Das Bewußtsein für Qualität und für die Konsequenzen aus schlechter Arbeit ist im Software-Engineering nach wie vor nur kümmerlich verbreitet. In keiner anderen Ingenieurdisziplin werden Fehler so leichtfertig akzeptiert. Entsprechend hoch ist das Potential zur Verbesserung des Software-Entwicklungsprozesses. Helmut Sandmayr faßt mit dem Begriff «konstruktive Software-Qualitätssicherung» eine ganze Reihe von Maßnahmen zusammen, die einen Beitrag liefern können, Fehler zu vermeiden, und damit einen positiven Einfluß auf Projekt- und Produktqualität haben.

Konstruktive Software-Qualitätssicherung

Qualität ist nicht Sache der Götter

Von Helmut Sandmayr

Es ist wichtig und sinnvoll, für die folgenden Betrachtungen zuerst ein «Koordinatensystem» als Basis für die weiteren Überlegungen festzulegen. Es stehen zunächst die Begriffe *Qualität, Qualitätssicherung, Software* und *konstruktiv* zur Diskussion.
Es ist nicht Absicht, den Begriff Qualitätssicherung zum n+1. Mal aus dem Lexikon oder nach einer Norm zu zitieren. Das Problem läßt sich indirekt angehen. Was ist das, oder was sind die Objekte, deren Qualität zur Diskussion steht?

Dr. math. ETH Helmut Sandmayr, INFOGEM AG, Informatiker-Gemeinschaft für Unternehmensberatung, Stadtturmstraße 18, Postfach 639, CH-5401 Baden.

Es bieten sich zwei Objekte an:

- der Herstellungs*prozeß* und
- das *Produkt* als Resultat des Herstellungsprozesses

Die Qualität des Prozesses ist gekennzeichnet durch Termin-, Kosten- und Sachziele sowie durch den Grad der Erreichung dieser Ziele. Die Qualität des Produkts ist charakterisiert durch die Sachziele und durch den Grad deren Einhaltung.
Die beiden Qualitäten sind dabei gemäß Bild 1 verknüpft.
Die bisherigen Überlegungen gelten allgemein; sie sollen als nächstes auf Software zugeschnitten werden. Was ist denn das *Spezifische an Software*, das hier eine Rolle spielt? Die folgende Aufzählung enthält typische Eigenschaften von Software. Die Punkte sind nicht unabhängig; die meisten lassen sich aus dem ersten ableiten.

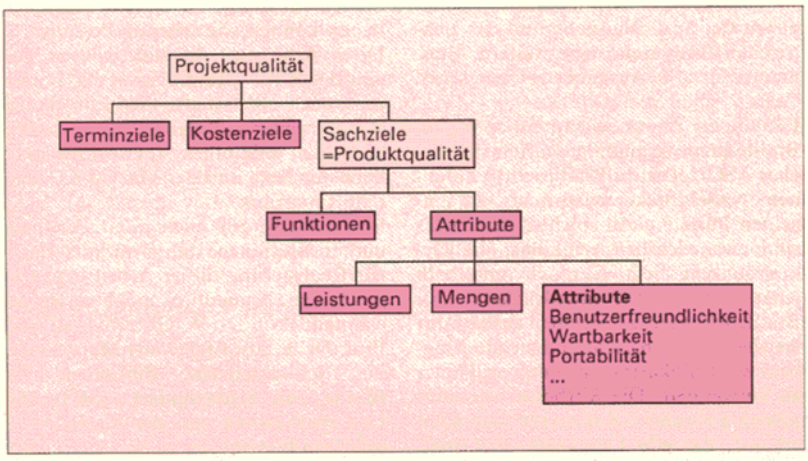
- Software ist nicht an Material gebunden, sie ist das abstrakte Resultat eines Denkprozesses.
- Software ist nicht stetig, das heißt, aus dem Verhalten eines Programms für zwei Eingabewerte können wir keinen Schluß ziehen, wie sich das Programm für einen dazwischenliegenden Wert verhält.
- Der Herstellungsprozeß für Software ist die Entwicklung; die Produktion beziehungsweise die Reproduktion ist trivial und wird (meistens) beherrscht.
- Software nutzt sich nicht ab, sie altert nicht, es entstehen keine neuen Mängel (wenn wir sie in Ruhe lassen).

Folgen dieser Eigenschaften finden sich in jeder Aufzählung der Schwierigkeiten in Softwareprojekten beziehungsweise in der Anwendung von auf Software basierenden Produkten:

- Der Fortschritt in einem Softwareprojekt ist schwer oder gar nicht feststellbar.
- Das Testen von Software ist aufwendig und schwierig.
- Das Ändern von Software führt zu unerwarteten Nebenwirkungen.

