12. Produktivitätsfaktoren

12.1 Werkzeuge

12.1.1 Allgemeines

Im Informatik-Sprachgebrauch sind *Werkzeuge* rechnergestützte Hilfsmittel für die Entwicklung von Software. Mit dem Einsatz von Werkzeugen werden primär folgende Ziele verfolgt:

- Entlastung der Entwickler von Routinearbeiten
- Bearbeitung der Sprachen (graphisch oder textuell), mittels derer die Entwickler ihre Probleme und Lösungen formulieren
- Unterstützung des Einsatzes von Entwicklungsmethoden
- Vereinfachung von Änderungen.

Werkzeuge sind jedoch – entgegen einem weit verbreiteten Glauben – keine Wunderwaffen:

- sie steigern die Produktivität nicht um (dezimale) Größenordnungen (es sei denn, man hätte vorher die Bits noch einzeln ins Silizium gemeißelt)
- sie ersetzen das eigene Denken und sorgfältige Arbeiten nicht (d.h. sie sind *nicht* intelligent, weder künstlich noch natürlich)
- sie führen zwar in den meisten Fällen zu verbesserter Qualität der Produkte, machen aber das Qualitätsmanagement (Reviews, Tests, etc.) nicht überflüssig.

Ein isolierter Einsatz von Werkzeugen bringt in der Regel wenig oder kann sogar kontraproduktiv sein. Werkzeuge müssen als eine Komponente in einer Gesamtstrategie für rechnergestütztes Software Engineering aufgefasst werden (Bild 12.1; Glinz 1990). Die Sache (Software Engineering) und das Mittel (computergestützt) dürfen in ihrer Wichtigkeit nicht miteinander verwechselt werden.

Seit etwa 1986 werden Werkzeuge (vor allem solche für Spezifikation und Entwurf) von Software häufig mit dem Schlagwort *CASE* (computer aided software engineering) bezeichnet.

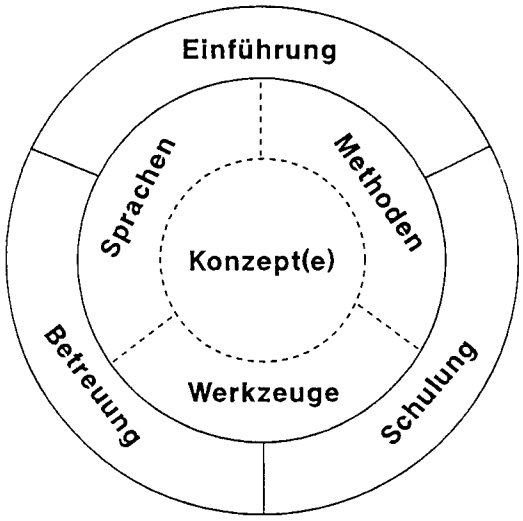

BILD 12.1. Elemente einer Strategie für computergestütztes Software Engineering

Produktivitätssteigerungen von 20-50% sind durch geeigneten Werkzeugeinsatz durchaus erreichbar. Dieser Produktivitätsgewinn stellt sich jedoch nicht sofort ein. Wird ein Werkzeug neu eingeführt, so müssen die Mitarbeiter intensiv geschult werden und müssen sich an den Umgang mit dem Werkzeug gewöhnen. Als Konsequenz sinkt die Produktivität erst einmal, anstatt zu steigen. Noch schlimmer ist die Situation bei Werkzeugen für Spezifikation und Entwurf von Software. Da beide Bereiche bei manueller Arbeit traditionell vernachlässigt werden, hat der Werkzeugeinsatz zur Folge, dass die Spezifikation und Konzipierung deutlich länger dauern als vorher. Der Gewinn macht sich erst in einer verkürzten Realisierung und noch später in niedrigeren Pflegekosten bemerkbar.

Werkzeugeinsatz ist daher keine Maßnahme, die sofort wirksam wird, sondern eine *Investition*, die sich erst mittelfristig (dann aber umso deutlicher) auszahlt.

