SCHATTENBOXEN: WARUM METRIKEN NICHT FUNKTIONIEREN KÖNNEN – UND ES TROTZDEM TUN

der autor

Jens Coldewey
(E-Mail: jens.coldewey@it-agile.de)
ist Geschäftsführer der it-agile GmbH und Chefredakteur von OBJEKTspektrum.

Seit mehreren Dekaden sind Metriken Kennzeichen „wissenschaftlich fundierten Managements" – sowohl im Projektmanagement als auch in der Unternehmensführung. Unfälle beim Einsatz von Metriken werden den Anwendern zugeschrieben, obwohl es längst kritische Stimmen gibt, die fundamentale Probleme anführen. Diese Kritiker legen überzeugend dar, dass Metriken nicht funktionieren können. Dennoch kennt jeder Profi Praxisberichte über den erfolgreichen Einsatz von Metriken. Dieser Artikel versucht eine Erklärung hierfür zu finden.

„If you can't measure it you can't manage it", ist ein Zitat, das so ziemlich jedem Management-Guru der letzten Jahrzehnte zugesprochen wurde, von Peter Drucker über Robert Kaplan und Edwards Deming, bis hin zu Tom DeMarco. Im Jahr 2000 ging die Firma Kinex AHA sogar so weit, den zitierten Satz in den USA als Warenzeichen eintragen zu lassen. Ganze Lehrstühle, Arbeitskreise und Unternehmen haben sich dem Versuch verschrieben, Projekte und Organisationen durch Messungen, „Key Performance Indicators" und „Total Quality Management" zu verbessern. Zahlen sind gut, denn sie geben weichen und unhandlichen Gebilden wie Organisationen etwas Objektives, etwas Wissenschaftliches.

Erste Zweifel an der Praxistauglichkeit des flotten Spruchs kamen bei mir auf, als mich der Abteilungsleiter eines Kunden bat, ihm jede Woche den Zeilenzuwachs unseres Smalltalk-Projekts zu übermitteln: „Ich habe Zahlen gefunden, mit denen man aus der Anzahl der Codezeilen die *Function Points* ermitteln kann, und würde gerne nachweisen, dass Smalltalk produktiver ist als COBOL". Obwohl sein Ziel meine herzlichste Sympathie genoss, musste ich ihn leider enttäuschen: Wir hatten in der letzten Woche leider ein größeres Stück Code refaktorisiert und eine Produktivität von Minus 8.000 Zeilen Code pro Woche erreicht – vermutlich „schlimmer", als das gefürchtetste COBOL-Projekt im Konzern jemals war. Wir mussten nie wieder über die Anzahl von Codezeilen berichten.

Es ist einfach, weitere Beispiele für versagende Metriken zu finden:

- Messungen der Testüberdeckung, die zu sinnlosen und schwer wartbaren Tests führen.
- Messung der *Velocity*, die zu schlechterer Qualität und pathologischem Schätzverhalten führt.
- Prämien für behobene Fehler, die zur Explosion der Fehleranzahl führen.

Diese Phänomene sind auch außerhalb der IT bekannt, zum Beispiel bei Steuern: So besteuerten beispielsweise einige österreichische Städte Häuser nach der Breite ihrer Front, was zu kaum mehr bewohnbaren Scheibenhäusern führte. In manchen Gegenden Griechenlands wird die Grundsteuer bei der Fertigstellung des Gebäudes erhoben – was ganze Ortschaften mit unverputzten Gebäuden zur Folge hat. Und die Bezahlung von Fondsmanagern nach der Quartalswertsteigerung ihres Fonds hat wesentlich dazu beigetragen, uns den schlimmsten Einbruch der Weltwirtschaft seit dem zweiten Weltkrieg zu bescheren

Allen diesen Beispielen ist gemeinsam, dass sie ursprünglich gut gemeint waren, sinnvoll aussahen und doch das Gegenteil dessen bewirkt haben, was beabsichtigt war. Ist das alles nur Zufall? Oder liegt es am Unvermögen der Beteiligten?

Nein, es handelt sich um ein inhärentes Problem metrikbasierter Ansätze, behauptet zumindest der Harvard-Professor Robert Austin. In seinem Buch „Measuring and Managing Performance in Organisations" (vgl. [Aus98]) entwickelt er die Behauptung, solche Fehlentwicklungen seien keine bedauerlichen Ausnahmen einer ansonsten guten Idee, sondern seien bereits in der Idee des metrikbasierten Managements angelegt. Schlimmer noch: Sie sind ihre normale Konsequenz.

Aber auch wenn man Austins Argumentation folgt, kann man meiner Ansicht nach Metriken gewinnbringend einsetzen, vorausgesetzt, man ist sich ihrer Schadmechanismen und Gefahren bewusst und setzt sie entsprechend ein. Doch bevor ich darauf im Detail eingehe, zunächst eine Kurzfassung der Argumentation Austins.