Das ist eine Problemliste, die das Herz jedes Qualitätssicherers höher schlagen läßt – ein Eldorado oder ein Alptraum für die Qualitätssicherung?
Für die konstruktive Qualitätssicherung ergeben sich zwei wichtige Konsequenzen. Erstens: Der Herstellungsprozeß beschränkt sich auf die Entwicklung; die Produktion beziehungsweise die Reproduktion ist unwesentlich. Es sind Maßnahmen zu suchen, die dem immateriellen Charakter der Software Rechnung tragen; dabei ist es erlaubt, über die Grenzen der Disziplin zu schauen und zum Beispiel Anleihen bei der Konstruktion zu machen.
Das Thema dieses Aufsatzes ist die *konstruktive Software-Qualitätssicherung*. Er befaßt sich also mit allen Maßnahmen, die einen Beitrag liefern können, Fehler zu vermeiden. Die analytischen Maßnahmen, das Prüfen des Entwicklungsablaufs und das Prüfen des Produkts – die Fehlersuche –, werden bei der weiteren Betrachtung ausgeklammert.
Konstruktive Aspekte für das Produkt, zum Beispiel Robustheit gegenüber falschen Eingaben oder Fehlern in der ausführenden Hardware, sollen hier auch ausgeklammert werden. Sie sind Themen für die Spezifikation der An-

Bild 1. Prozeß- und Produktqualität.



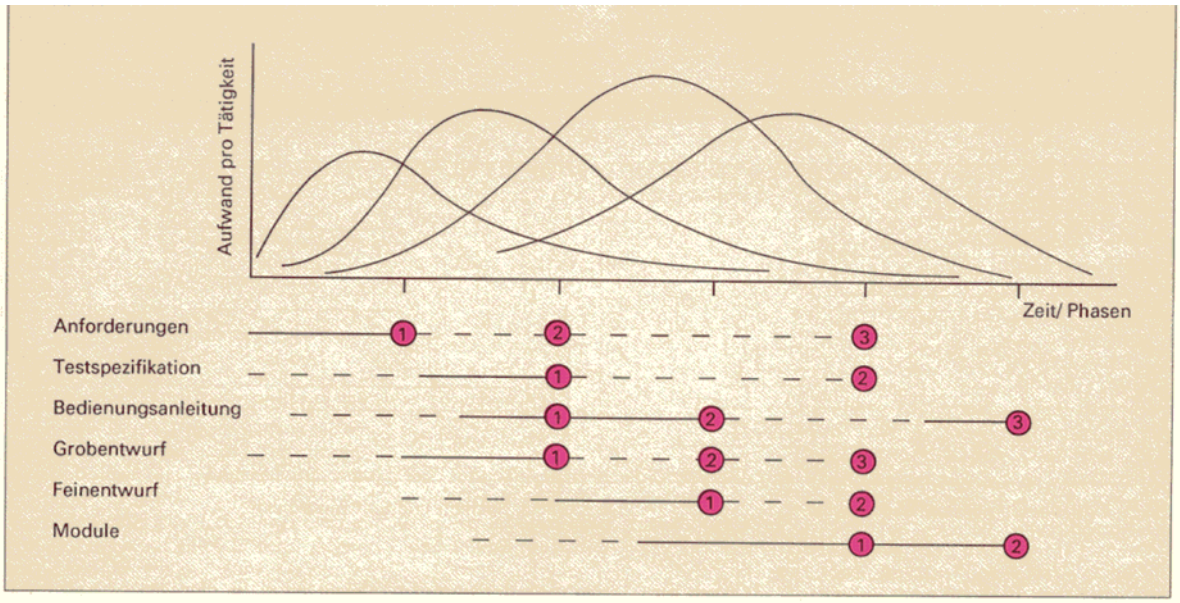


Bild 2. Tätigkeiten und Phasen. (Die eingekreisten Ziffern bezeichnen die Versionen der entsprechenden Dokumente.)

forderungen und die Realisierung, weniger für die Qualitätssicherung.
Es bleibt nun der konstruktive Aspekt für den Entwicklungsablauf, also Maßnahmen in der Entwicklung beziehungsweise im Projekt, die einen positiven Einfluß auf die Projekt- und die Produktequalität haben.

