Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association Suisse des Electriciens, de l'Association des Entreprises électriques suisses

Band: 84 (1993)

Heft: 25

Artikel: Auf der Jagd nach Software-Läusen : Software-Metriken

Autor: Schild, Rudolf

DOI: https://doi.org/10.5169/seals-902767

Software dringt in immer neue Gebiete vor, und gleichzeitig nimmt ihre Komplexität in starkem Masse zu. In ähnlichem Ausmass gewinnt in der Folge die Qualitätssicherung an Bedeutung, wobei man sich darüber einig ist, dass nicht nur die Qualität der Produkte selbst, sondern auch diejenige der Entwicklungsprozesse überwacht und verbessert werden muss. Dazu bedient man sich heute, wenn auch noch zögernd, der Technik der Software-Metriken. Was man sich darunter vorzustellen hat, zeigt dieser Artikel anhand einiger Beispiele.

Auf der Jagd nach Software-Läusen

Software-Metriken

■ Rudolf Schild

Fehlerhafte Software ruiniert Versicherungsgesellschaft

In der «Montreal Gazette» vom 22. Februar 1992 fand sich ein Artikel von Jay Bryan über die Auswirkungen von Fehlern in einem integrierten Computerprogramm einer Versicherungsgesellschaft (frei übersetzt und zitiert nach [1]):

«Als die Montreal Life Insurance Co., ein florierendes Versicherungsunternehmen, vor zehn Jahren beschloss, ihr Informationssystem auszubauen, wurden keine halben Sachen geplant. Das neue ‹integrierte› System würde sämtliche Bereiche und Aspekte der Firma miteinander verknüpfen und alle Bedürfnisse abdecken. Es gab allerdings ein kleines Problem. Das umfangreiche Programm – eine Million Zeilen – musste in ziemlicher Eile angepasst und installiert werden. So geschah es denn, dass sich hier und dort im Programm auch einige unentdeckt gebliebene Bugs, wie die Software-Leute ihre Fehler gerne liebevoll und verharmlosend zu bezeichnen pflegen, versteckten. Und weil das System ja integriert war, wurden durch einen Fehler in den Daten einer Abteilung jedesmal auch die Daten von mehreren anderen Abteilungen gleichzeitig mitbetroffen und verfälscht.

Es ging kein Jahr ins Land, da waren die Auswirkungen bereits zu spüren: Montreal Life schrieb rote Zahlen. Nach drei Jahren war das Unternehmen dem Zusammenbruch nahe. Fehler in den Provisionszahlungen vertrieben die meisten der Agenten, welche gleich auch ihre Kunden mitnahmen. Wenn die Agenten nicht unterbezahlt waren, so waren sie überbezahlt, was sich zu einem weiteren Millionenverlust summierte. Schliesslich wurde das, was von Montreal Life noch übrig war, von den Eigentümern verkauft; die meisten der obersten Manager verloren ihre Stelle ...»

Horrorgeschichten dieser und ähnlicher Art gibt es viele, einige mit noch schlimmerem Ausgang (Verlust von Menschenleben), die meisten aber harmloser, wenn auch lästig für die Betroffenen. Und immer wieder läuft es auf dasselbe hinaus: fehlerhafte Software. Natürlich kommt es auch bei Nicht-Software-Produkten vor, dass etwas schief geht, zum Teil ebenfalls mit katastrophalen Folgen. Aber zwischen Software und konventionellen Produkten besteht ein grundlegender Unterschied, der sich auch auf die Zuverlässigkeit auswirkt:

Software ist nicht stetig

Anders ausgedrückt: In einem Softwareprodukt kann eine beliebig kleine Änderung eine beliebig grosse Auswirkung haben. Das hat in zwei Bereichen einen wesentlichen Einfluss: Zum ersten kann man aus kleinen Änderungen an den Eingangsdaten nicht unbedingt auf kleine, mehr oder weniger proportionale Änderungen der Ausgangsdaten schliessen; Interpolation und Extrapolation sind grundsätzlich nicht zulässig, was das Testen von Software äusserst schwierig macht. Wenn ein Balken mit 10 Tonnen bela-

Dieser Aufsatz ist die erweiterte Fassung eines Vortrags, den der Autor an der ITG-Frühjahrstagung 1993 vom 11. März 1993 über Softwarequalität technischer Systeme in Zürich gehalten hat.

Adresse des Autors:
Dr. *Rudolf Schild*,
Infogem AG, Rütistrasse 9, 5401 Baden.