Warum Metriken nicht funktionieren

Der Einfachheit halber möchte ich mich auf die Diskussion von Projekten beschränken, weil sie deutlich übersichtlicher sind als ganze Organisationen. Um das Problem von Metriken zu verstehen, muss man ein Projekt als komplexes System interpretieren. In einem solchen System gibt es verschiedene Personen, die miteinander interagieren, um gemeinsam ein Ergebnis zu erzielen, z. B. eine bestimmte Funktionalität.

Konzentrieren wir uns in diesem System einmal auf die Produktivität eines Entwicklers – das ist eine betriebswirtschaftlich durchaus interessante Größe. In der Theorie kann man diese Produktivität als kurz-, mittel- und langfristigen Mehrwert über die gesamte Projektlaufzeit verstehen, den der Programmierer zum Beispiel innerhalb eines Tages schafft oder auch vernichtet: Zum Beispiel könnte er an einem Tag eine bestimmte Funktionalität einbauen, die ab dem nächsten Release pro Tag 100 Euro zusätzlichen Gewinn einbringt, bis sie in drei Jahren durch eine noch bessere Funktion ersetzt wird. Zudem hat er einen Fehler eingebaut, der sich in einem Jahr erstmals bemerkbar macht und der inklusive Behebung einen Gesamtschaden von 20.000 Euro verursacht.

Betriebswirtschaftler können aus diesen Angaben und einigen Annahmen über die Zinsentwicklung einen Gesamtgewinn berechnen, der aufgrund von Zinsfaktoren

zwischen 70.000 und 80.000 Euro liegen dürfte. Dass nur ein Bruchteil dieser Angaben zum Zeitpunkt der Programmierung bekannt ist, soll uns erst einmal nicht stören – wir sind ja noch in der Theorie. Aber verfolgen wir weiter Austins Argumentation.

Die Produktivität des Programmierers hängt von verschiedensten Faktoren ab. Zum einen gibt es Basisfaktoren, wie z. B. die Qualifikation des Programmierers, die Qualität und Stabilität der Entwicklungsumgebung, die Komplexität des existierenden Codes und das gewählte Vorgehen. Darüber hinaus gibt es tagesabhängige Faktoren, die der Programmierer in der Regel nicht zielgerichtet beeinflussen kann oder will: seine psychische Tagesverfassung, die Schlafmenge der letzten Tage, die aktuelle Müdigkeit, die Komplexität des Features oder die Genauigkeit, mit der er den Geschäftsnutzen dieses Feature versteht. Außerdem gibt es einige Parameter, die der Programmierer in Grenzen bewusst beeinflussen kann, wie z. B. die Komplexität der Lösung, den Umfang der Testabdeckung, die Qualität des Designs und nicht zuletzt seinen Zucker- und Koffein-Spiegel im Blut.

In dieser Auflistung fehlen noch einige hundert andere Faktoren, von denen die meisten weder bekannt noch quantitativ erfassbar sind. Interessanterweise beeinflussen einige dieser Faktoren die kurzfristige Produktivität anders als die langfristige. Treibt der Entwickler beispielsweise weniger Aufwand für die Designqualität, kann er die aktuelle Aufgabe möglicherweise schneller „abschließen", senkt aber seine langfristige Produktivität, weil der Code schlechter wartbar wird. Anders gesagt, hat er die Aufgabe nicht wirklich abgeschlossen, sondern technische Schulden hinterlassen, die aktuell kaum zu sehen sind, später aber möglicherweise immense Kosten verursachen werden. Auf diesem Prinzip wird ein großer Teil der zum Festpreis vergebenen Entwicklungsprojekte an das bezahlte Budget angepasst: Die Produktivität des Lieferanten wird erhöht auf Kosten der Wartungsproduktivität des Dienstleisters. Man kauft sozusagen Software mit verdeckten Schulden.

Was passiert nun, wenn wir ein Metrikprogramm einführen, um die Produktivität zu verbessern? Die Idee ist einfach: Ich messe die Produktivität des Entwicklers und motiviere ihn, diesen Wert zu verbessern. Die Motivation kann positiv sein, z. B. durch ein Bonussystem, sie kann negativ sein, wenn einmal im Jahr die beiden Entwickler mit der geringsten Produktivität gefeuert werden; sie kann prominent sein, wie bei millionenschweren Zahlungen, oder sehr subtil durch das simple Wissen, dass die Produktivität gemessen wird, und durch die *Vermutung*, dass diese Werte einen Einfluss auf die eigene Beurteilung haben. Unabhängig von den gewählten Mechanismen wirkt also die Messung der Produktivität zurück auf den Entwickler: Er wird sein Verhalten so ändern, dass sich die gemessenen Werte in die gewünschte Richtung verändern. Wir haben also eine Feedback-Schleife eingebaut, die bewirkt, dass die gemessene Produktivität verbessert wird.