Der Entwicklungsablauf

Ein wichtiges Element der Entwicklung ist der Ablauf beziehungsweise das Verständnis des Ablaufs, die Soll-Strukturen, Ordnungskriterien und Überwachungsmechanismen. Die einzelnen Tätigkeiten im Ablauf werden mit Methoden systematisiert und mit den sie unterstützenden Werkzeugen automatisiert. Der Ablauf hat damit eine Schlüsselfunktion; er bildet die Basis und definiert die Kriterien für die Auswahl von Methoden. Diese haben ihrerseits die gleiche Rolle bei der Auswahl von Werkzeugen.
Die Liste der unter Entwicklungsablauf zu behandelnden Themen umfaßt:

- Phasen
- Tätigkeiten und Methoden für die einzelnen Tätigkeiten
- Änderungskontrolle
- Projektplanung
- Fortschrittserfassung
- Regelungen

Je nach dem Qualitätsbewußtsein einer Firma sind es gerade diese Themen, die im Rahmen eines Qualitätssicherungs-Handbuchs und zugehöriger Verfahren beziehungsweise Richtlinien geregelt sind. Die Erstellung oder die Nachführung dieses Handbuchs, der Verfahren und Richtlinien ist sicher ein konstruktiver Beitrag. Dabei gilt es aber einige Punkte zu beachten, damit der Beitrag nicht zu einem Qualitätsverhinderer wird, indem zum Beispiel ein falsches Verständnis für den Entwicklungsablauf festgeschrieben wird.
Um diese Aussage zu untermauern, soll auf zwei Punkte speziell eingegangen werden: auf den *Phasenplan* und auf die *Schnittstelle Hardware–Software*.

Phasenplan

In verschiedenen Qualitätssicherungs-Handbüchern stößt man auf ein Verständnis von Phasen, das nur folgenden Schluß zuläßt: Entweder hält sich in der betreffenden Firma niemand an den Phasenplan, oder es gibt nie ein Resultat einer Softwareentwicklung.
Analysiert man das Problem etwas genauer, so findet man eine Vermischung von verschiedenen Konzepten, die zum Teil durch eine ungünstige Wahl von Namen verursacht wird. Zu diesen Begriffen gehören *Phase, Tätigkeit, Meilenstein, Review, Wasserfallmodell*. Im folgenden wird eine konsistente Belegung der Begriffe aufgezeigt – ein Modell, das in einem QS-Handbuch verwendet werden kann.
Zunächst zum Begriff Phase: Eine *Phase* ist eine Zeitspanne, ein Abschnitt, der durch irgendwelche Eigenschaften definiert wird, zum Beispiel die Zeitspanne, in der der Mond von der Sonne beleuchtet wird.
Eine Phase in einer Entwicklung ist ein Abschnitt, in dem etwas Typisches geschieht, an dessen Resultat jemand interessiert ist. Im Sinn einer aktiven Projektführung wird der am Ende einer Phase erreichte Stand mit dem geplanten verglichen. Zu große Abweichungen geben (hoffentlich) Anlaß zu Korrekturmaßnahmen im Projekt. Die radikalste kann der Abbruch des Projekts sein.
Das Ende einer Phase ist ein *Meilenstein*, das heißt ein ausgezeichneter Zeitpunkt, an dem geprüfte und bewertete Resultate vorliegen. In den Normen findet man häufig ein *Review*, an dem die Resultate erwahrt werden. Dieses Freigabe-Review ist eine informelle Sitzung, an der geprüft wird, ob die Resultate mit den zugehörigen Prüfberichten (Review-, Testprotokolle) vorliegen. Es ist nicht die Meinung, daß am letzten Tag der Phase mit einem Kraftakt alle Resultate wie Dokumente oder Programm-Module einem technischen Review unterzogen werden.
Die *Tätigkeiten* bei der Softwareentwicklung sind die bekannten: Analysieren, Spezifizieren, Entwerfen, Codieren, Testen usw. Wer schon in einem Softwareprojekt mitgearbeitet hat, weiß, daß diese Tätigkeiten mehr oder weniger überlappend ausgeführt werden (Bild 2).
Aus der Darstellung in Bild 2 ist offensichtlich, daß pro Phase verschiedene Tätigkeiten ausgeführt werden. Von diesen Tätigkeiten dominiert meistens eine. Üblicherweise gibt sie der Phase den Namen. Damit ist die Wurzel des Mißverständnisses freigelegt. Die Phase ist durch die Resultate der Phase definiert. Dabei gehört zu den Resultaten einer Phase auch die Überarbeitung eines Resultats einer früheren Phase. In Bild 2 ist dargestellt, wie die verschiedenen Versionen einiger Resultate in verschiedenen Phasen anfallen können. Die Überarbeitung kann durch Fehler bei der vorangehenden Arbeit, aber auch durch neue Erkenntnisse durch die detaillierte Arbeit an der Lösung bedingt sein.
Die Überlappung der Tätigkeiten ist um so stärker, je kürzer die Terminvorgabe ist. Der Preis für den hoffentlich kürzeren Termin ist das Risiko, daß eine im Detail bereits ausgeführte Arbeit weggeworfen werden muß, da in einem über-

