Formale Systemmodelle der Informatik als Basis für eine bessere Beherrschung von KPM-Szenarien

Jan Friedrich

4Soft GmbH
Mittererstr. 3
80336 München
friedrich@4soft.de

Abstract: Kollaborativ gemanagte Projekte (KPM-Szenarien) werden noch nicht ausreichend durch Werkzeuge und Vorgehensmodelle unterstützt. Ursache sind u.a. eine fehlende theoretische Durchdringung der PM-Domäne und die Fokussierung der SW-Prozessforschung auf Prozessautomatisierung. Die guten Erfahrungen mit der Übertragung formaler Systemmodelle auf Prozesse sollten herangezogen werden, um auch Planung und Kontrolle auf diese Weise formal-theoretisch zu durchdringen. Der vorliegende Impulsbeitrag führt in die oben formulierten Thesen ein, belegt sie und leitet in die Diskussion zum Thema über.

1 Einführung und Thesen

Kollaboratives Projektmanagement (KPM) dient zur Planung, Kontrolle und Steuerung[1] großer, komplexer und sowohl räumlich als auch organisatorisch verteilter Projekte und gewinnt zunehmend an Bedeutung [CN+03]. Statt eines einzelnen zentralen, alles überblickenden Projektmanagers existiert eine Hierarchie (bzw. ein Netz) von miteinander interagierenden Managern, die jeweils einzelne Projektteile mit eigenen Projektteams planen, kontrollieren und steuern und dabei mit den anderen Managern interagieren und kooperieren. Diese Definition macht deutlich, dass KPM Überschneidungen und Ähnlichkeiten mit Programm- und Projektportfoliomanagement, *Cross Company Planning* (CCP) und *Network-Centric Planning* (NCP) besitzt.

Zusätzlich zu den „normalen" Herausforderungen eines Projekts bestehen in KPM-Szenarien meist folgende Schwierigkeiten: 1.) *Größe und Komplexität* erfordern eine wohldurchdachte Projektstrukturierung und Schnittstellenmanagement. Außerdem muss ein adäquater Informationsfluss sichergestellt werden, sodass jeder Manager die notwendigen Informationen erhält, aber trotzdem nicht in Informationen „ertrinkt". 2.) Die einzelnen Teams stammen oft aus unterschiedlichen Organisationen und befolgen *unter-*

[1] Die genannten Begriffe werden im Rahmen dieses Beitrags als deutsche Äquivalente der entsprechenden Schritte im PDCA-Zyklus verwendet: Planung (PLAN), Kontrolle = Berichtswesen und Soll/Ist-Vergleich (CHECK) und Steuerungsentscheidungen (ACT).

schiedliche Prozess- und Vorgehensmodelle, die oft nicht kompatibel zueinander sind und im Projekt an den Schnittstellen ad-hoc miteinander abgeglichen werden müssen. Außerdem werden klassische und agile PM-Techniken oft im selben Projekt kombiniert. 3.) Die im Projekt miteinander kooperierenden Organisationen sind an anderer Stelle *Konkurrenten* oder stammen aus *unterschiedlichen Bündnissystemen* (z.B. deutsch-chinesische Kooperationsprojekte). Informationssicherheitsaspekte müssen deswegen auch im Projektmanagement beachtet werden, da beispielsweise Pläne auch Ingenieurs- oder Prozesswissen enthalten können [DS02]. Dabei existiert das Dilemma: Zu wenig Informationsweitergabe gefährdet den Projekterfolg, zu viel Informationsweitergabe schwächt ggf. die eigene Position gegenüber dem Konkurrenten.