Alles in Ordnung, oder? Wo liegt das Problem? Dafür muss man sich genauer ansehen, wie die Produktivität gemessen wird, und man muss die Dimension „Zeit" berücksichtigen. Damit eine Metrik überhaupt das Verhalten des Entwicklers beeinflussen kann, muss sie zeitnah zur Entwicklung gemessen werden. Drei Jahre zu warten, bis man zumindest in der Theorie die Endrechnung für das Feature aufmachen könnte, führt dazu, dass die Feedback-Schleife nicht mehr auf die aktuellen Tätigkeiten wirkt – einmal ganz abgesehen vom Verwaltungsaufwand, der mit einer solchen Verfolgung aller Programmierleistungen verbunden wäre. Die Lösung: Man ignoriert langfristige Effekte und definiert die Produktivität zum Beispiel als „Story Points pro Tag", die der Entwickler eincheckt, oder als „Function Points" – oder was immer Sie als Schätzgröße einsetzen. Mit anderen Worten: Man misst nur einen Teilaspekt der Produktivität, eine „Projektion", die einige Aspekte erfasst, andere aber ignoriert.

Für Austins Argumentation ist es wichtig, dass diese Projektion kein Unfall ist, sondern der Kernpunkt der Metrik: Eine extrem komplexe Größe wie die Produktivität wird auf einen einfachen, eindimensionalen Wert reduziert, indem man Informationen eliminiert. Genau diese eliminierten Informationen sind das Problem, wie man an unserem Beispiel gut sehen kann: Hier besteht die reale Produktivität zum einen aus einer kurzfristigen Komponente, der bereit gestellten Funktionalität, die von der Metrik abgedeckt wird. Man kann allerdings selbst darüber streiten, ob man diese Funktionalität überhaupt sinnvoll messen kann. Aber unterstellen wir einmal, das wäre möglich (auch wenn ich persönlich das nicht glaube). Zum anderen besteht die reale Produktivität aus einer langfristigen Komponente, die von der Metrik sicherlich nicht abgedeckt wird. Der betriebswirtschaftlich erstrebenswerte Zustand ist eine ausgewogene Balance beider Komponenten. Durch die Feedback-Schleife wird der Fokus aber auf die gemessene Komponente gelenkt: Die Entwickler werden motiviert, die kurzfristige Produktivität zu erhöhen. Dadurch entsteht ein Ungleichgewicht zwischen der messbaren Komponente und der nicht messbaren Komponente, die Optimierung fokussiert sich auf die messbare Komponente. Austin spricht in diesem Zusammenhang von „metrischer Verzerrung" (*Measurement Distortion*). Die Gefahr ist groß, dass die messbare Komponente *auf Kosten* der nicht messbaren optimiert wird. Noch schlimmer: Ob das Risiko eintritt, kann die verantwortliche Managerin nicht einmal feststellen, weil dieses vom Metriksystem nicht erfasst wird. Dadurch, dass der gemessene Wert über die Rückkopplungsschleife wieder in das System „eingespeist" wird, wird seine Bedeutung immer weiter verstärkt. In Anlehnung an den flotten Beraterspruch zu Beginn des Artikels könnte man sagen, „what you measure *is* what you manage", und daraus folgern, dass metrikbasiertes Management ein kastriertes Management ist, das nur Teilaspekte des Problems erfasst und den Rest ignoriert (siehe Abb. 1).

Mit Hilfe von Metriken seine Ziele zu erreichen, ist wie der Versuch, seinen Gegner im Schattenboxen KO zu schlagen: Ist man zufällig gleich weit entfernt von der Leinwand, kann der Versuch gelingen – aber das wäre eben Zufall. Ein im Schattenriss perfekt gesetzter Schlag liegt in der Realität meistens weit daneben.

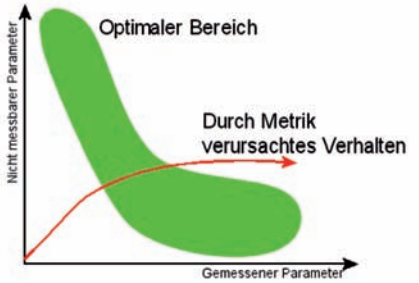

Abb. 1: *Eine Metrik ist immer eine Projektion. Können nicht alle relevanten Parameter gemessen werden, liegt das durch die Metrik erzeugte Verhalten möglicherweise weit außerhalb des wünschenswerten Bereichs.*

Es ist wichtig, dass diese Verzerrung *immer* auftritt, wenn die Metrik die relevanten Parameter *nicht vollständig* erfasst. So bestimmen der Geschwindigkeitsvektor relativ zur Luft und die aktuelle Position eines Flugzeugs seinen Kurs vollständig. Beide können in allen drei Dimensionen mit hoher Genauigkeit erfasst werden. Fällt jedoch die Messung in einer Dimensionen aus, zum Beispiel weil der Geschwindigkeitsmesser vereist ist, befindet sich der Pilot in großen Schwierigkeiten. Ein Projekt ist aber kein deterministisches System, wie ein Flugzeug, sondern eben ein komplexes System, das sich nicht vorausbestimmen lässt[1]).