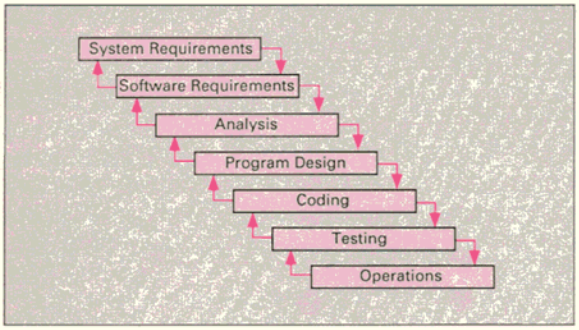


Bild 3. Das Wasserfallmodell nach Royce.

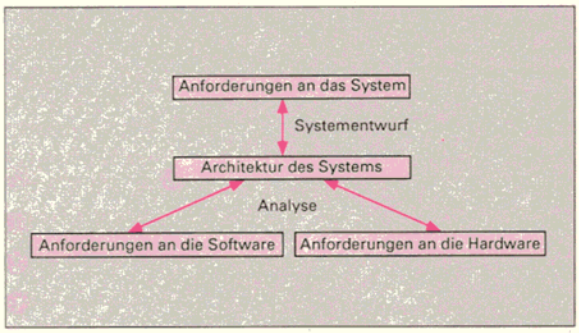


Bild 4. System- und Softwareaspekte.

gordneten Arbeitsschritt neue Vorgaben resultieren. Zum Beispiel kann ein Teil des bereits ausgeführten Entwurfs durch eine später eingebrachte Anforderung überholt werden.

Bild 3 zeigt die erste von vielen Versionen des Wasserfallmodells nach ROYCE [1]. Für dieses Wasserfallmodell gibt es drei verschiedene Interpretationen für die Bedeutung seiner Kästchen: erstens Tätigkeiten, zweitens organisatorische Einheiten, drittens Phasen.

Die beiden ersten Interpretationen verursachen die geringsten Schwierigkeiten, sie nützen aber auch nichts. Am besten stützt man sich auf die weiter oben bereits erwähnte Definition für die Phase und vergißt den Wasserfall.

Die nachstehenden Aussagen sind Folgerungen aus den bisherigen Ausführungen. Wenn ein Phasenplan Widersprüche zu diesen Aussagen enthält, sollte man nochmals über die Bücher gehen.

- Eine Phase ist durch Resultate definiert.
- Die Bewertung der erreichten Resultate anhand der geplanten Resultate kann Anlaß geben zu Korrekturmaßnahmen im Projekt.
- In einer Phase werden normalerweise verschiedenste Tätigkeiten ausgeführt.
- Phasen können mehr oder weniger überlappen.
- Die Wahl der Phasen hängt von der Art der Aufgabenstellung beziehungsweise dem zu entwickelnden Objekt ab.
- Die Tätigkeiten einer Phase werden nach ihrem Beitrag zur Risikominderung ausgewählt.
- Phasen nützen dem Entwickler nichts.

Anzahl und Art der gewählten Phasen hängen vom Problem und vom Ansatz seiner Behandlung ab. Ein und der gleiche Satz von Phasen wird kaum sowohl für die Entwicklung von Gerätesoftware als auch von Datenbankapplikationen angewendet.

Schnittstelle Hardware–Software

Bei der Schnittstelle Hardware–Software handelt es sich um ein Problem, das in der Struktur der Lösung liegt. Bei der Entwicklung von Hardware/Software im Umfeld von Rechnersystemen handelt es sich selten um eine reine Hardware- oder um eine reine Softwareentwicklung. Meistens soll ein System oder ein Gerät mit Hardware- und Softwarekomponenten entwickelt werden. Bild 4 zeigt den Zusammenhang zwischen Anforderungen an das System und an die Software.

Beide Anforderungen sind verknüpft durch die Architektur des Systems, das heißt durch die Lösungsstruktur auf der obersten Ebene. Die Unterscheidung dieser Elemente ist zur Vermeidung von Mißverständnissen wesentlich.

Dieser Punkt scheint trivial, der Raum für die Beschreibung im Verhältnis zur Diskussion des Phasenverständnisses untergeordnet. Die Erfahrung bei der Beratung in Entwicklungen zeigt jedoch, daß gerade in diesem Punkt immer wieder Unklarheiten bestehen, die für die Beteiligten zu nicht zuordbaren Problemen führen. Wie soll zum Beispiel ohne diese Differenzierung ein Entwickler den Auftrag verstehen, die Anforderungen für die Software zu spezifizieren? Er wird Schwierigkeiten begegnen und diese bei seiner Spezifikationsmethode und/oder seinem Werkzeug suchen.

