mehr zum thema:
http://www.dpunkt.de/buecher/3322.html
http://d-nb.info/1000557650

PRODUKTIVE SOFTWAREENTWICKLUNG: ACHT FAKTOREN FÜR MEHR PRODUKTIVITÄT UND QUALITÄT

Die Produktivität steht bis heute nicht im Zentrum der Betrachtung des Software-Engineerings und des Projektmanagements. Das ist die initiale These unseres Buchs „Produktive Softwareentwicklung". Wir fordern darin eine Professionalisierung der Softwareentwicklung durch messbare Produktivitäts- und Qualitätssteigerungen. Der folgende Auszug aus dem Buch befasst sich mit den acht elementaren Faktoren, die nach unserer Erkenntnis wirklich relevant für eine produktive Softwareentwicklung sind.

die autoren

Dr. Hans-Jürgen Plewan
(Hans-Juergen.Plewan@f-i-solutions-plus.de)
beschäftigt sich vor allem mit der Frage, wie sich Projekte besser und zuverlässiger durchführen lassen und wie sie eine noch höhere Qualität bei noch größerem Nutzen liefern können.

Dr. Benjamin Poensgen
(benjamin.poensgen@quantimetrics.de)
ist als Unternehmensberater auf dem Gebiet der Bewertung und Verbesserung von Organisation und Prozessen der betrieblichen IT-Anwendungsentwicklung tätig.

Professionalisierung ist ein Anspruch, dem sich auch die Softwareentwicklung als Ingenieurdisziplin stellen muss. Dabei gehört es zu den Grundanforderungen professioneller Softwareentwicklung, über ein gemeinsames Verständnis für Produktivität und Qualität zu verfügen.

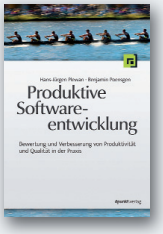
Aus unserer Sicht liegt der Schlüssel zur Professionalisierung in der Verstärkung der Bedeutung der Produktivität. Produktivität ist dabei kein Selbstzweck, sondern eine strategische Voraussetzung für die Beherrschbarkeit der Entwicklung und Weiterentwicklung (großer) Informationssysteme. Wir plädieren mit unserem Buch für eine produktive Softwareentwicklung, der die Zusammenhänge von Projekterfolg, Produktivität und Qualität bewusst sind.

Die Leistungsproduktivität wird in dem Buch allgemein als Verhältnis von Projektergebnis zu Projektaufwand eingeführt. Dabei werden funktionale Größen wie beispielsweise *Function Points* oder *Use Case Points* als Maße für Projektergebnisse empfohlen, sodass die Leistungsproduktivität in *Function Points* oder *Use Case Points* pro Personenmonat angegeben werden kann.

Damit wird Produktivität zu einer messbaren Größe. Entwicklungsprojekte und -einheiten, die ihre Produktivität messen und bewerten können, sind auch in der Lage, diese zu steigern. Dabei besteht naturgemäß ein enger Zusammenhang zu Qualität und Geschwindigkeit.

Das Buch „Produktive Softwareentwicklung" basiert auf jahrelanger praktischer Erfahrung aus komplexen Entwicklungsprojekten und ist aus der Praxis für die Praxis geschrieben. Es gliedert sich in vier Teile. Die ersten beiden Teile befassen sich mit der Messung von Produktivität und Qualität sowie entsprechenden Kennahlen. Im dritten und vierten Teil gehen wir der Frage nach, wie die Produktivität auf Basis eines solchen Koordinatensystems gesteigert werden kann. In diesem Kontext steht das zehnte Kapitel, „Die acht elementaren Produktivitätsfaktoren", das der folgende Auszug wiedergibt.

Hans-Jürgen Plewan und Benjamin Poensgen:
Produktive Softwareentwicklung.
dpunkt.verlag,
September 2011,
262 Seiten,
39,90 Euro (D).