Im einführenden Beispiel der „negativen Produktivität" war die fehlende Dimension die „Funktionalitätsdichte" des Codes, die wir durch den Umbau erhöht hatten. Bei der Testüberdeckung werden die fachliche Qualität der Tests und die Designqualität des Testcodes ausgeblendet, bei der missbrauchten *Velocity* wurde übersehen, dass diese neben der Produktivität auch noch von Schätzverhalten und Designqualität abhängt; bei der Fehlermetrik konzentrierte man sich auf die behobenen Fehler und nicht auf die eingebauten. In allen diesen Fällen ist die fehlende Dimension nur sehr schwer oder gar nicht quantifizierbar.

Ist es nicht besser, wenn man wenigstens ein paar Werte misst, als dass man sich im völligen Blindflug bewegt? Das mag sein, meint Austin, oder auch nicht. Der vermeintliche Blindflug ist nämlich gar nicht so blind, wie er scheint: Der Mensch ist sehr gut darin, auch extrem komplexe Informationen zu bewerten und ein Experte kann die aktuelle Situation häufig sehr gut „aus dem Bauch heraus" einschätzen. Allerdings nur, wenn er nicht durch äußere Anreize, wie z. B. ein Bonussystem, in eine bestimmte Richtung manipuliert wird. Niemand kann also in einer gegebenen Situation sagen, ob das Bauchgefühl die bessere Information liefert oder die Messungen inklusive der metrischen Verzerrung. Die metrische Verzerrung kann aber zur völligen Dysfunktionalität eines Systems führen – ein Risiko, das einer informellen Bewertung der aktuellen Situation nicht innewohnt. Es entsteht die scheinbar paradoxe Situation, dass die Führbarkeit des Projekts durch die Einführung der Metriken verschlechtert werden kann (nicht muss!) und dass sein Risiko erhöht wird – und nicht umgekehrt. Und genau in dieser wichtigen Frage bewegt man sich in völligem Blindflug. „Man verstand schon vor langer Zeit, dass Metriken möglicherweise auch mal falsche Impulse setzen können", schreibt Tom DeMarco in seinem Vorwort zu Austins Buch, aber dieses Buch erklärt überzeugend, dass „dies nicht die Ausnahme von der Regel ist, sondern die Regel: Alles was man misst, wird mit hoher Wahrscheinlichkeit wenigstens einige falsche Impulse setzen" (vgl. [Aus96], Seite xiii]). Soweit Robert Austin.

[1]) *Das Verhalten eines Flugzeugs in stabilen Strömungsverhältnissen folgt im Wesentlichen den Gesetzen der Aerodynamik, die gut verstanden und in hinreichender Näherung deterministisch sind – auch wenn ein modernes Linienflugzeug zugegebenermaßen ziemlich kompliziert ist.*

Warum Metriken manchmal doch funktionieren

„Es gibt aber doch Metriken, die seit Jahren sehr erfolgreich eingesetzt werden", wenden Sie jetzt vielleicht ein. Ein Beispiel für eine solche Metrik ist die Anzahl der scheiternden Tests auf dem Build-Server, die nach guten XP-Praktiken stets „0" betragen sollte. „Ist das etwa keine Metrik?"

Doch, es ist eine Metrik – und jeder agile Coach kennt auch die zugehörige metrische Verzerrung: Scheiternde Tests werden mal eben auskommentiert, um „den Balken wieder grün zu bekommen". Versucht man, dem mit einer weiteren Metrik zu begegnen (die Anzahl auskommentierter Tests zu messen), werden diese eben gelöscht und die daraus folgende Metrik (die durch Versionsvergleich ermittelte Anzahl der gelöschten Tests) kann problemlos durch eine Änderung des Tests umgangen werden. All dies sind Beispiele, wie man die gemessene Größe auf Kosten der nicht gemessenen Größe optimiert, bis man an einen Punkt kommt, der sich der Messung entzieht: metrische Verzerrung eben.

Dass diese Metrik dennoch in den meisten Teams gut funktioniert, liegt daran, dass sie in allen gängigen agilen Verfahren in ein soziales System eingebunden ist, das der Verzerrung entgegenwirkt. Bleiben wir beim XP als Beispiel:

- Ein roter Balken ist das Problem des gesamten Teams. Niemand außerhalb des Teams kontrolliert die Metrik.
- Das Team ist durch den Kunden vor Ort auf das eigentliche Ziel fokussiert, nämlich qualitativ gute Software auszuliefern.
- Durch die kurzen Iterationen erfahren die Teammitglieder die Konsequenzen getürkter Tests innerhalb weniger Tage. Manipulierte Tests lösen also ihr eigentliches Problem nicht.
- Bei Pair-Programming ist die Wahrscheinlichkeit, dass sich ein Pärchen zum „Sündenfall" verleiten lässt, deutlich geringer, als dass ein „einsamer Entwickler" die Disziplin aufgibt.
- In Retrospektiven wird ein Gruppendruck aufgebaut, die gemeinsamen Regeln einzuhalten.
- Die Regeln „gehören" dem Team, das Team steht also hinter den Zielen, die mit der Metrik erreicht werden sollen.

Alle diese begleitenden Maßnahmen können dennoch nicht mit Sicherheit verhindern, dass ein genervtes Entwicklerpärchen gelegentlich mal den „Pfad der Tugend" verlässt und Tests auch „mal türkt". Die Maßnahmen können lediglich verhindern, dass dies sooft passiert, dass dadurch das ganze System ad absurdum geführt wird.