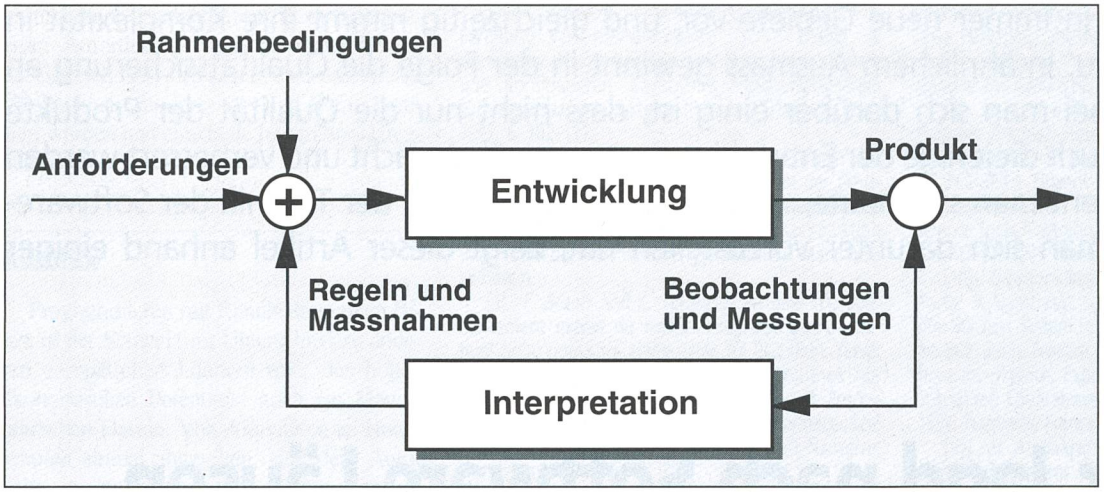

Bild 1 Regelkreis für Produktivität

stet wird und sich um 1 cm durchbiegt, dann kann man daraus schliessen, dass die Durchbiegung bei kleinerer Belastung höchstens gleich gross sein wird. Nicht so bei der Software: wenn die Eingangswerte von 1 bis 9 und von 100 bis 199 die richtigen Resultate liefern, ist für die Werte von 10 bis 99 prinzipiell noch nichts ausgesagt. Es könnte zum Beispiel sein, dass zweistellige Zahlen nicht richtig verarbeitet werden.

Zum zweiten können kleine Fehler im Programmcode beliebig grosse, vollkommen unvorhersagbare Auswirkungen auf die Funktion des Systems haben. Das macht das Testen von Software noch schwieriger. Wenn in einem elektrischen Schwingkreis ein Widerstand ein wenig von seinem Soll-Wert abweicht, dann wird die Resonanzfrequenz des Kreises ebenfalls ein wenig von ihrem Soll-Wert abweichen, aber diese Abweichung ist, zumindest für kleine Störungen, beschränkt. Nicht so bei Software. Ein einziges falsches Bit im Code kann unter Umständen – zum Beispiel bei einer ganz bestimmten zeitlichen Abfolge von Eingangssignalen – das System zum Abstürzen bringen; das heisst es funktioniert nicht ein wenig anders, sondern überhaupt nicht mehr.

Es ist also nicht verwunderlich, dass ausgelieferte Software mit Fehlern behaftet ist, manchmal sogar noch nach jahrelangem Einsatz und praktischer Erprobung. Und es ist auch verständlich, dass die Verantwortlichen sich mehr und mehr mit der Frage beschäftigen, was man dagegen unternehmen kann. Wie, so lautet die Frage, kann man das Ziel, qualitativ hochwertige Software zu produzieren, erreichen?

Kann man Softwarequalität messen?

Um zu wissen, ob man sich diesem Ziel wirklich nähert, muss man sich zunächst im klaren sein, worüber man spricht. Meistens wird eine Entwicklerin oder ein Anwender intuitiv sagen können: «Diese Software ist besser als jene» oder «Die Qualität dieser Version wurde gegenüber der vorhergehenden verbessert». Was aber ist genau darunter zu verstehen? Wenn wir eindeutige, nachprüfbare, quantifizierbare Aussagen über Softwarequalität machen wollen, dann müssen wir sie zunächst einmal messen können. Aus den gemessenen Werten hoffen wir anschliessend, bei richtiger Interpretation, Schlüsse bezüglich der Qualität ziehen zu können. Auf diese Art entsteht der Regelkreis von Bild 1, der uns dem Ziel immer näher bringt.

Solche Kennzahlen, ermittelt nach Software-Metriken, werden heute bereits an verschiedenen Orten mit Erfolg eingesetzt, zum Beispiel im Zusammenhang mit Software-Qualitätssicherung und Verbesserung des Entwicklungsprozesses. Warum also wird diese Technik nicht in grösserem Umfang eingesetzt? Wo noch nicht gemessen wird, hört man im allgemeinen die folgenden Einwände:

Softwareprodukte sind zu unterschiedlich

Denselben Einwand könnte man bei manchen anderen Produkten auch anführen, beispielsweise bei Automobilen. Dennoch werden für Automobile umfangreiche Mengen von Kennzahlen ermittelt, und es hängt von den Prioritäten des Kunden ab, welche Kennzahlen er verwenden und wie er sie interpretieren will.

Softwareprojekte sind zu unterschiedlich

Dagegen lässt sich sagen: Es ist gewiss nicht sinnvoll, völlig verschiedene Projekte miteinander zu vergleichen; hingegen wird ein Vergleich zwischen ähnlichen Projekten in ähnlicher Umgebung sehr wohl aussagekräftig sein.

Die Anwendungsgebiete sind zu unterschiedlich

Natürlich ist es nur sinnvoll, Produkte aus demselben Anwendungsgebiet miteinander zu vergleichen.

Messungen haben keinen Einfluss auf die Qualität oder Produktivität

Das ist prinzipiell überall wahr, wo Messungen gemacht werden. Eine Messung verbessert die Qualität nicht direkt; sie sagt aber etwas aus über die Qualität und kann damit dazu beitragen, dass die Qualität des nächsten Produktes besser sein wird.