Die acht elementaren Produktivitätsfaktoren

Welches sind die aus Sicht der Praxis wirklich relevanten Produktivitätsfaktoren der Softwareentwicklung? 250 potenzielle oder theoretische Faktoren, wie sie etwa bei Caspers Jones beschrieben werden (vgl. [Jon00]), überfordern jeden Praktiker. Wir wollen den komplexen Raum der Faktoren und Variablen, die die Produktivität beeinflussen können, auf das in der Praxis wesentliche Koordinatensystem reduzieren. Dabei haben wir die Produktivität und vor allem die Steigerung der Produktivität im Fokus. Dafür ist es hilfreich, sich an einem Schema für produktive Prozesse zu orientieren. Dieses Schema oder „Produktivitätsmodell" wollen wir im Folgenden in drei Stufen entwickeln. Damit werfen wir einen ersten Blick auf die wesentlichen Produktivitätsfaktoren und auf die Ansätze zur Produktivitätssteigerung.

Ein einfaches Modell produktiver Prozesse

Im Kern ist ein produktiver Prozess ein Prozess, der eine festgelegte Aufgabe mit einem festgelegten Ziel erledigt und dabei ein Ergebnis produziert. Das Schneiden der Hecke meines Gartens ist ein produktiver Prozess, der in dem Ergebnis einer gekürzten und geschnittenen Hecke resultiert. Abbildung 1 verdeutlicht das einfachste Schema eines produktiven Prozesses.

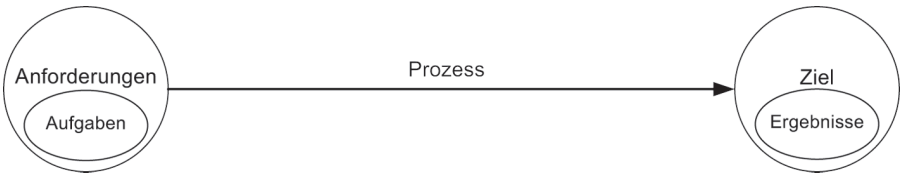


Abb. 1: Einfaches Schema eines produktiven Prozesses.

In diesem Sinne ist jedes Softwareentwicklungsprojekt ein produktiver Prozess. Die Entwicklung eines neuen Internetportals ist ein produktiver Prozess, der eine lauffähige Anwendung, aber auch ein Fachkonzept, viele Testfälle, eine Systemdokumentation sowie einige Zwischenergebnisse wie Status- und Testreports erzeugt.

In dieser Form ist das Modell natürlich noch viel zu einfach, um die Einflüsse auf die Produktivität erkennen zu können. Hierzu müssen wir das Modell um weitere Faktoren erweitern (siehe Abbildung 2). Ein wichtiges Element ist das konkrete Vorgehen. Dieses beinhaltet die Nutzung von Werkzeugen, aber auch andere Handlungsweisen, die die Arbeit erleichtern oder auch erschweren. Am Beispiel der Hecke könnte ich eine alte manuelle Heckenschere benutzen oder auch eine Elektroschere, mit der ich in der Regel viel schneller bin. Ich könnte die Hecke alleine schneiden oder meinen Schwager bitten, mir zu helfen. Ich könnte die Hecke von unten nach oben oder in umgekehrter Richtung schneiden. Ich könnte die Hecke in mehreren Stufen schneiden usw.

Es ist klar, dass das konkrete Vorgehen die (Leistungs-)Produktivität unseres Vorhabens stark beeinflusst. Das gilt natürlich auch für die Softwareentwicklung mit Blick auf die vielen Entwicklungsmethoden, Entwicklungsprozesse und Reifegradmodelle.

Darüber hinaus ist es für den produktiven Prozess entscheidend, welche Menschen oder welches Team die Arbeit des Prozesses verrichtet. Wie groß ist das Team? Welche Erfahrungen und Kompetenzen hat es? Wie ist das Team und sind die einzelnen Mitglieder motiviert? Wie lange und wie gut arbeitet das Team zusammen? Am Beispiel der Hecke stellt sich die Frage, wie körperlich fit ich bin, wie stark meine Oberarme sind usw. Habe ich ein Interesse oder die Motivation, die gesamte Hecke sehr zügig und möglichst innerhalb von zwei Stunden zu schneiden, oder will ich mir eher Zeit lassen und mich nebenbei mit meinem Nachbarn über Fußball unterhalten?

In Softwareentwicklungsprojekten und allgemeinen Prozessen ist aber nicht nur das Team ein entscheidender Produktivitätsfaktor. Wichtig sind auch die Umgebung und das Umfeld, in denen der produktive Prozess durchgeführt wird: also die Stakeholder, die Kommunikation und die Politik im Umfeld des Projekts, die Unternehmenskultur des Hauses, in dem das Projekt durchgeführt wird usw.