Es existieren mannigfaltige Projekterfolgsfaktoren, die natürlich auch in KPM-Szenarien gelten. Im Folgenden sind lediglich drei ausgewählte näher beschrieben, weil sie in KPM-Szenarien schwer (bzw. schwerer) erreichbar sind: 1.) *Vorgehensmodelle* können ein Projekt positiv beeinflussen, wenngleich der Erfolg nicht immer statistisch signifikant ist [BE+10]. Wie bereits eingangs erwähnt, finden sich in KPM-Szenarien aber oft mehrere Vorgehensmodelle, die ggf. inkonsistent zueinander sind. 2.) Die *Kontrollfrequenz* (also die Häufigkeit des Soll/Ist-Vergleichs) beeinflusst den Projekterfolg positiv [BE+10]. Bei größeren Projekten braucht es aber Zeit, entsprechende Berichte in einer Projekthierarchie „von unten nach oben“ (bzw. in einem Netzwerk „von links nach rechts“) weiterzuleiten, zu filtern und zu verdichten, insbesondere wenn dies manuell erfolgt. Je größer ein Projekt ist, desto kleiner kann in der Regel die Kontrollfrequenz sein. 3.) Die *Planqualität* hat ebenfalls einen positiven Einfluss auf den Projekterfolg, wie Dvir und Lechler in [DL04] belegen. Sie zeigen außerdem, dass die wahre Herausforderung nicht darin besteht, zu Beginn des Projekts einen „guten“ Plan zu besitzen, sondern diesen Plan über die Projektlaufzeit „gut“ zu erhalten. In KPM-Szenarien gibt es aber meist viele Planer, die nicht am selben Standort sind, unabhängig voneinander umplanen und sich darüber hinaus ihre jeweiligen Pläne nicht offenlegen. Inkonsistenzen sind vorprogrammiert, die Planqualität leidet.

Folgende zentrale Thesen bilden die Grundlage dieses Beitrags:

1.) Das theoretische Fundament des Projektmanagements ist – im Gegensatz zur praktischen Erfahrung in dem Bereich – mangelhaft ausgeprägt. Dies wirkt sich gerade in KPM-Szenarien negativ aus.

2.) Die Übertragung formaler Systemmodelle auf Software-Prozesse hat sich bewährt und wertvolle wissenschaftliche Beiträge hervorgebracht. Allerdings führte der Schwerpunkt der Prozessautomatisierung dazu, dass in der Praxis nicht Projekte, sondern insbesondere Geschäftsprozesse profitieren konnten.

3.) Formale Systemmodelle der Informatik lassen sich gewinnbringend auf KPM-Szenarien adaptieren. Dabei sollte vom Ziel der Planautomatisierung Abstand genommen werden. Vielmehr sollte der Schwerpunkt auf die Planung an sich (z.B. auf Qualitätsaspekte von Plänen) und die Unterstützung des Soll/Ist-Vergleichs gelegt werden.

2 Argumente und Belege

Die folgenden Unterkapitel liefern Argumente und Belege für die genannten Thesen. Jedes Unterkapitel ist dabei genau einer These zugeordnet.

2.1 Theoretisches PM-Fundament (These 1)

Koskela und Howell appellieren mit ihrem Beitrag [KH02] an die „PM-Gemeinde“, die theoretischen Grundlagen des Projektmanagements besser zu erforschen. Sie argumentieren, dass spätestens durch das Aufkommen agiler PM-Ansätze die PM-Theorie überholt ist, wenn sie überhaupt jemals in adäquater Form existiert hat. Sie belegen, dass das mangelhafte theoretische Fundament insbesondere bei großen, komplexen Projekten mit vielen Änderungen zum Problem wird. Unter anderen weisen sie auf die Planungsproblematik hin: Pläne werden zu Projektbeginn einmal erstellt und dann nicht weiter angepasst und aktuell gehalten – wenngleich sie deutlich machen, dass Pläne in der Praxis niemals 100%ig aktuell und konsistent sein können.

Alberts und Hayes gehen im Kontext militärischer Planung auf Unterschiede zwischen dem klassischen Planungsansatz mit einem alles überblickenden Planer und dem Network-centric Planning (NCP) mit vielen verteilten Planern ein [AH07]. Die Analyse und die Schlussfolgerungen können in weiten Teilen direkt auf klassisches PM und KPM übertragen werden. Die Autoren unterscheiden zwischen Planungsqualität und Planqualität und bemängeln ebenfalls eine mangelnde theoretische Durchdringung der Domäne. Zudem regen sie weitere Forschung beispielsweise zur Ableitung konkreter Qualitätsmetriken für Pläne und Planung an, um das NCP-Feld weiter zu entwickeln.

