk, Mark et al.: *Capability Maturity Model*^{SM} *for Software, Version 1.1*. Technical Report CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie Mellon University, 1993.

[Pe87] Peters, Tom: *Thriving on Chaos. Handbook for a Management Revolution*. Alfred A. Knopf, Inc., 1987.

[PG12] Petrick, Klaus; Graichen, Frank: *25 Jahre ISO 9001. Erfolgsweg einer Systemnorm*. In QZ Qualität und Zuverlässigkeit, Heft 3, 2012, S. 3-5. Verfügbar unter http://www.qz-online.de/qz-zeitschrift/archiv/artikel/erfolgsweg-einer-systemnorm-271979.html.

[Po84] Potts, Colin (Hrsg.): *Proceedings of a Software Process Workshop, February 1984, Egham, UK*. IEEE Computer Society, 1984.

[Ro67] Rosove, Perry E.: *Developing Computer-Based Information Systems*. John Wiley and Sons, Inc, 1967.

[Ro70] Royce, Winston W.: *Managing the Development of Large Software Systems*. In Proceedings, IEEE Wescon August 1970. S. 1-9. 1970.

[Sc97] Schwaber, Ken: *SCRUM Development Process*. In OOPSLA ’95 Workshop Proceedings 16 October 1995, Austin, Texas. Springer, 1997. Verfügbar unter http://jeffsutherland.org/oopsla/schwaber.html.

[TN86] Takeuchi, Hirotaka; Nonaka, Ikujiro: *The New New Product Development Game*. Harvard Business Review, Januar 1986.

[Tu51] Turing, Alan M.: *Programmer’s Handbook for Manchester Electronic Computer Mark II*. Faksimile verfügbar unter http://www.alanturing.net/turing_archive/archive/m/m01/M01-001.html. Transkript verfügbar unter http://curation.cs.manchester.ac.uk/computer50/www.computer50.org/kgill/mark1/RobertTau/turing.html. Undatiert,

erstellt um 1951.

[Wa90] Warboys, Brian C.: *The IPSE 2.5 Project: Process Modelling as a Basis for a Support Environment*. In (Madhavji, N.; Schäfer, W.; Weber, H., Hrsg.): Proceedings of the First International Conference on System Development Support Environments and Factories, S. 59-74, Pitman, 1990. Beitrag verfügbar unter http://apt.cs.manchester.ac.uk/ftp/pub/IPG/bw89.pdf.

[We92] Weber, Herbert: *Die Software-Krise und ihre Macher*. Springer-Verlag, 1992.

[WJR91] Womack, James P.; Jones, Daniel T.; Roos, Daniel: *The Machine That Changed the World : The Story of Lean Production*. Harper Perennial, 1991.

[ZR68] Zurcher, F. W.; Randell, B.: *Iterative multi-level modeling — a methodology for computer system design*, in Proc. IFIP Congress 68. IEEE CS Press, 1968, S. 138–142.