Es besteht die Gefahr der Fälschung von Kennzahlen

Das ist vor allem dann wahr, wenn Messungen zur Beurteilung von Einzelpersonen und nicht von Produkten, Prozessen oder auch Teams verwendet – oder eher missbraucht – werden. Wenn aber die Messungen den Entwicklern zugute kommen, so ist der Anreiz zur Verfälschung kaum noch vorhanden.

Messungen sind zu teuer

Die Frage ist in Wirklichkeit nicht, ob wir es uns leisten können, Messungen durchzuführen, sondern ob wir es uns leisten können, sie nicht durchzuführen.

Was spricht für Messungen?

Nachdem die Einwände entkräftet sind, stellt sich die Frage, welchen Nutzen wir aus Messungen ziehen können. Dabei kann man vier Gebiete unterscheiden:

- Überwachung des Projektfortschritts, mit Abschätzung des weiteren Verlaufs
- Sicherung der Produktqualität, das heisst der Qualität des ausgelieferten Softwareprodukts
- Sicherung der Projektqualität, das heisst der Qualität des Entwicklungsprozesses, nicht des entwickelten Produkts

– Entscheidungshilfen für die Produktentwicklung, beispielsweise Kennzahlen für zugekaufte Software

Auf allen vier Gebieten, mit unterschiedlicher Ausprägung, lassen sich drei Arten von Nutzanwendungen für Kennzahlen ausmachen:

1. Das Erkennen von Schwachstellen: Wo sind im Produkt am ehesten Fehler zu erwarten? Wo kann der Entwicklungsprozess verbessert werden?
2. Die Beobachtung von Trends und Verteilungen: Lassen sich Vermutungen aufstellen oder erhärten?
3. Die Verwendung von Bezugsgrössen für Vorhersagen: Welcher Aufwand wird benötigt? Wie gross ist die Anzahl der verbliebenen Fehler?

Dazu könnte in Zukunft, wenn sich Software-Metriken etabliert haben, vielleicht noch das folgende Gebiet kommen: Vorgaben für Spezifikation und Entwicklung. Wenn Begriffe wie Benutzerfreundlichkeit – heute schon in aller Munde – nicht mehr nur Schlagwörter sind, sondern auch messbar werden, könnte beispielsweise eine «Benutzerfreundlichkeit von mindestens 15 Smiles» verlangt werden (1 Smile = Einheit der Benutzerfreundlichkeit).

Was sind Software-Metriken?

Wenn man sich einmal dafür entschieden hat, Software zu messen, geht es als nächstes darum, festzustellen, was man messen will und was man aus den gemachten Messungen schliessen kann. Für eine Metrik müssen folgende fünf Eigenschaften definiert sein:

– ihr Name
– eine Regel, was zu messen oder zu zählen ist
– der Algorithmus, nach welchem die Kennzahl aus den Messungen berechnet wird
– ihre Einheit
– eine Interpretation, welche angibt, was wir für unseren Problemkreis aus der Zahl schliessen können

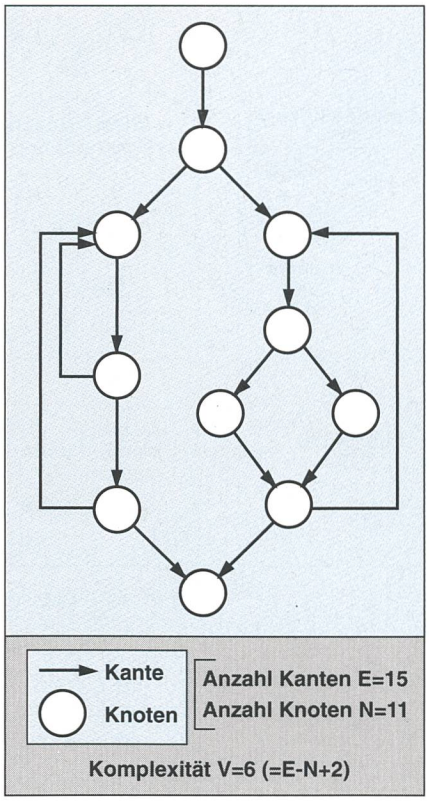

Bild 2 Zyklometrische Komplexität eines Programmgraphen

Zwei Beispiele sollen das verdeutlichen. Das erste ist absichtlich aus einem völlig anderen Gebiet gewählt, während das zweite eine Software-Kennzahl liefert, welche in der Praxis verwendet wird.

Beispiel 1

Name	Ackerfläche
Regel	Schreite die Länge (L) und die Breite (B) ab und zähle die Schritte
Algorithmus	$F = L \cdot B$
Einheit	Quadratschritte
Interpretation	Die benötigte Saatmenge und die erwartete Ernte sind proportional zur Ackerfläche F

Beispiel 2

Name	Zyklometrische Komplexität
Regel	Zähle die Kanten (E) und die Knoten (N) im Steuerflussgraphen des Programms
Algorithmus	$V = E - N + 2$
Einheit	Dimensionslos
Interpretation	Je grösser V ist, desto schwieriger wird das Testen des Codes sein

Bild 2 veranschaulicht diese Metrik. Man beachte, dass die Angabe der Interpretation nicht etwa fakultativ ist. Eine Grösse kann fast immer auf verschiedene, unter anderem auch unzulässige Arten, interpretiert werden, und es muss daher angegeben werden, wozu die Kennzahl wirklich brauchbar ist. So ist es beispielsweise nicht statthaft, die zyklometrische Komplexität als Gütemass zu postulieren, in der Art, dass etwa vorgeschrieben würde, keine Routine darf ein $V > 15$ aufweisen.