Für einen Manager ist es allerdings extrem einfach, das System auszuhebeln: Führen Sie eine Strichliste, wer wie oft an einem roten Balken „schuld" war, und geben Sie nur den „besten 30 %" am Jahresende einen Bonus von 10.000 Euro. Innerhalb weniger Monate werden Sie Ihre Testbasis zerstört haben und steigende Fehlerraten in der Produktion zu verzeichnen haben.

Rein informative Metriken, die sich das Team selbst gibt und die von selbst organisierten Kontrollmechanismen wie Retrospektiven flankiert werden, können also funktionieren, wenn man sich darüber im Klaren ist, welche Fehlmechanismen sie bewirken können, und wenn man mit nicht-metrischen Maßnahmen dagegen arbeitet. Sie an ein „Belohnungssystem" zu koppeln, birgt ein hohes Risiko der metrischen Verzerrung.

Metrik als Störung

Für Coachs gibt es noch einen anderen Einsatz von Metriken, der trotz Austins Bedenken funktioniert: Die Metrik als „Jiggler". Diese Bezeichnung stammt aus dem zweiten Weltkrieg und wurde von Gerald Weinberg in die IT übertragen (vgl. [Wei85]). Während des Kriegs hatten die

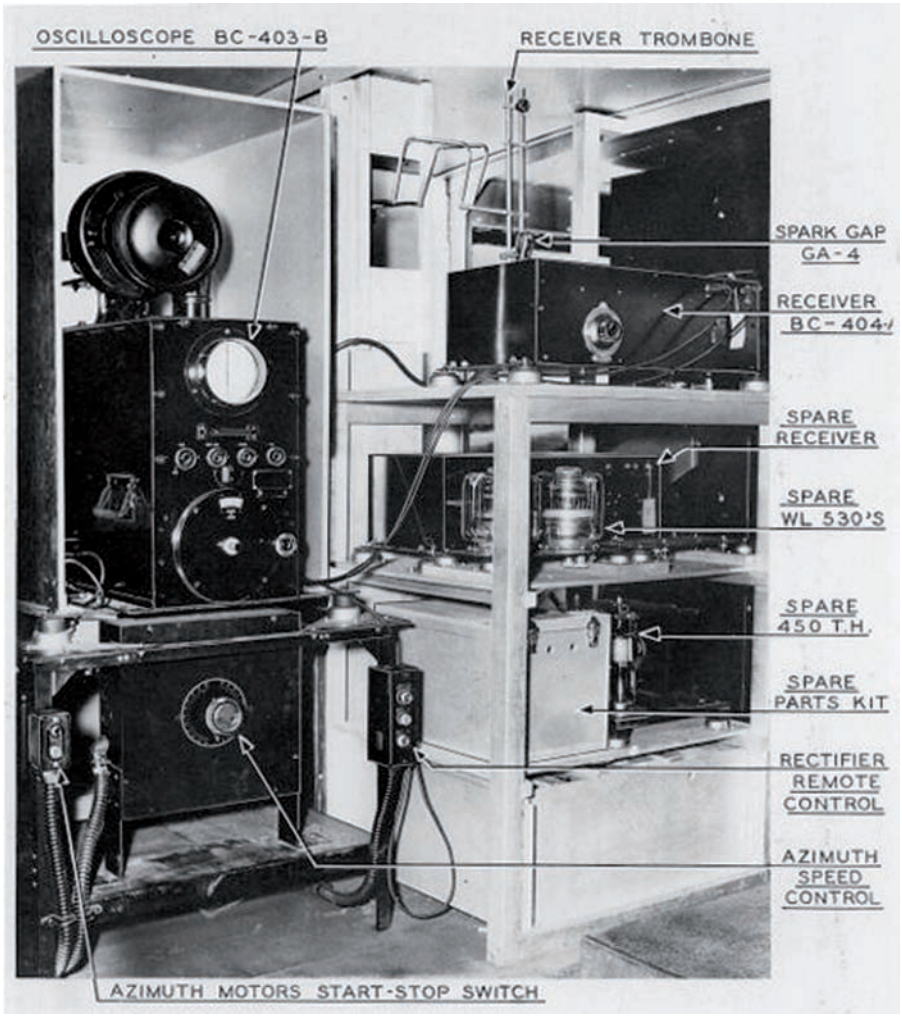

Abb. 2: *Eine mobile SCR-270 Radaranlage, wie sie von der US Armee im zweiten Weltkrieg verwendet wurden. Einige Typen fest montierter Radaranlagen mussten damals künstlich immer wieder gestört werden, um sich nicht in dysfunktionalen Zuständen aufzuhängen.*[2])

Entwickler der ersten Radargeräte ein Problem (**vgl. Abb. 2**): Die auszuwertenden Signale waren so schwach, dass man sehr leistungsfähige Verstärker brauchte – damals Röhrenverstärker. Diese Verstärker gerieten aber von Zeit zu Zeit in eine stabile Rückkopplungsschleife und der Schirm wurde einfach grün. Das geschah allerdings nur bei den fest installierten Geräten, nicht bei denen, die in Fahrzeugen installiert waren. Zudem reichte es, kräftig am Regal zu rütteln, um den stabilen Zustand zu durchbrechen. Darauf wurden so genannte „Jiggler" installiert – Geräte, die in zufällig gewählten Abständen am System rüttelten: eine sehr pragmatische Lösung.