Auf Werkzeugseite sind insbesondere KPM-Software und Projektcockpits zu nennen. An diesen lässt sich das ausbaubedürftige theoretische PM-Fundament ebenfalls erkennen. Die Software RPlan unterstützt explizit KPM/CCP-Szenarien und ist in der deutschen Automobilindustrie weit verbreitet. Sie unterstützt die Konsistenthaltung von Plandaten zwischen verschiedenen Planern. Auf der Kontrollseite sieht das Werkzeug ein Ampelsystem vor. Ein Automatismus für das „Rechnen“ mit Projektampeln war in früheren Versionen vorhanden, scheiterte jedoch an mangelnder theoretischer Grundlage: Beispielsweise müsste eine solche Theorie eine Aussage darüber erlauben, ob und ggf. wie weit sich eine rote Ampel auf unterster Ebene nach oben durchschlägt.

Münch und Heidrich beschreiben in [MH04] den Stand in Praxis und Wissenschaft zum Thema Projektcockpits. Sie weisen auf grundsätzliche Defizite bei der Verknüpfung zwischen der Planung und dem Soll/Ist-Vergleich (Kontrolle) in einem Projektcockpit hin. Die in [RB+10] gezeigte Cockpit-Implementierung zeigt außerdem Probleme bei der Beherrschung der Informationssicherheit und damit bei der Anwendung in KPM-Szenaren: Repositorydaten aus dem gesamten Projekt werden zunächst in einem großen zentralen „Datentopf“ gesammelt, bevor daraus die Statusinformationen für die einzelnen Verantwortlichen abgeleitet werden.

2.2 Forschung zu SW-Prozessen und Vorgehensmodellen (These 2)

Seit den 1980er Jahren befasst sich das Software Engineering intensiv mit der Modellierung von Software-(Entwicklungs-)-Prozessen. Richtungsweisend war Osterweils Idee des *Process Programmings*: Er votierte dafür, die Erkenntnisse über Software-Programme auf den SW-Entwicklungsprozess selbst zu übertragen [Ost87]. In Folge entstanden viele wertvolle Beiträge (z.B. FUNSOFT Netze [EG91] oder Little-JIL [WC+11]), die aber zumeist das direkte Enactment von Vorgehensmodellen adressieren, sie also davon ausgehen, dass sich die Handlungen eines Mitarbeiters direkt (und idealerweise werkzeuggestützt) aus dem Prozess-/Vorgehensmodell ableiten lassen. Profitiert von den Ergebnissen haben insbesondere wiederholbare Prozesse (z.B. Geschäftsprozesse), die durch Workflow-Maschinen unterstützt werden können [FB11].

Beim planbasierten Enactment wird aus dem Vorgehensmodell zunächst ein Plan abgeleitet, aus dem sich dann die Handlungen der Mitarbeiter ableiten [FB11]. Diese Enactment-Variante ist prinzipiell kompatibel zum Projektmanagement, insbesondere wenn davon ausgegangen wird, dass der abgeleitete Plan ständig aktualisiert und an den tatsächlichen Projektverlauf angepasst werden muss. Obwohl in diese Richtung geforscht wurde (z.B. [BL+95] [Fis10] [Gna05]), genoss diese Variante weitaus weniger Aufmerksamkeit. Manche Beiträge gehen sogar so weit, sie nicht als „richtiges Enactment" zu zählen, weil sie nicht vollständig automatisiert abläuft und projektspezifische Informationen manuell hinzugefügt werden müssen [BC+07].

Auch Mischformen existieren, z.B. MILOS [MD+00]: Der Ansatz geht prinzipiell von einem Plan aus, der aus dem Vorgehensmodell abgeleitet werden muss. Allerdings wird hier der Plan als automatisierbare Arbeitsanweisung aufgefasst: Ein Planer erstellt den Plan und verlässt sich dann auf eine Workflow-Maschine, die das Projekt für ihn steuert und die Mitarbeiter „antriggert". Umplanungen sind vorgesehen; Soll/Ist-Vergleiche um Planabweichungen zu bestimmen, kommen in dem Konzept aber nicht vor.