Natürlich gibt es Kriterien im Umfeld eines Projekts, die dessen Erfolg und die Produktivität fördern, genauso wie es Faktoren im Projektumfeld gibt, die das Projekt eher behindern, stören und bremsen. In der Analogie des Heckenschneidens ist es ein Unterschied, ob ich die Hecke bei strahlendem Sonnenschein oder Regen und starkem Wind schneide oder ob mich beispielsweise mein Nachbar in ein Gespräch verwickelt. Insofern können das Wetter oder andere Aspekte des Umfelds mein Voranschreiten und meine Produktivität stark beeinflussen.

Ein weiterer wichtiger Faktor unseres produktiven Prozesses ist die explizite Steuerung des Vorhabens, die gewissermaßen alle anderen Variablen im Fokus hat. Steuerung bezieht sich zunächst auf alle Vorbereitungs- und Planungsaktivitäten. Bevor ich am Nachmittag mit dem Heckenschneiden starte, werde ich einen ungefähren Plan entwickelt haben, in welchem Zeitraum ich das schaffen werde. Ich werde diesen Plan während der Durchführung gelegentlich verifizieren und prüfen, ob ich die Hecke bis zum Abend vollständig schneiden kann. Sollte ich das Vorhaben unterschätzt haben und mein Zeitrahmen wird knapp, werde ich entweder einen Zahn zulegen oder meinen Plan überdenken. Das ist vielleicht für das Schneiden einer kleinen Hecke nicht notwendig und überdimensioniert. Wenn ich aber 50 Meter Hecke schneiden muss, werde ich ohne grobe Zeitplanung, Erfahrung (Aufwandsschätzung) und Prüfung des Status (Kontrolle) nicht auskommen. Steuerung ist in diesem Sinne ein expliziter Faktor, der das Gesamtsystem im Blick hat (in Abbildung 2 nicht enthalten).

Damit haben wir ein erstes Modell für produktive Prozesse, das die wichtigsten Produktivitätsfaktoren schematisch und explizit benennt:

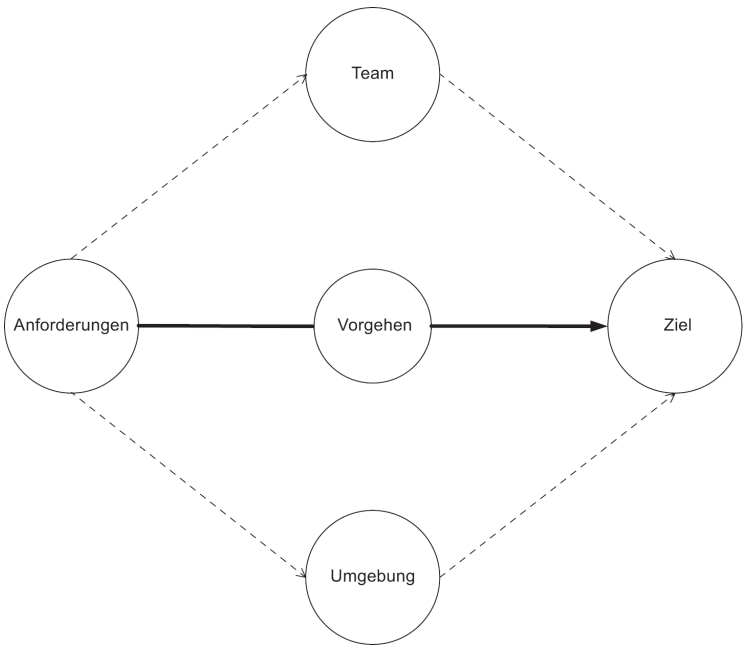


Abb. 2: Einfaches Modell eines produktiven Prozesses.