Methoden

Die Methoden sind verstärkt mit den Werkzeugen zur (grafischen) Unterstützung und mit den objektorientierten Ansätzen ins Gespräch gekommen. Ziel von Methoden ist es, Anleitung für ein planmäßigs Vorgehen bei bestimmten Aufgaben zu geben. Damit sollen auch Nichtgenies in die Lage versetzt werden, Aufgaben zielgerichtet zu lösen und ein brauchbares Ergebnis zu liefern.

Es gibt Methoden für verschiedene Tätigkeiten der Entwicklung: Konfigurationsverwaltung, Spezifizieren von Anforderungen, Entwerfen, Programmieren, Prüfen.

Im Rahmen dieses Beitrags kann nicht auf die Methoden im einzelnen eingegangen werden. In der Literatur findet man gute Beschreibungen der einzelnen Methoden. Zum Beispiel bietet PRESSMAN einen umfassenden Überblick [2]. Hier sollen nur einige Bemerkungen zum Umgang mit Methoden gemacht werden.

Wichtig ist die richtige Einstellung zu Methoden: Methoden sind kein Selbstzweck, sondern Hilfe für den Entwickler im Rahmen des gewählten Entwicklungsablaufs. Es muß erlaubt sein, Methoden intelligent zu interpretieren beziehungsweise an die konkrete Aufgabenstellung anzupassen. Sonst besteht die Gefahr, daß die Methode für das Mißlingen des Projekts mitverantwortlich ist, daß beispielsweise Laufmeter von Papier erzeugt werden und das Projekt trotzdem (oder gerade deswegen) mißlingt (DEMARCO [3]).

Daraus kann jedoch kein Freibrief für den einzelnen Entwickler abgeleitet werden. Es kann nicht jeder tun und lassen, was er will. In einem Projekt muß es aber die Möglichkeit geben, Methoden den Bedürfnissen anzupassen; selbstverständlich sind diese Anpassungen zu dokumentieren und, falls notwendig, durch das Qualitätswesen freizugeben.

Der schwierigste Punkt ist die Auswahl der geeigneten Methode für eine bestimmte Entwicklungsumgebung beziehungsweise für ein konkretes Projekt. Die Auswahl setzt einerseits die Kenntnis verschiedener Möglichkeiten voraus, anderseits die Kenntnis dessen, was man eigentlich erreichen will. Nach der

am häufigsten in Zeitschriften genannten Methode zu springen heißt nur, daß jemand ein großes Werbebudget hat und etwas verdienen will. Manche Methoden sind so günstig, daß niemand dafür wirbt, weil es keine Werkzeuge zu ihrer Anwendung braucht. Also inseriert auch niemand. Konkret, was wissen Sie von Review-Technik?
Die Auswahl und die Einführung von Methoden bedeutet einen größeren Aufwand mit langfristigen Konsequenzen. Methoden bedingen einen Einführungsaufwand in Form von Schulung und Lehrgeld für Fehler. Diesen Aufwand kann man nicht alle drei Jahre abschreiben und dann auf einen anderen Modetrend umschwenken. Zum Glück beschränken sich die Moden ja meistens auf ein neues Gewand, in dem sich bei genauerer Betrachtung Altbekanntes wiederfindet. Das Umfeld und das Vorgehen in einem Projekt zur Auswahl und Einführung von Methoden beschreibt PRESSMAN in [4] ausführlich.
Um das Ziel des Einsatzes von Methoden zu erreichen, eine Tätigkeit ihrer Zufälligkeit zu berauben, muß eine Methode nach FRÜHAUF [5]:

- einen erklärten Zweck haben
- auf einem Konzept basieren
- ein Arbeitsergebnis bestimmter Qualität anstreben
- die Form für das Ergebnis festlegen
- eine Vorschrift für das Vorgehen zum Erstellen des Resultats beinhalten

Gemessen an diesen Kriterien, müssen manche sogenannten Methoden zu Notationen degradiert werden.
Eng verknüpft mit dem Thema Methoden sind *Richtlinien.* In Richtlinien werden die Methoden festgeschrieben. Dabei sind aber unbedingt folgende Punkte zu beachten:

Weniger ist mehr

Nicht die Dicke der Richtliniensammlung entscheidet über die Wirksamkeit; auch wenn man über Unwesentliches leichter einen Konsens erreicht, sollte man überlegen, ob es notwendig ist, alle Details zu regeln.