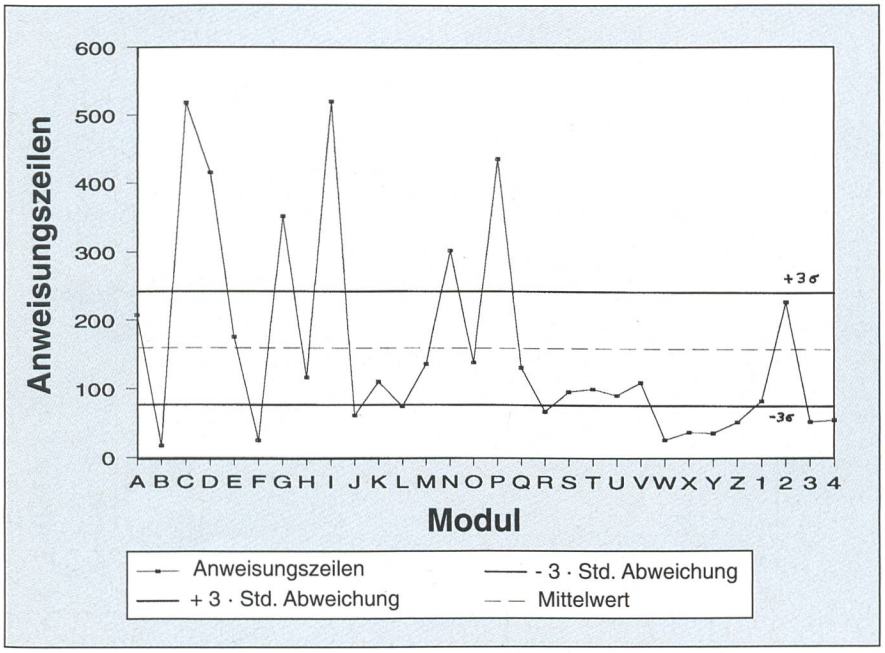

Bild 3 Schwachstellen erkennen: Grössen von Moduln

Was bringt der Einsatz von Software-Metriken?

Wie oben erwähnt, können Metriken auf verschiedene Art zur Qualitätsüberprüfung und -verbesserung eingesetzt werden. Einige Beispiele sollen das erläutern.

Erkennen von Schwachstellen

Eine der ersten verwendeten Software-Kennzahlen war die Programmgrösse, wobei verschiedene Möglichkeiten zu deren Definition verwendet wurden. Die wohl am häufigsten angewandte Regel bestand darin, ganz einfach die Programmzeilen zu zählen – vielleicht unter Ausschluss der Kommentare und der Leerzeilen –, denn das war praktisch ohne Aufwand zu machen. Die Kennzahl Lines of Code (LOC, häufig auch KLOC = Kilo Lines of Code, also tausend Programmzeilen) ist allerdings ziemlich umstritten. Ohne hier auf die Argumente dafür und dagegen einzuge-