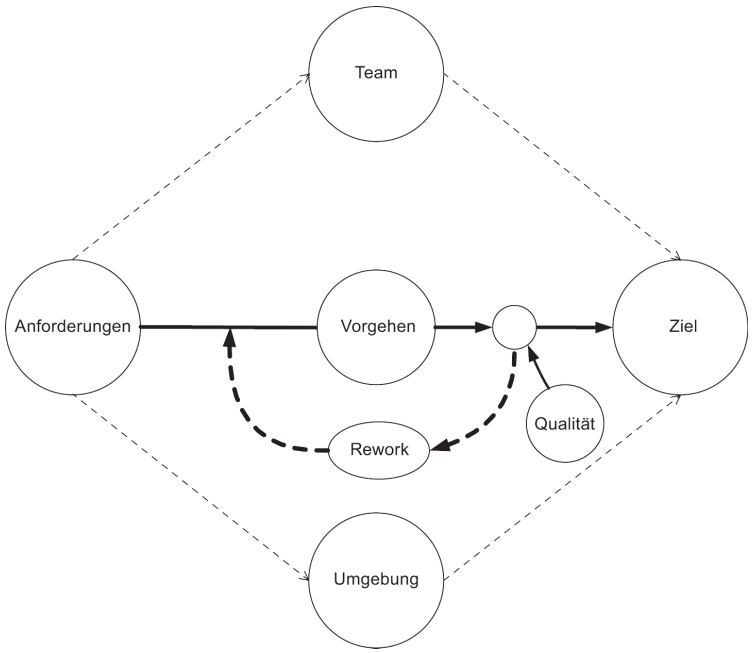


Abb. 3: Erweitertes Modell eines produktiven Prozesses.

- Anforderungen (Aufgabenstellung)
- Ziel (erwartetes Ergebnis)
- Vorgehen
- Team
- Umgebung
- Steuerung

Gerade für die Softwareentwicklung ist es dabei wichtig, diese Faktoren in ihrer Gesamtheit zu sehen. In der Regel spielen die Faktoren bei der Steigerung der Produktivität eines Unternehmens oder eines Entwicklungsteams zusammen. Dietrich Dörner spricht in [Dör94] allgemein von den „kritischen Variablen" eines komplexen, vernetzten und dynamischen Systems: „Kritische Variablen sind die zentralen Variablen eines Systems; beeinflusst man sie, so beeinflusst man in hohem Maße den Zustand des gesamten Systems." In diesem Sinne sind die Faktoren unseres Produktivitätsmodells die kritischen Variablen des Systems der Softwareentwicklung.

An diesem Modell wollen wir im Folgenden festhalten und dieses zunächst um einen wichtigen Faktor erweitern.

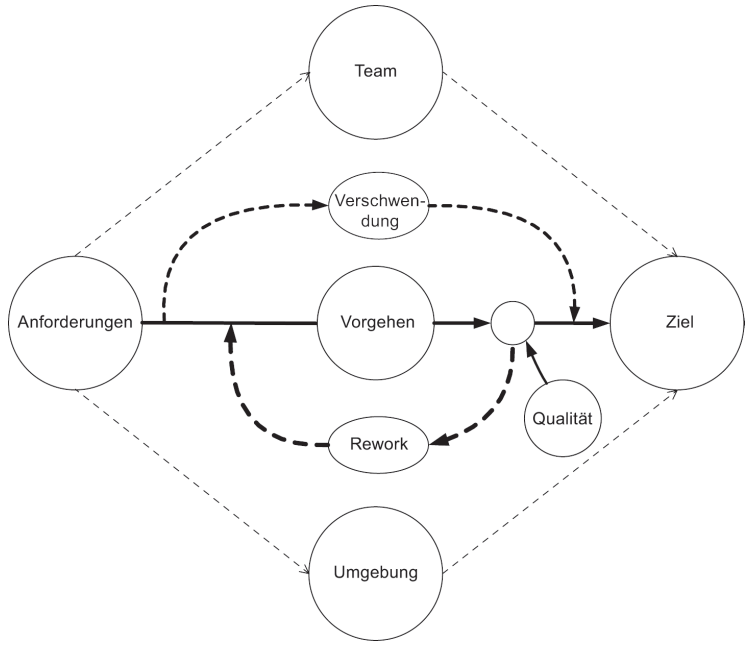


Abb. 4: Vollständiges Modell eines produktiven Prozesses.

Berücksichtigung von Qualität und Rework

In der Softwareentwicklung spielt die Qualität des Ergebnisses eine besonders wichtige Rolle. Unser Modell berücksichtigt dies noch nicht explizit – wir erweitern es deshalb entsprechend, wie in **Abbildung 3** dargestellt. Um in der Analogie des Heckenschneidens zu bleiben, werde ich bestimmte Qualitätsanforderungen an mein Ergebnis stellen. Eine sichtbar schief geschnittene Hecke werde ich nicht akzeptieren. In diesem Fall werde ich nacharbeiten und versuchen, die Hecke durch weiteres und richtiges Schneiden zu begradigen. Erst wenn die Hecke meinen Qualitätsanforderungen entspricht, werde ich mein Projekt beenden.