Nur Erprobtes in Richtlinien gießen

Noch nicht ausgegorene Ideen können in unverbindlicheren Checklisten auf ihre Wirksamkeit erprobt werden.

Nur überprüfbare und tatsächlich überprüfte Richtlinien werden eingehalten

In vielen Entwicklungsumgebungen finden sich dicke, oft verstaubte Ordner mit Richtlinien zur Programmierung. Geprüft werden die Programme nur mit Tests, gelesen werden sie höchstens vom Autor!

Richtlinien sind verbindlich

Richtlinien sind generell einzuhalten. Wenn sie nicht einhaltbar sind, sind sie zu ändern oder für ein Projekt explizit außer Kraft zu setzen.

Richtlinien beschreiben den Konsens der Entwickler

Es lohnt sich, vor dem Festschreiben von Richtlinien ein gemeinsames Verständnis zu erarbeiten, zum Beispiel in einem Workshop.

Richtlinien sind kein Ersatz für Lehrbücher

Wenn Beschreibungen von Methoden notwendig sind, ist es zweckmäßiger, auf Literatur zurückzugreifen.

Werkzeuge

Im Bereich der Programmierung sind Werkzeuge ein schon lange nicht mehr wegzudenkender Bestandteil für den Entwickler. Wer möchte die Editoren, Compiler, Linker oder Debugger missen? Bei der heutigen CASE-Diskussion stehen zum einen Werkzeuge für die Spezifikation der Anforderungen und den Entwurf im Vordergrund, zum anderen Werkzeuge, die die verschiedenen Zwischenresultate über die verschiedenen Phasen hinweg miteinander verknüpfen.
Die Bemerkungen zu den Werkzeugen sollen hier für fünf Anwendungsgruppen getrennt dargelegt werden.

Spezifikation von Anforderungen und Entwurf

Der Nutzen der Werkzeuge dieser Gruppe liegt primär in einer verbesserten Dokumentation der Arbeitsresultate. Die zugrunde liegenden Methoden helfen die Aufgaben vollständiger, systematischer zu bearbeiten. Die Werkzeuge erlauben die ansprechende Aufbereitung der Resultate und unterstützen deren Überarbeitung. Die von den Methoden verwendeten grafischen Notationen sind ohne die Unterstützung der Werkzeuge nicht durchzuhalten. Der Elan geht bei Handbetrieb spätestens bei der dritten Überarbeitung der Grafiken verloren.
Der Zwang zur Definition von Begriffen und anderen verwendeten Elementen in einem Datenlexikon trägt zur Aufdeckung von Unklarheiten und Mißverständnissen bei. Die Disziplin in der Durchführung dieses Arbeitsschritts läßt jedoch häufig zu wünschen übrig. Bilder zu erstellen und zu präsentieren ist spektakulärer.
Eine Warnung gegenüber den Werkzeugen ist angebracht. Es handelt sich um mächtige Dokumentationswerkzeuge, nicht aber um Wunderwaffen [6]. Ein schlechter Analytiker wird durch ein Werkzeug kein guter Analytiker, genausowenig wie ein schlechter Holzfäller durch eine Motorsäge ein guter Holzfäller wird. Außerdem ist die Methodentreue der Werkzeuge beschränkt. Je mehr Methoden und Varianten unterstützt werden, desto größer ist der Markt für das Werkzeug, und um so unspezifischer ist die Unterstützung für den Entwickler.
Gerade für diese Gruppe von Werkzeugen ist es wichtig, daß man zuerst über Methoden und dann über Werkzeuge redet. Dabei empfiehlt sich die Konzentration auf den Entwurf, da es hier mehr Varianten beziehungsweise Kombinationen von Methoden braucht und viele Werkzeuge (auch sehr teure) gerade hier am wenigsten bieten.

Implementation

Im Bereich der eigentlichen Programmierung tragen die Werkzeuge am meisten zu einem automatisierten Ablauf bei. Übersetzer, Binder usw. haben wesentlich zur Reduktion an sich einfacher, aber fehleranfälliger Tätigkeiten der Programmierer beigetragen.