Nicht zu vernachlässigen ist neben dem Produktivitätsgewinn ein u.U. massiver Qualitätsgewinn bei der mit Werkzeughilfe entwickelten Software.

12.1.2 Software-Produktionsumgebungen

Die vollständige Software Engineering-Umgebung, welche alle Aktivitäten von der Formulierung der ersten Anforderungen bis zur Systemabnahme durchgehend und konsistent unterstützt, ist zwar schon vielfach postuliert und skizziert worden, aber es gibt sie immer noch nicht zu kaufen.

Die heutige Situation ist geprägt von einer Vielfalt von Systemen mit unterschiedlichen Schwerpunkten, welche jeweils einen Teil der Software Engineering-Aktivitäten gut, einen Teil halbwegs und einen Teil überhaupt nicht unterstützen.

12.1.3 Klassifizierung von Werkzeugen

Die heute verfügbaren Werkzeuge lassen sich grob in folgende Gruppen klassifizieren:

Editoren

Editoren rationalisieren die Schreib- und Zeichenarbeiten. Strukturediteren und syntaxgesteuerte Editoren beschleunigen die Dokumentationserstellung und die Programmierung.

Spezifikations- und Entwurfssysteme

Diese Systeme unterstützen das Erstellen von Anforderungen und von Konzepten (Architekturentwürfen) für Systeme vorzugsweise durch graphische Darstellungen und durch geeignete Speicherung der Information in einer Datenbank.

Programmentwurfssysteme

In diese Kategorie fallen Werkzeuge zur Bearbeitung von Struktogrammen, Pseudocode, Jackson-Diagrammen, Programmablaufplänen, etc.

Compiler, Browser und Programmierumgebungen

Auch die gewöhnlichen Compiler für Programmiersprachen sind Werkzeuge. Programmierumgebungen vereinigen Editoren, Compiler, Browser (Software zum Durchstöbern der Arbeitsumgebung und von Software-Bibliotheken), evtl. Interpreter, Laufzeitumgebung, Debugger und evtl. ein Konfigurationsverwaltungssystem zu einem Satz aufeinander abgestimmter Werkzeuge für die Programmierung.

Programmgeneratoren

Programmgeneratoren sind Übersetzer, welche aus einer Hochsprache (v.a. Datenbanksprachen, sogen. Sprachen der 4. Generation) Programme in einer niedrigeren Sprache (v.a. COBOL und Job-Steuersprachen) erzeugen.

Mess- und Testwerkzeuge

Werkzeuge dieser Kategorie dienen dazu, interessierende Merkmale eines Programms (zum Beispiel Größe oder Komplexitätsattribute) zu messen und zu analysieren, Programme für die Ausführung und Auswertung von Tests zu instrumentieren, Testfälle, Testdaten und Testumgebungen zu generieren sowie Testergebnisse auszuwerten.

Konfigurationsverwaltungssysteme

Konfigurationsverwaltungssysteme vereinfachen das Verwalten verschiedener Versionen von Programmen und Dokumenten und ermöglichen das Generieren von Konfigurationen, d.h. das automatische Zusammensetzen eines Systems aus einer Reihe von Programmen und Dokumenten, unter Umständen mit entsprechender Vorverarbeitung wie Compilieren, Binden, Sortieren, etc.

12.1.4 Planung des Werkzeugeinsatzes

Wenn man Werkzeuge für die Software-Entwicklung erfolgreich einführen will, muss man sich zunächst über folgende Fragen im Klaren sein:

- Welche Aktivitäten sollen die Werkzeuge unterstützen bzw. automatisieren?
- Wie wirtschaftlich ist der Werkzeugeinsatz? (Ein syntaxgesteuerter Editor beispielsweise, welcher den Aufwand für die Codierung um 25% vermindert, bringt nur 5% Verminderung des Gesamt-Projektaufwands, wenn man 20% Aufwandsanteil für die Codierung veranschlagt)
- Welche Entwicklungskonzepte, und damit verbunden, welche Methoden und Sprachen sollen eingesetzt werden?
- Ist die Schulung geregelt (Freistellung der Mitarbeiter, Finanzierung, Verfügbarkeit von Kursen)?
- Wie sieht die Einführungsstrategie aus?
- Ist die Betreuung der Werkzeugverwender sichergestellt?