In einem Softwareentwicklungsprojekt habe ich zum Beispiel Qualitätsanforderungen in Form von durchzuführenden Testfällen. Erst wenn alle 800 Testfälle fehlerfrei durch die Anwendung gelaufen sind, ist die vereinbarte Qualität erreicht. Aber auch Änderungsanforderungen (*Change Requests*) des Fachbereichs, die aus bestimmten Gründen noch im Projekt realisiert werden müssen, sind typische Nacharbeiten in Softwareentwicklungsprojekten.

Natürlich gehören diese Nacharbeiten zu meinem Projekt und beeinflussen meine Produktivität unmittelbar. Je mehr Nacharbeiten notwendig sind, desto mehr Zeit und Aufwand muss ich in das Projekt stecken und desto schlechter wird meine Produktivität. Im Kontext des *Lean Management* und des *Lean Software Development* (vgl. [Pop03]) werden diese Nacharbeiten *Rework* genannt. Diesen englischen Begriff werden wir im Folgenden auch verwenden.

Berücksichtigung von Verschwendungen

Die oben beschriebenen Nacharbeiten lassen sich durch die frühzeitige Entwicklung guter Qualität im Sinne eines „Doing it right the first time" vermeiden. In diesem Sinne kann man *Rework* bereits als Verschwendung (*Waste*) ansehen. In der Begriffswelt des *Lean Management* wird das in der Regel auch so verstanden.

Wir wollen hier *Rework* und *Waste* als zwei unterschiedliche Faktoren ansehen und unser Produktivitätsmodell entsprechend um das Element *Verschwendung* erweitern (siehe Abbildung 4). Um beim Beispiel des Heckenschneidens zu bleiben: Falls ich auf die Idee kommen sollte, vor dem Heckenschneiden ein Konzept darüber auf ein Papier zu malen, wäre das absolute Verschwendung. Das Papier bräuchte ich weder für die Aktivitäten des Heckenschneidens noch für die Zeit danach. Sollte ich aus meiner Hecke allerdings ein grünes Kunstwerk im Stile einer englischen Parklandschaft machen wollen, könnte es schon sinnvoll sein, das vorher auf einem Papier zu entwerfen.

Ein Beispiel für eine typische Verschwendung wäre es auch, wenn ich die Hecke radikal auf einen Meter kürzen würde, obwohl ich mir nur 1,5 Meter vorgenommen habe oder 1,5 Meter ausreichen würden. In diesem Fall hätte ich die Anforderung übererfüllt, was aus Sicht der Produktivität reine Verschwendung ist, weil ich mich zwei Stunden länger abgemüht habe, als notwendig gewesen wäre, um das Ziel zu erreichen.

Verschwendungen sind in der Softwareentwicklung und in Softwareprojekten ein häufig zu beobachtendes Phänomen: Wertlose Aktivitäten und Zwischenergebnisse schaffen keinen wirklichen Beitrag zum Projektziel und zur Wertschöpfung des Projekts. Tabelle 1 gibt eine Übersicht über die wichtigsten Verschwendungsarten in der Softwareentwicklung, erläutert anhand der Begriffe aus dem *Lean Software Development*.

Arten von Verschwendung	Beispiele
Überproduktion	Unbenötigte Funktionalität, redundante Funktionalität
Übertriebene Prozesse	Überflüssige Dokumente, überflüssige Meetings
Rework	Anwendungsfehler, geänderte Anforderungen, geändertes Design
Unnötige Bewegung (*Motion*)	Ungeplanter Aufgabenwechsel, Austausch eines Teammitglieds, An-/Abreisen
Wartezeiten	Fehlende Zulieferung, langsame Infrastruktur (Netz), Warten auf einen Ansprechpartner
Skills	Über-/Unterforderung, fehlendes fachliches Wissen, wenig Methodenerfahrung

Tabelle 1: Verschwendungsarten in der Softwareentwicklung.