Test und Metrik

In diesem Bereich gibt es Werkzeuge zum Ermitteln der Komplexität von Programm-Modulen, zum Erstellen von

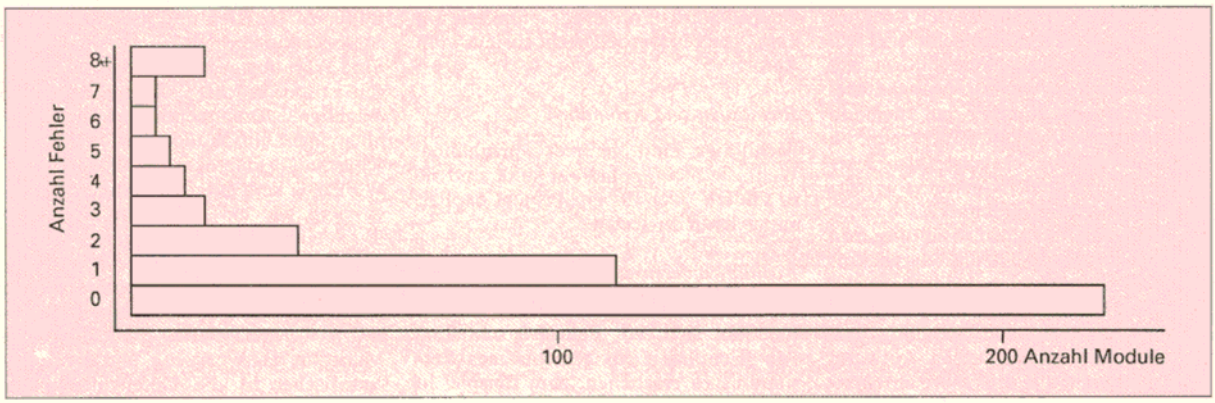


Bild 5. Fehlerverteilung auf Programm-Module.

Testfällen, zum überwachten Ablauf der Testfälle und zum Prüfen der Resultate. Diese Werkzeuge sind aber noch nicht sehr weit verbreitet. Zum einen sind viele Entwicklungsumgebungen noch nicht auf der Stufe, auf der Metriken eine Rolle spielen, zum anderen sind die Testwerkzeuge je nach Anwendungsbereich noch nicht sehr mächtig oder verlangen einen großen Aufwand zur Vorbereitung der Tests.

Debugger

Im Bereich Fehlerlokalisierung und -behebung gilt das für den Bereich Implementation Gesagte. Da Programmierer viel Zeit mit der Behebung von Fehlern verbringen, haben sie bald einmal begonnen, diese Arbeit durch Debugger zu unterstützen. Die Schnittstelle zu diesen Werkzeugen auf dem Niveau der verwendeten höheren Programmiersprache beziehungsweise Designdarstellung kommt den Bedürfnissen der Entwickler sehr entgegen.

Konfigurationsverwaltung

Die Konfigurationsverwaltung ist vielleicht *die* grundlegende, alle Tätigkeiten umfassende Klammer in der Softwareentwicklung. Sie verfolgt zwei Ziele:

die Definition eines Originals der Software

Alle Kopien eines Dokuments oder Programms auf einem Rechner sehen gleich aus. Nur ein Verfahren und die Unterstützung durch Werkzeuge erlauben die Identifikation eines künstlichen Originals;

die Verwaltung der vielen Komponenten unserer Software

Im allgemeinen gilt es viele Komponenten und von jeder Komponente verschiedenste Versionen zu verwalten. Ohne Unterstützung durch ein geeignetes Werkzeug geht der Überblick über die sinnvollen Kombinationen schnell verloren.
Aus dem Blickwinkel Qualitätssicherung müssen alle Prüfmaßnahmen ohne eine geeignete Konfigurationsverwaltung in Frage gestellt werden. Ohne Konfigurationsverwaltung ist es zum Beispiel nicht möglich sicherzustellen, daß ein ausgeliefertes Programm auch tatsächlich die getestete Version ist.

Der Mitarbeiter

Die wichtigste Komponente in der Entwicklung von Software bleibt der Mensch (nicht der Computer!). Das Können der Entwickler ist sehr breit gestreut; ein Faktor fünf bei der Bewertung der Produktivität von Entwicklern ist allgemein akzeptiert. Gilt dies auch für andere Ingenieurdisziplinen?
Sicher nicht. Insbesondere das Bewußtsein für Qualität und für Konsequenzen, die schlechte Arbeit hat, ist nicht weit verbreitet. Dafür hört man häufig Argumente wie dieses: Warum soll man die teure Zeit der Entwickler für sorgfältiges Arbeiten verwenden, wenn doch anschließend Programme wie statische Programmanalyse-Tools und Übersetzer die Fehler im Programmtext viel billiger und schneller aufzeigen. Die Fehler, die in diesem Schritt nicht gefunden werden, kommen ja beim Testen zum Vorschein. Und außerdem ist bekannt, daß es keine fehlerfreien Programme gibt.
Die gerade beschriebene Haltung ist typisch für eine große Mehrheit der Entwickler. Noch schlimmer ist aber die Tatsache, daß diese Haltung vom Management nicht erkannt oder gar akzeptiert wird.
Hier tut Aufklärung not. Gerade weil es (zu viele) Fehler in Programmen gibt, gilt es Anstrengungen zu unternehmen, keine (unnötigen) Fehler zu machen. Wenn man die Fehlerverteilung in Programmen analysiert [7], sieht man, daß sich die Fehler im Sinn einer Pareto-Verteilung in einigen Modulen häufen, während andere Module fehlerfrei sind (Bild 5).
Ein anderer Zusammenhang, der immer wieder Erstaunen hervorruft, zeigt unter anderem DeMarco in [3]. In mehreren Projekten wurde beobachtet, daß die Häufigkeit von Syntaxfehlern mit den logischen Fehlern in einem Modul zusammenhängt. Das heißt, wer es mit der Programmiersprache nicht so genau nimmt, nimmt es offenbar auch mit der Logik des Programms nicht so genau.
Das Argument vom Sparen von teurer Entwicklerzeit durch den (häufigen) Einsatz von Compilern ist offensicht-