Insgesamt ist in der Domäne ein Hang zur Prozess-/Planautomatisierung zu beobachten, die die Anwendbarkeit in KPM-Szenarien erschwert: Das angestrebte Ideal ist oft ein Unterstützungssystem, das die Mitarbeiterkoordination (z.B. die Zuweisung von ToDos) vornimmt. Die im PM so wichtigen Faktoren wie gemeinsame Planung, Vertrauen auf die Mitarbeiter (die ggf. auch bewusst vom Plan abweichen, weil ihnen eine bessere Lösung einfällt), Soll/Ist-Vergleiche, etc. werden meist nicht einbezogen.

2.3 Adaption formaler Systemmodelle (These 3)

Prinzipiell ist die Idee Osterweils dennoch richtig: Die Übertragung informatischer Konzepte auf Entwicklungsprojekte kann einen wichtigen Beitrag leisten, diese besser zu verstehen und mit Werkzeugen zu unterstützen. Betrachtet man KPM-Szenarien, dann bieten sich insbesondere komponentenbasierte Systemmodelle (z.B. [BS01]) als Basis für eine Adaption im Rahmen einer KPM-Theorie an, weil sie Prinzipien wie Abstraktion, Modularisierung, (De-)Komposition, Skalierbarkeit, Information Hiding und Schnittstellen von Haus aus bieten. Aus der Einführung dieses Beitrags ist ersichtlich, dass

diese Prinzipien auch in KPM-Szenarien anzutreffen sind, beispielsweise beim Schnittstellenmanagement oder bei Informationssicherheitsanforderungen.

Gemäß [Sta73] ist bei jedweder Modellierung das pragmatische Merkmal zu beachten. Hier sollte die Erkenntnis stehen, dass kein Plan perfekt sein kann und Planabweichungen oft unvermeidlich und manchmal sogar sinnvoll sind. Tritt man deshalb vom Automatisierungsanspruch zurück und betrachtet einen Plan insbesondere als Ergebnis der Planung und als Basis für den Soll/Ist-Vergleich, so hat dies Konsequenzen: Ein Plan muss nicht mehr (zwingend) ausführbar sein, sondern sollte insbesondere geeignet sein, Planungsschwächen zu identifizieren und den Soll/Ist-Vergleich zu ermöglichen. Im Sinne der Metapher des „Process Programmings" müssen Pläne damit nicht prozedural „ausformuliert" werden, sondern sie können deklarativ auf beliebiger Abstraktionsebene spezifiziert werden – Unter- und Überspezifikation eingeschlossen. Die Umsetzung des Plans geht von intelligenten Mitarbeitern aus, die mit ihrer Intelligenz und Erfahrung selbst dafür sorgen, sich möglichst im Rahmen der Deklaration zu bewegen. Sie können auch bewusst und eigenverantwortlich vom Plan abweichen, wenn es die Projektumstände erfordern, da sie keine Maschine zur Einhaltung des Plans „zwingt".

Die Vorteile einer solch deklarativen Betrachtungsweise sind anhand des „Feedback-Loop-Problems" [Mau96] erkennbar: Wie Abbildung 1 zeigt, seien die Tasks A und B wechselseitig voneinander abhängig: Es sollen zwei Dokumente entstehen, die sich gegenseitig referenzieren. In der herkömmlichen Workflow-Betrachtungsweise mit Eingabe- und Ausgabe-Artefakten und einem Taskbeginn erst dann, wenn alle Eingaben vorliegen (z.B. [MD+00]) ist dafür keine Lösung zu finden (vgl. auch [Mau96]). Wenn Alice ihr Artefakt fertig stellt und es als Input für Bobs Task dient, dann kann Bob zwar Alice' Dokument referenzieren, aber nicht mehr umgekehrt. Wenn die Tasks hingegen wechselseitig aufeinander warten, entsteht ein Deadlock.