Heute kann man das Verhalten des Systems aus der Chaostheorie erklären: Wie alle Verstärker hatten auch die Radarverstärker aus verschiedenen Gründen eingebaute Rückkopplungsschleifen, zeigten also ein „Eigenleben", das sich zum Beispiel durch Resonanzen bemerkbar machte. Waren die Eingangsdaten zu „sauber", geriet das System in einen stabilen Schwingungszustand, der dazu führte, dass es nicht mehr auf Signale von außen reagierte. Durch das Rütteln wurde der Zustand kurzzeitig gestört und die stabile Schwingung damit durchbrochen. Einen ähnlichen Effekt machen sich Notärzte und Rettungsassistenten zunutze, wenn sie ein Herzkammerflimmern defibrillieren, oder eine Kindergärtnerin, die das allgemeine Chaos durch einen lauten Schrei unterbricht. All dies sind Jiggler, also Störungen, die unerwünschte aber stabile Systemzustände unterbrechen, damit das System wieder reagieren kann und nicht mehr nur in sich selbst gefangen ist.

Natürlich garantiert eine solche Störung alleine nicht, dass sich anschließend der gewünschte Zustand einstellt. Bei den Radargeräten war das wohl der Fall. Bei der Defibrillation muss das Notfall-Team – anders als im Fernsehen – in der Regel mit Sauerstoff, Lidocain, fast letalen Mengen von Adrenalin und einer guten Portion Glück nachhelfen, damit das Herz wieder seinen Eigenrhythmus findet: ein Vorhaben, das oft genug schief geht. Die Kindergärtnerin wird den Schrei auch nur dann nicht vergebens von sich geben, wenn sie anschließend ein entsprechend attraktives Angebot macht, zum Beispiel eine Geschichte vorliest. Die Störung hat lediglich dazu geführt, dass der stabile, aber unerwünschte, Zustand beendet wurde und in einen instabilen Zustand übergegangen ist, der Veränderungen ermöglicht. Er bildet damit die Voraussetzung dafür, dass mit anderen Maßnahmen ein neuer, erwünschter Zustand erreicht werden kann.

Was haben nun Radargeräte, Notfallmedizin und Kleinkinderpädagogik mit Softwareprojekten und Metriken zu tun? Führt ein Team eine Metrik neu ein, verändert diese Metrik zunächst den Blick des Teams auf das Projekt. Das stört den stabilen Zustand, in dem sich das Team befindet, und erzeugt damit die notwendige Instabilität für Veränderungen. Zu theoretisch? Nun gut.

Vor einiger Zeit stand ich während einer Retrospektive vor einem Team, das sich eine geraume Zeit über die Frage stritt, ob man die Tests vor der Implementierung schreiben solle, oder ob es auch in Ordnung wäre, die Tests nach der Implementierung zu bauen. Da niemand im Team Erfahrungen mit echter testgetriebener Entwicklung hatte, wurde die Diskussion reichlich theoretisch und entsprechend wenig zielführend. Um den Disput ein wenig zu erden, schlug ich ein Experiment vor: Ein Teil des Teams sollte mit meiner Unterstützung für eine Iteration sauber testgetrieben arbeiten. Am Ende der Iteration würden wir die Testüberdeckung dieses Teams mit der vergleichen, die andere Teams erreichten. Das Experiment wurde aus verschiedenen Gründen nicht durchgeführt, aber einige Tage später kam einer der Entwickler auf mich zu: Er habe auf Grund der Diskussion zum ersten Mal sei-

2) *Foto: Harold Zahl, Public Domain,* http://www.monmouth.army.mil/historian/photolist.php?fname=Radio%2FSCR+270+and+271.

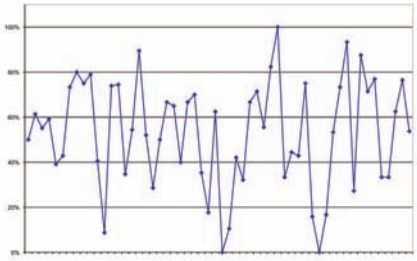

Abb. 3: Beispiel für eine Indikator-Metrik: Anteil der erfolgreichen Builds in einem Projekt über einen längeren Zeitraum. Die Einbrüche signalisieren Krisenmomente.

ne Testüberdeckung gemessen und sei dadurch auf einige üble Lücken in den Tests gekommen. Am liebsten hätte er die Messungen sofort in den Build-Prozess integriert und für alle Entwickler verbindlich gemacht. Ich riet ihm, zunächst ein wenig Erfahrungen zu sammeln. Er solle die Messungen für sich bis zum Ende der Iteration durchführen und dann auf der nächsten Retrospektive darüber berichten.