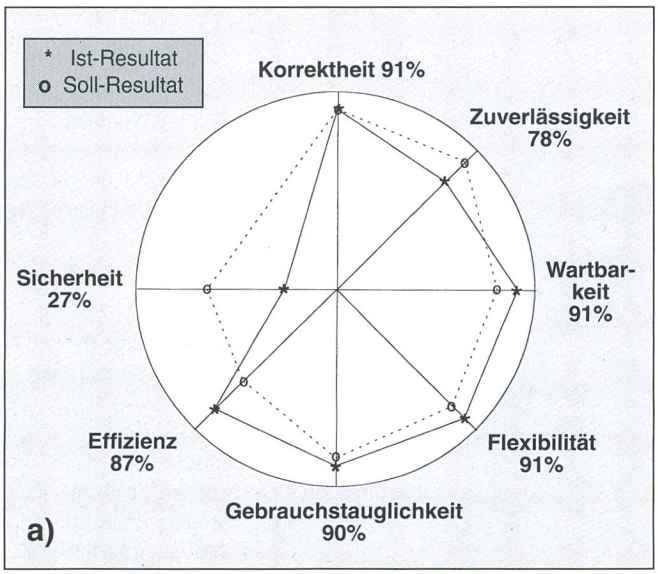

Bild 4a Kiviat-Diagramm mit Prozent-Darstellung

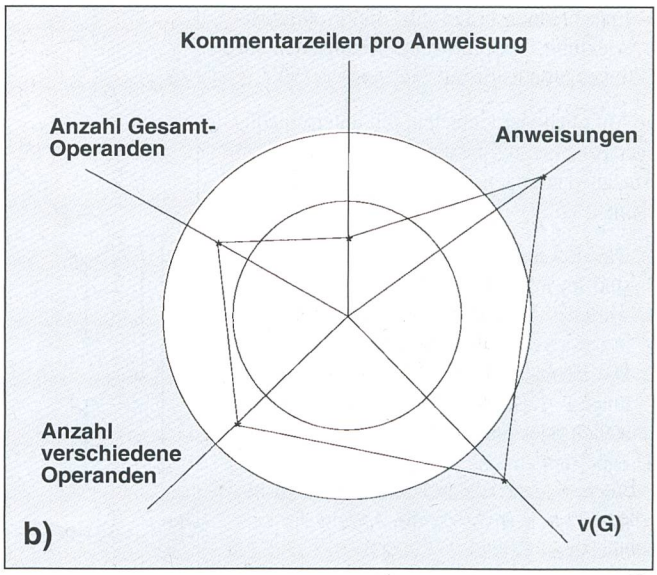

Bild 4b Kiviat-Diagramm mit Soll-Bereich

hen, kann doch gesagt werden, dass LOC in beschränktem Masse durchaus als Kennzahl beigezogen werden kann, wie das Beispiel von Bild 3 zeigt. Wenn die Anzahl einzelner Moduln genügend gross ist, kann LOC als statistische Grösse betrachtet werden. Bei den Ausreissern ausserhalb des Bereichs von drei Standardabweichungen wird speziell sorgfältig untersucht, ob es für sie einen guten Grund gibt, so klein bzw. so gross zu sein. Unter Umständen müssen einzelne von ihnen überarbeitet werden. Moduln, welche begründetermassen ausnehmend gross sind, müssen möglicherweise auch auf spezielle Art getestet werden.

Eine ausgezeichnete Methode, verdächtige Moduln zu erkennen, besteht im Aufzeichnen eines Satzes von Kennwerten in einem Kiviat-Diagramm (auch Radardiagramm oder Fussspur genannt), von denen zwei Arten in Bild 4 gezeigt sind. Pro Modul wird ein Diagramm erstellt. Beim Diagramm 4a sind Werte zwischen 0 und 100% möglich, wobei 100% immer das Optimum ist; der angestrebte Soll-Wert und der gemessene Ist-Wert werden eingetragen. Beim Diagramm 4b liegen die Soll-Werte innerhalb eines Bereichs, der sowohl über- wie unterschritten werden kann. Alle Axen werden hier so normiert, dass der akzeptable Bereich die Form eines Ringes erhält. Um Missverständnissen vorzubeugen, wurden keine Zahlen angeschrieben; in Wirklichkeit gehören sie natürlich dazu. Ausreisser sind auf einen Blick erkennbar und können einer zusätzlichen Inspektion unterzogen werden. Werte der Art, wie sie in Diagramm 4b vorkommen, sind leicht automatisch zu erfassen; es existieren auch Softwarewerkzeuge, welche die Werte nicht nur ermitteln, sondern sie auch in der gezeigten Form darstellen.

Verfolgen von Trends

Absolute Zahlen sagen wenig aus. Wichtiger ist das Entdecken und Verfolgen von Trends: dieselben Messungen werden zu verschiedenen Zeitpunkten wiederholt. Aus den daraus entstehenden Graphiken können sowohl lauernde Probleme wie auch vorgenommene Verbesserungen erkannt werden. Aus Bild 5 lassen sich zwei Dinge ablesen:

a. Die Anzahl der Programmzeilen pro Modul bleibt – auch über längere Zeit hinweg – ungefähr konstant, was auf einen stabilen Entwicklungsprozess schliessen lässt.
b. Die Releases werden abwechselnd grösser und kleiner. Das stimmt vollkommen überein mit der Strategie des Projektleiters: Nach jeder Auslieferung mit neuen Funktionalitäten folgt eine Konsolidierungsphase, welche mit einer weiteren Auslieferung abschliesst. Erst dann wird wieder neu ausgebaut.

Aufspüren einer Verteilung

Das Bild 6 zeigt die Schachtelungstiefe von Routineaufrufen. Dabei bedeutet zum Beispiel die Tiefe 3: eine Routine ruft eine zweite auf, welche selbst wieder eine dritte aufruft. Auf den ersten Blick zieht man aus dieser Graphik den Schluss: hier wurde eine reichlich komplexe und daher fehleranfällige Lösung gewählt; die maximale Schachtelungstiefe beträgt 17.

Bei genauerem Hinsehen erkennt man aber in der Kurve zwei Buckel, welche drei Schichten trennen: die oberste Schicht (rechts, mit der grössten Schachtelungstiefe)

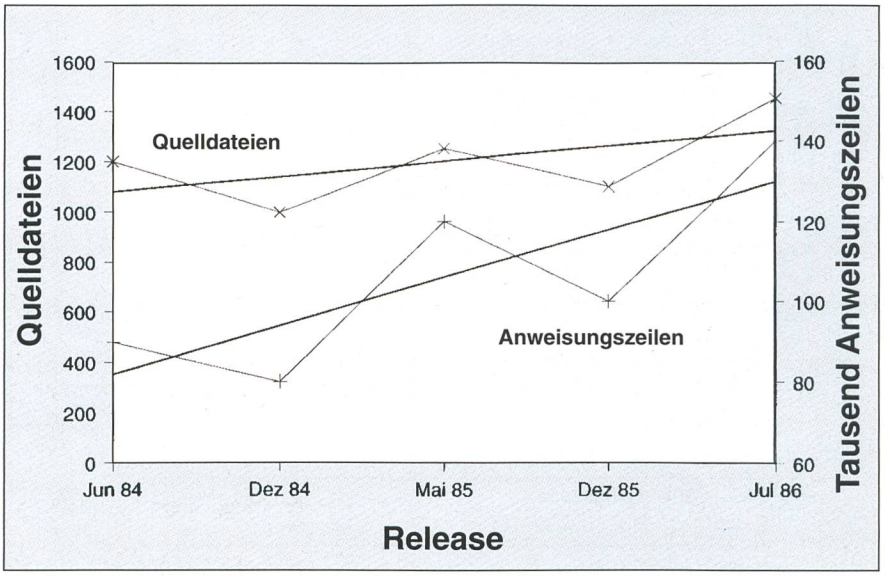

Bild 5 Trends verfolgen: Release-Strategie

ist die eigentliche Anwendung, darunter liegt ein Graphikpaket, und die unterste Schicht machen die hardwarenahen Routinen aus. Unter Berücksichtigung dieser Aufteilung erscheint nur noch die mittlere Schicht als komplex, weil sie in sich selbst immer noch eine maximale Schachtelungstiefe von 9 aufweist.