lich nur die halbe Wahrheit: Der Aufwand für Test- und Fehlerbehebung wird später noch anfallen.
Einen konstruktiven Beitrag kann hier entsprechende Aufklärung bei den Entwicklern und beim Management leisten. Zu den Themen gehören die Zusammenhänge des Entwicklungsablaufs, beobachtete Gesetzmäßigkeiten und die möglichen Maßnahmen sowie deren realistischer Nutzen.
Das Qualitätsziel der Führung färbt auf die Entwickler ab. Wenn Qualität etwas ist, um das man sich auch noch kümmert, wann man gerade Zeit hat, dann ist klar, was in einem Softwareprojekt passiert. Wenn unter Termindruck Reviews oder Tests gestrichen oder abgekürzt werden, ist jedem Entwickler klar, was die früheren Appelle wert waren.

Effizienz und Qualität: kein Widerspruch

Das Potential zur Verbesserung der Qualität des Entwicklungsprozesses ist groß. Die Erfahrung zeigt, daß die Effizienz der Entwicklung und die Qualität des resultierenden Produkts, der Software, nicht im Widerspruch zueinander stehen.
Für die erfolgreiche Realisierung braucht es die Zusammenarbeit des Linienmanagements und des Qualitätswesens sowie ein Konzept für das Vorgehen. Sicher falsch ist es, das Pferd vom Schwanz her aufzuzäumen und mit einer Investition in (CASE-)Werkzeuge im Sinn eines Rauchopfers die Götter gnädig stimmen zu wollen, sprich: von den Sünden der Vergangenheit abzulenken.
Werkzeuge sind ein wesentliches Element der Softwareentwicklung. Ein Ausbau des Einsatzes ist darum sinnvoll, wenn er mit anderen Maßnahmen abgestimmt wird.
Abläufe, Methoden und Werkzeuge müssen zusammenspielen. Abläufe und Methoden ändern langsamer als Werkzeuge, und ihre Einführung erfordert höhere Investitionen. Daher ist deren Festlegung vor der Investition in neue Arten von Werkzeugen von größter Bedeutung.
Auch im Zeitalter von CASE bestimmen immer noch drei Faktoren die Produktivität:

- die Qualifikation der Mitarbeiter
- die Effizienz der Entwicklungsschritte
- der Entwicklungsumfang

Daher muß man Maßnahmen zur Verbesserung dieser Faktoren ins Zentrum der Betrachtung stellen.

Literatur

1. Royce W. W.: Managing the Development of Large Software Systems: Concepts and Techniques, Proceedings WESCON, August 1970.
2. Pressman R. S.: Software Engineering, A Practioner's Approach, McGraw Hill, International Edition, 1987.
3. DeMarco T., Lister T.: Peopleware, Productive Projects and Teams, Dorset House, New York, 1987.
4. Pressman R. S.: Making Software Engineering Happen, Prentice Hall, Englewood Cliffs/N.J., 1988.
5. Frühauf K.: Konstruktive Maßnahmen in der Software-Qualitätssicherung, in Tomica (1987), pp. 43–60.
6. Sandmayr H.: Nutzen und Grenzen von CASE, Bulletin SEV/VSE 80 (1989), Heft 17, pp. 1063–1067.
7. Endres A.: An Analysis of Errors and Their Causes in System Programs, International Conference on Reliable Software, IEEE, 1975.
8. Frühauf K., Ludewig J., Sandmayr H.: Software-Projektmanagement und -Qualitätssicherung, B. G. Teubner, Stuttgart, 1988.
9. McClure C.: CASE is Software Automation, Prentice Hall, Englewood Cliffs/N.J., 1989.
10. Tomica K. (Hrsg.): Software-Qualitätssicherung 1987, Tagungsband, SAQ, Bern, 1987.