Die acht Gebote der Produktivität

Wie kann man eine potenzielle Beschleunigung in der Softwareentwicklung erreichen? Wie kann man die gleichen Ergebnisse mit einer guten Qualität, aber mit weniger Aufwand schneller erreichen? Wir wollen dazu das vorgestellte Produktivitätsmodell und dessen Produktivitätsfaktoren nutzen. Auf dieser Basis können die Richtungen der Produktivitätsverbesserung erklärt und dargestellt werden. Alle acht Faktoren des Produktivitätsmodells bieten dafür Ansätze:

1. Ziel
2. Team
3. Anforderungen
4. Vorgehen
5. Qualität und Rework
6. Verschwendung
7. Umgebung
8. Steuerung

Für jeden dieser Faktoren gibt es in der Praxis ein bis zwei wesentliche Kriterien, die die Produktivität steigern. Wir sprechen von „Richtungen der Beschleunigung" als Produktivitätsfaktoren. Zusätzlich gibt es für jeden der Faktoren eine Handvoll bewährter Praktiken, mit denen man die Produktivitätspotenziale heben kann.

Dabei steht jeder Produktivitätsfaktor unter einem primären *Motto*. Für uns sind das die „Acht Gebote produktiver Softwareentwicklung". Jeder der acht elementaren Produktivitätsfaktoren bietet das Potenzial, die Produktivität zu steigern, und für jeden Produktivitätsfaktor gilt ein spezifisches Gebot der Beschleunigung, das zum Schluss noch kurz erläutert wird (siehe Kasten 1)[1]).

Jeder der acht elementaren Produktivitätsfaktoren (Ziel, Team, Anforderungen, Vorgehen, Qualität, Verschwendung, Umgebung und Steuerung) bietet das Potenzial, die Produktivität zu steigern. Für jeden Produktivitätsfaktor gilt ein spezifisches Gebot der Produktivität:

1. Die Macht der Ziele nutzen.
2. Produktive Hochleistungsteams aufbauen.
3. Den Kern der richtigen Anforderungen treffen.
4. Vorgehen ohne effektive Methodik abstellen.
5. Qualität steigern und Rework radikal reduzieren.
6. Verschwendungen erkennen und eliminieren.
7. Projekte richtig in die Umgebung integrieren.
8. Fortschritt, Qualität und Produktivität steuern.

Kasten 1: Die acht Gebote produktiver Softwareentwicklung.

Bedeutung der Produktivitätsfaktoren im Überblick

Projekte, in denen die genannten acht Gebote beachtet werden, sind auf dem besten Weg zu einer deutlich höheren Produktivität. Was das für die Praxis bedeutet, wird im Folgenden skizziert.

Die Macht der Ziele nutzen

Natürlich benötigen Softwareprojekte klare Ziele, an denen man ihren Erfolg messen und die Vorgehensweise ausrichten kann. Das klingt wie eine Binsenweisheit. Trotzdem ist es immer wieder erschreckend, wie ziellos manche Projekte durchgeführt werden. Produktivität erreicht man nur über

[1]) *An dieser Stelle endet der Auszug aus dem Buch. Der gesamte darauf folgende Teil IV befasst sich intensiv mit den einzelnen Produktivitätsfaktoren und produktiven Praktiken bzw., damit wie man diese nutzen kann. Welche Bedeutung diese jeweils haben und worum es uns dabei geht, ist im folgenden Abschnitt zusammengefasst.*

Zielorientierung und Zielstrebigkeit. Eindeutig formulierte, transparente Ziele, erreichbare Zwischenziele und eine klare Selbstverpflichtung aller Beteiligten sind dabei von größter Bedeutung für die Projektarbeit.

Der berühmte Zug zum Ziel entsteht dann, wenn ein fester Steuerungsrhythmus und auch die Zielorientierung im Kleinen etabliert werden. Produktive Teams zeichnen sich durch Zielorientierung in allen täglichen Aktivitäten und Schritten des Projekts aus. Das gilt für jedes Projektgespräch, jeden Workshop, jedes Meeting und jede übernommene Aufgabe.