Verwendung von Bezugsgrössen

Regelmässig gemessene Kennwerte können mit der Zeit als Bezugsgrössen für gewisse Vorhersagen verwendet werden. Auch wenn diese Prognosen nie exakt sein werden, besser als über den Daumen gepeilte sind sie auf jeden Fall.

Eine Kennzahl, welche schon lange an vielen Orten ermittelt wird, ist die Fehlerdichte, das heisst die Anzahl Fehler pro KLOC, und zwar einerseits die Dichte aller Fehler, die gemacht wurden, von welchen aber ein grosser Teil bei den Prüfungen entdeckt und eliminiert wurde, und andererseits die Fehlerdichte nach der Auslieferung, also der Fehler, welche später noch gefunden wurden. Wenn diese, zugegebenermassen wenig schmeichelhaften Daten sorgfältig erfasst und ausgewertet werden, so kann man bei einem stabilen Entwicklungsprozess aus der Programmgrösse auf die Anzahl noch verbliebener Fehler schliessen, nachdem in den Prüfungen bereits eine bestimmte Anzahl davon gefunden wurde.

Eine weitere Anwendung von gesammelten Kennwerten als Bezugsgrössen besteht in der Erhärtung oder Widerlegung von Hypothesen (inkl. überlieferter Softwaremythen). Beispielsweise besagen die Prinzipien des Software-Engineering, dass die Kopplung von Moduln über Parameter der Kopplung über gemeinsame Datenbereiche vorzuziehen ist. Messungen ergeben, dass die Art der Kopplung auf die Fehlerhäufigkeit keinen Einfluss hat, wohl aber auf die Wartbarkeit der Software.

Die Schachtelungstiefe von Routineaufrufen, wie sie in Bild 6 dargestellt wurde, hingegen hat, wie Messungen zeigen, einen Einfluss auf die Fehlerhäufigkeit, so dass damit im nachhinein die Bemerkung betreffend der Fehleranfälligkeit des dort erwähnten Graphikpakets gerechtfertigt ist.

Auch bei dieser Art der Verwendung der Kennzahlen ist wichtig – und nicht immer leicht –, diese richtig zu interpretieren. So kann die Tatsache, dass in der Testphase relativ wenig Fehler gefunden wurden, dahin gedeutet werden, dass die Programme sehr sorgfältig entwickelt und bereits vor dem Testen, beispielsweise in Reviews, gründlich geprüft wurden. Das ist die optimistische Interpretation; die pessimistische Interpretation besagt, dass die Testfälle nicht gut gewählt sind, so dass sie die vorhandenen Fehler nicht aufspüren können.