Vier Wochen später berichtete er fast 20 Minuten lang über typische Testlücken, die er so aufgespürt hatte, und darüber, wie er diese Lücken in Zukunft vermeiden könnte. Den Gesichtern der Zuhörer war anzusehen, dass ihre Tests nicht sehr viel besser aussahen und jeder bei sich einen Merker setzte, in Kürze einmal die Testüberdeckung zu messen. Auf die Frage aus dem Team, ob man die Überdeckungsmessung nicht in den Build-Prozess integrieren solle, winkte der Kollege aber ab: „Das bringt nichts, da fängt man nur an, Tests gegen die Zahlen zu schreiben. Ich werfe die Messung alle paar Tage mal an, um neue Lücken zu finden. Die Zahl, die hinten raus kommt, finde ich nicht besonders spannend." Da der Entwickler einen sehr guten Ruf im Team genoss, einigte man sich auf seine Vorgehensweise.

In den kommenden Monaten wurden so einige interessante Testlücken gefunden. Die Überdeckung wurde auf ca. 60 % verbessert – ein exzellenter Wert für Teams, die nicht testgetrieben arbeiten. Die Frequenz der Messungen nahm allerdings immer weiter ab: „Da kommt nicht mehr viel Neues, wir kennen die wichtigsten Fallen jetzt", antworteten die Entwickler fast unisono, wenn sie darauf angesprochen wurden.

Die Überdeckungsmessung diente in diesem Beispiel als Jiggler: Die Entwickler waren bei der Formulierung ihrer Tests in einen stabilen Trott geraten, den der Blick auf die Testüberdeckung gelegentlich störte. Die Messung war mit keinerlei Zielen verbunden, sondern wurde rein informativ als weiteres Werkzeug eingesetzt. Die wichtigste Wertschöpfung war zudem die Einführung der Messung: Sie störte und änderte das System. War die Metrik einmal etabliert, brachte sie keine deutliche Verbesserung mehr, das System wurde wieder stabil. Eine metrische Verzerrung kam allerdings auch kaum vor, weil es keine Zielvorgaben für die Metrik gab.

Dieser Einsatz von Metriken ist typisch: Ob es eine Statistik über die erfolgreichen Builds ist (siehe Abb. 3), Ergebnisse automatisierter Performancetests oder statische Codeanalysen: Der Nutzen einer Metrik ist am höchsten, wenn sie gerade eingeführt wurde und das Team verstanden hat, wie die Ergebnisse zu bewerten sind. Je mehr das Team lernt, um so weniger braucht es noch die Metrik, um Kurs zu halten: Es übernimmt diesen in seinen Erfahrungsschatz, so wie ein erfahrener Autofahrer ein recht gutes Gefühl für seine aktuelle Geschwindigkeit hat und die Situationen kennt, in denen ihn dieses Gefühl betrügt und er besser auf den Tacho schaut. Der Nutzen der Metrik nimmt also mit der Zeit ab. Tritt dazu noch metrische Verzerrung ein, wird der Nutzen sogar negativ (siehe Abb. 4).

Beim Einsatz von Metriken sollte man sich dieser Dynamik bewusst sein. Eine mögliche Konsequenz für das Team ist es, Metriken immer wieder zu überprüfen:

- Hilft uns diese Metrik noch weiter?
- Gewinnen wir aus ihr wirklich noch wertvolle Informationen?
- Oder gibt es gar erste Ansätze, sie zu unterlaufen?

Wenn die letzte Frage mit „Ja" beantwortet wird, wäre es ein Fehler zu versuchen, mit weiteren Metriken zu verhindern, dass die Zahlen unterlaufen werden. Dieser Versuch ist praktisch immer zum Scheitern verurteilt und führt in der Regel zu einem Metriksystem, das so komplex ist, dass niemand mehr seine Auswirkungen versteht. Stattdessen kann man die Metrik entweder in die entsprechenden sozialen Prozesse einbetten oder einfach auf sie verzichten.

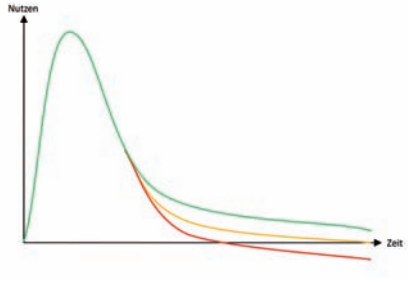

Abb. 4: Der Nutzen einer Metrik über die Zeit. Der anfängliche Nutzen ist hoch und eine Folge der Störung festgetretener Pfade. Dann nimmt der Nutzen ab und schlägt bei metrischer Verzerrung in Schaden um. Der Einsatz der Metrik muss daher rechtzeitig abgebrochen werden.

Wer regelmäßig neue Metriken ausprobiert, wird bald froh sein, wenn er auf diese Weise auch einmal wieder metrischen Ballast abwerfen kann.