Produktive Hochleistungsteams aufbauen

Das Team ist der größte Produktivitätshebel jedes Projekts. Ausgeprägte Fähigkeiten in Form von *Hard Skills* und *Soft Skills*, eine hohe Motivation und die Etablierung einer Leistungskultur sind wirksame Faktoren. Wesentliche Erfolgsfaktoren sind dabei einerseits die Organisation des Teams inklusive einer angemessenen Teamgröße, der richtigen Besetzung des Teams (Skill-Mix) und andererseits alle wesentlichen Praktiken, die die Fähigkeiten und die Leistungsbereitschaft des Teams erhöhen. Denn das sind die beiden wesentlichen Richtungen der Beschleunigung für diesen Produktivitätsfaktor.

Den Kern der richtigen Anforderungen treffen

Die Kenntnis der richtigen Anforderungen ist die Achillesferse der meisten Projekte. Wenn Projekte in Schieflage geraten oder sehr unproduktiv sind, liegt die Ursache oft in der Anforderungsanalyse. Mehrarbeit in späteren Projektphasen durch falsche, fehlende oder missverstandene Anforderungen sind klassische Produktivitätsthemen. Aus unserer Erfahrung können Anforderungsfehler 30 % bis 50 % aller Fehler in einem Entwicklungsprojekt ausmachen, die schlimmstenfalls erst im System- oder Abnahmetest oder sogar erst im laufenden Betrieb entdeckt werden. Entscheidend ist es, die wirklich wesentlichen Anforderungen zu finden. Eine Steigerung der Produktivität wirkt daher bei den Anforderungen vor allem in zwei Richtungen: in Richtung zu den wesentlichen Anforderungen und in Richtung zu präzisen und verständlichen Anforderungen.

Vorgehen ohne effektive Methodik abstellen

Der effiziente Weg zum Ziel wird vor allem durch das richtige Vorgehen eines Projekts bestimmt. Im Sinne des Software-Engineerings geht es dabei vor allem um die richtige Methodik. Beschleunigung durch das Vorgehen erreicht man vor allem in zwei Richtungen: durch eine funktionierende und etablierte Methodik sowie durch die Vermeidung eines strikten Wasserfalls beziehungsweise „Big Bangs".

Dabei ist der wichtigste Produktivitätshebel für das Vorgehen, insgesamt eine systematische Entwicklungsmethodik zu haben und diese konsequent umzusetzen. Es gilt die Faustregel, dass die konsequente Einführung einer umfassenden Entwicklungsmethodik in einer unreifen Organisation eine um den Faktor zwei bis drei höhere Produktivität erreichen kann.

Qualität steigern und Rework radikal reduzieren

Qualität schafft Produktivität durch weniger Nacharbeiten beziehungsweise *Rework*. Die Richtungen der Beschleunigung gehen für den Produktivitätsfaktor Qualität vor allem in Richtung eines messbaren Qualitätsbegriffs und in Richtung einer frühzeitigen, permanenten Qualitätssicherung. Eine gute Qualitätssicherung entdeckt Fehler und Qualitätsmängel früh.

So kann man zum Beispiel feststellen, dass es im allgemeinen praktikabler ist, Code-Reviews und Tests durchzuführen, als sich nur auf Tests zu verlassen. Auch wenn es oft nicht geglaubt wird, Code-Reviews sind eine effizientere Methode, um Fehler zu finden, als Tests.

Verschwendungen erkennen und eliminieren

Verschwendungen in Softwareprojekten zu erkennen und zu eliminieren gehört bereits zur hohen Kunst der Produktivitätssteigerung. Dabei kann viel Verschwendung schon durch Zielorientierung vermieden werden, wenn Zielorientierung auch den kürzesten und angemessensten Weg zum Ziel bedeutet. Manche Verschwendungen, wie z. B. unproduktive Meetings, sind für einen Außenstehenden meist schnell ersichtlich, während andere nicht auf den ersten Blick erkennbar sind. Dabei zählen wir die Nacharbeiten wegen zu spät erkannter schlechter Qualität (*Rework*) nicht zur Verschwendung, obwohl auch das mit Blick auf Aufwände, Kosten und Zeit reale Verschwendungen erster Güte sind.

Allen produktiven Tätigkeiten, die die Produktivität durch Eindämmung von Verschwendung steigern können, geht eine grundsätzliche Haltung voraus, die wir in unseren Projekten „Angemessenheit" nennen. Angemessenheit wird zum Handlungsprinzip bei allen Entscheidungen. In einer solchen Kultur sind alle Methoden, Prozesse und Werkzeuge des Software-Engineerings „nur" Hilfsmittel, um das Projektziel möglichst einfach, schlank und direkt zu erreichen.