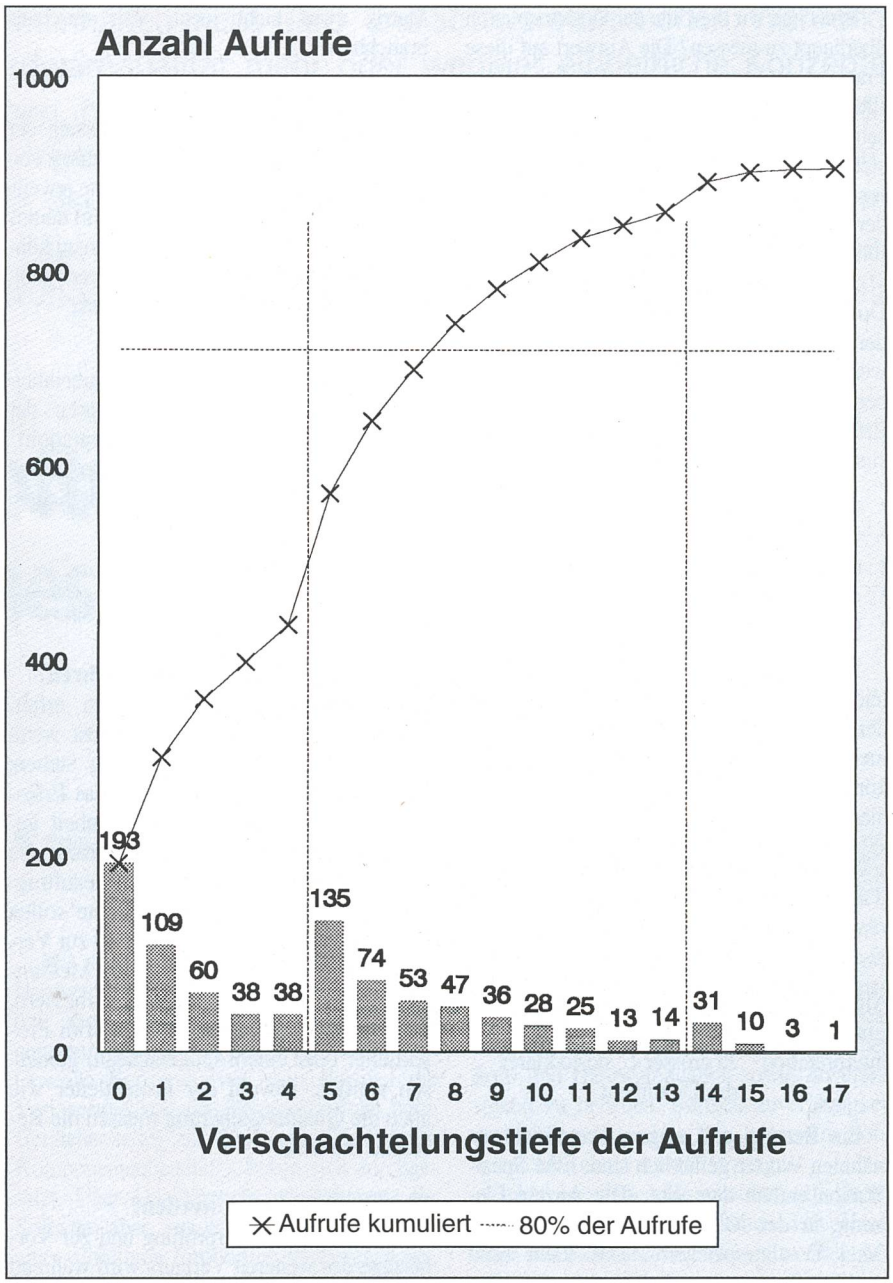

Bild 6 Verteilung aufspüren: Schachtelungstiefe von Routinen

Qualitätsindikatoren

Es kann hier nicht darum gehen, eine vollständige Liste von Indikatoren aufzuführen. Vielmehr sollen einige Hinweise auf mögliche Metriken gegeben werden; schliesslich wird jeder Betrieb die für ihn wichtigen Indikatoren selbst definieren müssen.

Produktindikatoren

Für die Softwareprodukte wird man sich zunächst der Codemetriken bedienen, deren es viele gibt. Sie sind relativ leicht zu ermitteln, allerdings sind nicht alle gleich aussagekräftig. Bei vernünftiger Interpretation sind beispielsweise

- die Programmgrösse (in KLOC)
- die zyklometrische Komplexität
- die Anzahl numerischer Konstanten pro KLOC (als Indikator für die Wartbarkeit)

nützliche Indikatoren. Je nach benötigtem Einsatz werden mit etwas Phantasie leicht weitere geeignete Metriken definiert.

Software besteht aber nicht nur aus Programmen, sondern auch aus Dokumentation und Daten, daher müssen auch diese Aspekte in die Messung von Softwarequalität einbezogen werden. Über die Datenqualität soll hier nicht gesprochen werden, da deren Bestimmung extrem von den spezifischen Daten abhängt; im übrigen kann dazu gesagt werden: Wenn schon die Daten qualitativ ungenügend sind, dann ist die Qualität der Programme und der Dokumentation gewiss nicht das dringendste Problem!

Brauchen wir die Güte der Dokumentation überhaupt zu messen? Die Antwort auf diese Frage ist ein klares Ja. Man gebe sich keinen Illusionen hin: die teuersten Fehler beruhen sehr häufig auf Missverständnissen, welche sich letztlich auf die mangelhafte Qualität irgendwelcher Dokumente, zum Beispiel der Anforderungs-Spezifikationen, zurückführen lässt.

Aber wie messen wir sie? Metriken für die Dokumentation sind nicht sehr leicht zu finden. Fünf Vorschläge sollen hier gemacht werden; einige werden an einzelnen Orten bereits verwendet, bei den andern muss die Erfahrung zeigen, wie brauchbar sie wirklich sind. Die fünf Metriken sind:

- die Grösse
- die Unzweideutigkeit
- der Fog-Index
- die Bilddichte
- das Jury-Urteil

Die *Grösse* (in Seiten, Wörtern usw.) ist leicht zu messen und kann eine Aussage über den Aufwand bei der Wartung (der Dokumentation) machen. Die *Unzweideutigkeit* könnte nach unserem Schema wie folgt definiert werden:

Name	Unzweideutigkeit
Regel	Ermittle Z = Anzahl Wörter wie «auch, immer, nie, usw., anders, ...», und W = Anzahl Wörter gesamt
Algorithmus	$U = (1 - Z/W) \cdot 100$
Einheit	Prozent
Interpretation	Je grösser U, desto klarer das Dokument

Ein Beispiel soll zeigen, warum die erwähnten Wörter gefährlich sind. Eine Spezifikation enthält den Satz «Die Anzahl Elemente in der X-Tabelle wird auch in der Datei D abgespeichert». Das kann (und wird!) auf zwei wesentlich verschiedene Arten interpretiert werden:

Interpretation 1: Ein weiterer Eintrag in D ist die Grösse von X.
Interpretation 2: Die Grösse von X ist sowohl in D wie auch sonstwo gespeichert.