Konsequenz zeigen hier einmal wieder die Anhänger des XP: Sie folgen der Regel, dass zu jedem Zeitpunkt immer nur eine Metrik gepflegt wird – neben der binären Metrik des Build-Servers, ob Build und Tests laufen. Führt diese Metrik nicht mehr zu spürbaren Verbesserungen, wird sie abgeschafft.

Erwähnen möchte ich hier noch die Option, eine lang laufende Metrik als Störungsindikator zu verwenden: Manche Teams integrieren Lasttests in ihre regelmäßigen Build-Prozess, sodass sie zu einer ausgewählten Menge von Anwendungsfällen regelmäßig Performanceangaben bekommen. Diese Angaben werden dann als Kurve über die Zeit dargestellt. Die absoluten Zahlen schwanken statistisch und sind dabei eher uninteressant. Wichtig sind plötzliche Ausreißer und längerfristige Tendenzen: Sie deuten darauf hin, dass sich das Performanceverhalten des Systems plötzlich dramatisch verändert hat. Werden solche Messungen oft genug gemacht – z. B. täglich oder einmal pro Woche –, lassen sich die Veränderungen der Performance oft recht einfach bestimmten Änderungen im Code zuordnen. Metrische Verzerrung spielt auch hier keine Rolle, wenn Folgendes gilt:

- Es werden nur Veränderungen der Werte beachtet, nicht die absoluten Messungen.
- Veränderungen ziehen nur die Verpflichtung zur Analyse nach sich, nicht die Verpflichtung, die Änderung wieder umzukehren.

- Das Team ergänzt die Lastmessungen immer wieder und prüft es auf Stichhaltigkeit.

Auch hier wird die Metrik also in ein umfangreicheres System eingebunden, das metrischer Verzerrung entgegenwirkt. Ähnlich setzen erfahrene Scrum-Teams auch Statistiken über ihre *Velocity* ein.

Fazit

Die verbreitete Technik, Systeme über Kennzahlen zu steuern, funktioniert gut bei deterministischen Systemen, deren Verhalten sich auf wenige Kennzahlen reduzieren lässt. Die Softwareentwicklung gehört nicht in diese Klasse von Systemen, sondern ist ein komplexes System, in dem Metriken nicht vorhersehbare Auswirkungen haben. Da jede Metrik nur einen Teil ihrer Auswirkungen erfasst, kann es dazu kommen, dass durch die Konzentration auf Kennzahlen letztlich das Gegenteil dessen erreicht wird, was die Metrik bezweckt – ohne dass die Metrik diese Abweichung registrieren würde. Man spricht dann von „metrischer Verzerrung". Ein komplexes System kann daher nicht sinnvoll über Kennzahlen gesteuert werden.

Dennoch lassen sich Metriken im Projektalltag sinnvoll einsetzen, um auf mögliche Risiken hinzudeuten. Dafür müssen aber einige Voraussetzungen erfüllt sein: Urheber, Erfasser und Auswerter der Metrik müssen identisch sein, ihre Auswirkungen müssen regelmäßig auf metrische Verzerrung hin geprüft werden und die Metrik muss in einen sozialen Kontext eingebunden sein, der sicherstellt, dass die ursprünglich mit der Metrik verbundenen Ziele auch erreicht werden. Die intelligente Kontrolle der Metrik durch die Gruppe stellt sicher, dass dem Missbrauch einer Metrik frühzeitig entgegengewirkt werden kann oder dass die Metrik schnell genug abgeschaltet werden kann, wenn sie sich als schädlich erweist.

Ein funktionierender Anwendungsfall in Veränderungsprojekten ist die Metrik als Störung: Eine neu erfasste Metrik rüttelt auf und hilft, eingefahrene Wege zu verlassen. Da metrische Verzerrung häufig verzögert einsetzt, wird die Metrik aber nur so lange erfasst, bis ihr Zweck erfüllt ist. Stellt man metrische Verzerrung fest, wird der Einsatz sofort abgebrochen. Keinesfalls sollte man versuchen, der Verzerrung mit weiteren, komplexeren Metriken entgegen zu wirken.

Ein letzter Anwendungsfall ist die Metrik als Monitor. Hier wird eine Metrik laufend erfasst, aber nur unerwartete Sprünge werden analysiert, weil sie auf Probleme hindeuten könnten. Bei dieser Einsatzart ist es besonders wichtig, dass Urheber, Erfasser und Auswerter der Metrik identisch sind.

Metriken sind ein scharfes und wichtiges Werkzeug im Projektmanagement. Man darf bei ihrem Einsatz aber nicht so tun, als beobachte man eine Maschine, sondern muss sich stets vergegenwärtigen, dass Projekte soziale Systeme sind, die durch die Beobachtung verändert werden. Beachtet man die Dynamik dieser Veränderung nicht, spielt sie gegen einen selbst. ■

Literatur & Links

[Aus96] R.D. Austin, Measuring and Managing Performance in Organisations, Dorset House 1996

[Wei85] G.M. Weinberg, The Secrets of Consulting – A Guide to Givving & Getting Advice Successfully, Dorset House 1985