Projekte richtig in die Umgebung integrieren

Selbst das beste Projekt kann durch ein schwieriges Projektumfeld gebremst oder im Extremfall sogar gestoppt werden. Die Umgebung eines Projekts bezieht sich dabei vor allem auf dessen organisatorisches und politisches Umfeld, auf den Auftraggeber und die verschiedenen Stakeholder und Interessenvertreter. Wichtig für ein Projekt und dessen Umgebung ist es, wie es organisatorisch in die Umgebung integriert ist. Dabei ist es notwendig, das Projekt als organisatorische Kapsel zu betrachten und vor dem „Draußen" angemessen zu schützen. Nach außen sollte es möglichst nur definierte Schnittstellen und kontrollierte Kommunikationswege geben. Ist das nicht der Fall und liegt das Projekt offen und ungeschützt in der Umgebung, wird die Umgebung versuchen, Einfluss auf das Projekt und dessen Ziele zu nehmen.

Fortschritt, Qualität und Produktivität steuern

Im weitesten Sinne haben fast alle Produktivitätshebel, die wir hier beschrieben haben, mit der Steuerung eines Projekts zu tun. So ist es Aufgabe der Projektsteuerung, Zielorientierung in ein Projekt zu bringen, das Team richtig zu besetzen und zu organisieren, eine passende Entwicklungsmethode auszuwählen oder auch die Qualitätssicherung richtig zu planen. Projektsteuerung *im engeren Sinne* bezieht sich dabei auf die *direkte* Steuerung der wesentlichen Ecken des Projektmanagement-Quadrats: Aufwand, Zeit, Qualität und Produktivität.

Projektsteuerung beginnt bereits vor dem Start eines Projekts. Dort werden auch die Grundlagen für seine Produktivität gelegt. Einem Projekt realistische Rahmenbedingungen in Form von Budget und Termin-

planung zu geben, ist eine Voraussetzung für ein erfolgreiches Projekt. Eine Schätzung des Aufwands und eine Planung, die nicht erfahrungsbasiert und nicht methodisch erfolgt, ist immer ein Glücksspiel. Bauchschätzungen oder politisch vorgegebene Aufwände sind riskant, im Zweifel führen sie zu einem unrealistischen oder kranken Projekt – ganz anders in Organisationen, in denen bewährte Methoden zur Aufwandsschätzung eingesetzt werden und die ihre Produktivität kennen. Aus Sicht der Produktivitätssteuerung ist daher eine wesentliche Richtung der Beschleunigung die hin zu einer methodischen Aufwandsschätzung und Planung.

Die zweite Beschleunigungsrichtung ist die Richtung zum Einsatz praktikabler Projektmanagement-Metriken während der Projektdurchführung. Für die Projektsteuerung ist es wichtig, dies nicht dem eigenen Gefühl zu überlassen, sondern den Fortschritt, die Qualität und die Produktivität während der Projektdurchführung konkret zu messen und zu bewerten.

Fazit und Ausblick

Abschließend stellt sich die Frage: Womit und wie anfangen? Natürlich mit der konsequenten Messung und Bewertung von Produktivität und Qualität der eigenen Projekte. Darüber Erfahrungen zu sammeln und zu wissen, wo man steht, ist die Voraussetzung für alle weiteren Maßnahmen. Erst dann ist man wirklich in der Lage, eine professionelle Softwareentwicklung zu betreiben, und erst dann kann man damit beginnen, die Fragen nach den wirksamsten Produktivitätshebeln und den wichtigsten Maßnahmen zur Verbesserung der Produktivität und Qualität im eigenen Unternehmen zu beantworten. ■

Literatur

[Dör94] D. Dörner, Die Logik des Mißlingens: Strategisches Denken in komplexen Situationen, Rowolth Verlag 1994

[Jon00] C. Jones, Software Assessments, Benchmarks and Best Practices, Addison-Wesley 2000

[Jon08] C. Jones, Applied Software Management, McGraw-Hill 2008

[Pop03] M. & T. Poppendieck, Lean Software Development: An Agile Toolkit, Addison-Wesley 2003

[Wag08] S. Wagner, M. Ruhe, A structured Review of Productivity Factors in Software Development, Institut für Informatik der TU München, Bericht TUM-I0832, September 2008