Für den *Fog-Index* werden die Anzahl Buchstaben pro Wort, die Anzahl Wörter pro Satz und die Anzahl Sätze pro Abschnitt ermittelt. Je länger im Durchschnitt die Wörter, die Sätze und die Abschnitte sind, desto mühsamer ist es, den Text zu verstehen. Die *Bilddichte* ist definiert als der Quotient *Anzahl Bilder pro Anzahl Wörter*. Die zugehörige Interpretation heisst: je gösser die *Bilddichte*, desto kleiner der Wartungsaufwand pro Fehler. Das *Jury-Urteil* kann verwendet werden, wenn objektiv messbare Kennzahlen nicht gefunden werden können. Wie die Erfahrung aus dem Eiskunstlauf zeigt, ist eine derartige Metrik zwar nicht ideal, aber durchaus brauchbar.

Projektindikatoren

Bei der Überwachung und Messung der Softwareprojekte, also der Entwicklung von Produkten, sind die ermittelten Werte jeweils mit den geplanten zu vergleichen, um daraus auf den Stand des Projekts schliessen zu können. Vier Möglichkeiten seien hier erwähnt; viele andere bieten sich ebenfalls an:

- die Anzahl Mitarbeiter
- die Anzahl fertiggestellter Softwareeinheiten in den verschiedenen Phasen der Entwicklung (Entwurf, Test, Integration)
- der Projektfortschritt, speziell die aktuell geschätzten noch zu erwartenden Kosten (Cost to Completion)
- der Testfortschritt

Wer misst wann wieviel?

Wer soll die Messungen durchführen?

Die Entwickler selber messen erfahrungsgemäss noch zu wenig, selbst wenn ihnen Werkzeuge zur Verfügung stehen. Das kann sich in Zukunft, wenn das Erfassen von Kennzahlen zur Gewohnheit geworden ist, ändern. Bis dann müssen die Produkt-Metriken von Qualitäts-Beauftragten ermittelt werden. Die Resultate sollen den Entwicklern aber auf jeden Fall zur Verfügung stehen; nur dann helfen die Messungen wirklich mit, die Qualität zu verbessern. Die Projekt-Kennzahlen müssen vom Projektleiter oder einem Qualitätsteam gemessen werden; sowohl der Projektleiter wie auch die Qualitätssicherung müssen die Resultate erhalten.

Wann soll gemessen werden?

Zur laufenden Überprüfung und zur Vorhersage des weiteren Verlaufs wird während der Entwicklung gemessen. Nach Abschluss des Projekts sollen Messungen gemacht werden, welche für die Aufnahme in eine Qualitätsdatenbank bestimmt sind, zur späteren Verwendung für Vorhersagen, Vergleichen, Trendbeobachtungen und als Bezugsgrössen.

Wieviel soll man messen?

Wenn man sich entschlossen hat, ein Software-Metrik-Programm einzuführen, so soll am Anfang nur wenig gemessen werden, etwa ein halbes Dutzend verschiedener Kennzahlen. Später lässt sich diese Zahl nach Bedarf erhöhen. Wenn von Anfang weg grosse Datenmengen vorliegen, so werden ungeübte Teams überwältigt; die Auswertung und die Interpretation wird darunter leiden, und das Metrik-Programm ist von Anfang an gefährdet. Das aber wäre der Qualität, um die es uns hier ja letztlich geht, nicht zuträglich.

Zusammenfassung

Software-Metriken werden im grossen ganzen noch zögernd eingesetzt. Die Erfahrung zeigt aber, dass sie einen wesentlichen Beitrag zur Qualität von Softwareprodukten liefern können, einerseits, indem aus ihnen Rückschlüsse auf das Produkt selber gezogen werden können, andererseits, weil sie auch dazu dienen, den Entwicklungsprozess zu verbessern.

Hätte das eingangs erwähnte Debakel bei der Montreal Life Insurance Co. mit Metriken vermieden werden können? Das Programm selbst wäre durch Messungen kaum verbessert worden. Wie der Zeitungsartikel damals aber weiter ausführte, beruhte der Untergang der Firma wesentlich darauf, dass «die System Manager prinzipiell nur ihren eigenen Gesetzen gehorchen. Sie sind niemandem Rechenschaft schuldig ...» Der Regelkreis war also gar nicht vorhanden. Wären bei der Einführung des neuen Programms die richtigen Messungen gemacht und beachtet worden, so hätten zwar nicht die ersten Verluste, wohl aber der völlige Kollaps vermieden werden können.

Angesichts des ständig zunehmenden Einsatzes immer komplexerer Software und ihres damit unaufhörlich wachsenden Einflusses in allen Gebieten nimmt auch die Bedeutung der Qualitätssicherung rapide zu. Software-Metriken, richtig angewandt, können dabei einen wichtigen Beitrag leisten.

Literatur

[1] *P.G. Neumann* (Moderator): Risks to the public in computers and related systems. ACM Sigsoft Software Engineering Notes, 17(1992)2, S. 3–27.

Métriques pour logiciels

Les logiciels deviennent toujours plus complexes, et le nombre de domaines qu'ils envahissent croît partout. Aussi devient l'assurance de la qualité plus et plus importante. Ce ne sont pas seulement les produits, mais aussi bien les processus de développement dont il faut contrôler et améliorer la qualité. A cette fin on emploie la technique des métriques. Quoiqu'elle ne soit pas encore très répandue, elle se montre très utile. Cet article présente quelques exemples de métriques et de leur emploi.