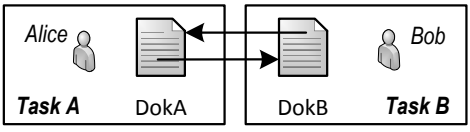

Abbildung 1: Feedback-Loop-Problem

Mit der oben beschriebenen deklarativen Herangehensweise kann das Problem sinnvoll als Plan modelliert werden (siehe Abbildung 2). Die Semantik der Ports und Konnektoren ist dabei kein Artefaktfluss, sondern eine Sichtbarkeit auf Artefakt-Repositorys: Alice macht Bob alle als „DokA" gekennzeichneten Artefakte sichtbar – dasselbe gilt umgekehrt für Bob und die Kennzeichnung mit „DokB". Alice sichert außerdem zu, dass sie für folgendes Sorge trägt: *Falls bei Bob mindestens ein „DokB" für sie sichtbar ist, dann existiert in ihrem Repository genau ein mit „DokA" beschriftetes Artefakt und dieses Artefakt referenziert alle von Bob mit „DokB" gekennzeichneten Artefakte.* Bob garantiert das Gleiche in umgekehrter Richtung.

Wenn zu einem beliebigen Zeitpunkt im Projekt nun überprüft wird, ob der Plan erfüllt ist, so existieren genau zwei Möglichkeiten, in denen sowohl Bob als auch Alice *gleichzeitig* ihre Zusicherungen erfüllen: Entweder keiner von beiden hat ein Dokument pro-

duziert, oder sie haben gemeinsam genau den erwünschten Zustand erreicht. *Wie* die beiden nun im Detail kooperieren, um ihre Ziele gemeinsam zu erreichen, ist bewusst offengehalten, um hier die Intelligenz und Verantwortung der Projektmitarbeiter zu nutzen. Der Ansatz ist damit vergleichbar mit deklarativen Sprachen wie PROLOG oder SQL bei denen der Programmierer nur die Eigenschaften der Lösung spezifiziert, ohne den Lösungsweg vorzugeben.

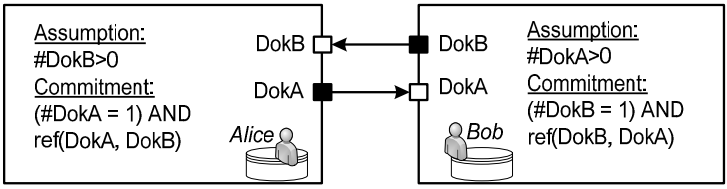

Abbildung 2: Deklarative Lösungsbeschreibung des Feedback-Loop-Problems

Auf Basis dieser Sichtweise können nun komplexere Planungsprobleme angegangen werden. Durch Übertragung bekannter Konzepte wie Verfeinerung, Korrektheit, oder Realisierbarkeit (vgl. [BS01]) können beispielsweise Pläne hinsichtlich ihrer Qualitätseigenschaften charakterisiert werden, um damit u.a. die in [AH07] genannten Forschungslücken zu adressieren.

3 Zusammenfassung und Diskussion

KPM-Szenarien sind herausfordernd und werden bisher nicht ausreichend durch Werkzeuge und Vorgehensmodelle unterstützt. Für eine bessere Unterstützung ist zunächst eine bessere theoretische Durchdringung der PM-Domäne und insbesondere der Planung notwendig. Die Informatik hat in der Software-Prozess-Forschung bewiesen, dass eine Übertragung ihrer formal-mathematischen Prinzipien von Computersystemen auf Softwareprozesse enorme Fortschritte und Einsichten bringen kann. Allerdings lag der Forschungsschwerpunkt meist auf der Prozessautomatisierung, weniger auf der Unterstützung von Planung und Kontrolle, sodass insbesondere Geschäftsprozesse profitieren konnten. Zukünftige Forschungsaktivitäten sollten die positiven Erfahrungen der SW-Prozessforschung aufgreifen und ganz gezielt auf Projektplanung übertragen, um die Domäne theoretisch und mit Hilfe der Mathematik zu durchdringen. Schwerpunkte sollten die Frage nach Qualitätseigenschaften von Plänen (z.B. Korrektheit, Realisierbarkeit etc.) sowie die Beachtung von Informationssicherheitsaspekten (z.B. zur Konsistenthaltung von Plänen ohne die Pläne zu enthüllen) sein. Wenn die formalen und theoretischen Grundlagen geklärt sind, können darauf aufbauend einerseits Werkzeuge gebaut werden. Andererseits ergeben sich ganz neue Fragestellungen im Kontext von Vorgehensmodellen, beispielsweise wie ein Vorgehensmodell dabei unterstützen kann, stets „gute Pläne“ abzuleiten.

Literaturverzeichnis

[AH07] Alberts, D. S.; Hayes, R. E.: Planning: complex endeavors. Assistant Secretary of Defense (C3I/Command Control Research Program) Washington DC, 2007.

[BC+07] Bendraou, R.; Combemale, B.; Crégut, X.; Gervais, M. P.: Definition of an Executable SPEM 2.0. In: Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific. IEEE, 2007; S. 390-397

[BE+10] Buschermöhle, R.; Eekhoff, H.; Frommhold, H.; Josko, B.; Schiller, M.: SUCCESS 2010. Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland (V. 2.0). 2010.

[BL+95] Bröckers, A.; Lott, C. M.; Verlage, M.; Rombach, H. D.: MVP-L language report version 2. 1995.

[BS01] Broy, M.; Stølen, K.: Specification and development of interactive systems. FOCUS on streams, interfaces, and refinement. New York, NY: Springer (Monographs in computer science). 2001.

[CN+03] Chen, F.; Nunamaker Jr, J. F.; Romano, N. C.; Briggs, R. O.: A collaborative project management architecture. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, IEEE, 2003; S. 12ff.

[DL04] Dvir, D.; Lechler, T.: Plans are nothing, changing plans is everything: the impact of changes on project success. In: Research Policy 33 (1), 2004; S. 1-15

[DS02] Damm, D.; Schindler, M.: Security issues of a knowledge medium for distributed project work. International Journal of Project Management, 2002; 20. Jg., Nr. 1, S. 37-47

[EG91] Emmerich, W.; Gruhn, V.: FUNSOFT nets: a Petri-net based software process modeling language. In: Proceedings of the 6th international workshop on Software specification and design. IEEE Computer Society Press, 1991; S. 175-184

[FB11] Friedrich, J.; Bergner, K.. Formally founded, plan-based enactment of software development processes. Proceedings of the 2011 International Conference on Software and Systems Process. ACM, 2011.

[Fis10] Fischer, E.: Operationalisierung des Projektcontrollings. Verlag Dr. Hut, 2010.

[Gna05] Gnatz, M. A. J.: Vom Vorgehensmodell zum Projektplan. VDM Publishing, 2005.

[KH02] Koskela, L. J.; Howell, G.: The underlying theory of project management is obsolete. In: The PMI Research Conference: PMI, 2002; S. 293-302

[Mau96] Maurer, F.: Working Group Report on Computer Support in Project Coordination. In: WETICE. 1996. S. 200-206

[MD+00] Maurer, F.; Dellen, B.; Bendeck, F.; Goldmann, S.; Holz, H.; Kötting, B.; Schaaf, M.: Merging Project Planning and Web-Enabled Dynamic Workflow Technologies. In: IEEE Internet Computing 4 (3), 2000; S. 65–74

[MH04] Münch, J.; Heidrich, J.: Software project control centers: concepts and approaches. In: Journal of Systems and Software, 2004; 70 (1–2), S. 3-19

[Ost87] Osterweil, L.: Software processes are software too. Proceedings of the 9th international conference on Software Engineering. IEEE Computer Society Press, 1987.

[RB+10] Ramler, R.; Beer, W.; Klammer, C.; Wolfmaier, K.; Larndorfer, S.: Concept, Implementation and Evaluation of a Web-Based Software Cockpit. In: Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on. IEEE, 2010; S. 385–392

[Sta73] Stachowiak, H. Allgemeine Modelltheorie. Springer. 1973.

[WC+11] Wise, A.; Cass, A. G.; Lerner, B. S.; McCall, E. K.; Osterweil, L. J.; Sutton Jr, S. M.: Using Little-JIL to coordinate agents in software engineering. In Engineering of Software. Springer Berlin Heidelberg, 2011; S. 383-397