12.2 Mehrfachverwendung von Software

Die Bedeutung von Mehrfachverwendung für die Produktivität wurde in Kapitel 3.4 diskutiert.

12.3 Die Rolle der Menschen

Software wird von Menschen gemacht. Die Software-Leute sind daher neben der Software-Mehrfachverwendung der entscheidende Produktivitätsfaktor in der Software-Entwicklung.

Aspekte der Psychologie und der Menschenführung spielen somit eine wichtige Rolle in der Software-Entwicklung. Wesentliche Gedanken dazu enthalten die Bücher von Brooks (1995), DeMarco und Lister (1991) und Weinberg (1971). In diesem Kapitel werden nur wenige Grundsätze und Regeln beschrieben.

12.3.1 Regeln über Menschen in der Software-Entwicklung

Produktivität

Die Schwankungsbreite der individuellen Produktivität von Software-Entwicklern ist sehr groß. Ein von Sackman (1968) durchgeführtes Experiment ergab Schwankungsbreiten in der Produktivität von ca. 20:1. Diese Daten sind zwar schon recht alt, aber es gibt keinen Grund zur Annahme, dass sich seither etwas Grundlegendes verändert hat. Boehm (1987) gibt eine Schwankungsbreite von rund 4 an, wobei er allerdings nicht die besten und die schlechtesten Entwickler vergleicht, sondern Gruppen, die einerseits unter 15% und andererseits über 90% auf einer relativen Fähigkeitsskala liegen.

Disponibilität

Personal ist kein beliebig disponierbarer Produktionsfaktor. Drei Regeln sind zu beachten:

- Personalbestände in einem Projekt können nur langsam auf- oder abgebaut werden. Es ist nicht ohne Schaden möglich, kurzfristig massive Veränderungen im Personalbestand eines Projekts vorzunehmen.
- Zu einem gegebenen Entwicklungsaufwand gibt es eine optimale Gruppengröße für die Bearbeitung und damit eine optimale Durchlaufzeit. Die Rechnung, die Durchlaufzeit durch den Einsatz von mehr Entwicklern zu verkürzen, geht nur in sehr engen Grenzen auf.
- Das Aufstocken des Personalbestands in einem verspäteten Projekt führt zu noch mehr Verspätung (Gesetz von Brooks; Brooks 1995).

Arbeitsverteilung

Software-Entwickler schreiben nicht nur Programme. Eine Studie der Bell Labs ergab, dass der im Hinblick auf Codeerzeugung produktive Anteil an der Arbeit eines Programmierers nur 13% beträgt (Bild 10.1).

Programme schreiben	13%	
Programme und Handbücher lesen	16%	
Arbeitsbezogene Kommunikation	32%	(!)
Persönliches	13%	
Ausbildung	6%	
Post	5%	
Diverses	15%	Studie der Bell Labs, 1964, zitiert aus Fairley (1985)

BILD 12.2. Verteilung der Arbeitszeit eines Programmierers

Gruppengröße

Jede Entwicklergruppe benötigt einen Teil ihrer Zeit für Organisation und Kommunikation innerhalb der Gruppe. Mit zunehmender Personenzahl nimmt dieser nicht produktive Aufwand überproportional zu (vgl. dazu Bild 1.6).

12.3.2 Emotionales und Rationales in der Software-Entwicklung

Emotionale Einstellungen der an Software-Entwicklung Beteiligten haben erhebliche Einflüsse auf das, was in einer Entwicklungsabteilung tatsächlich geschieht. Der Einfluss des Emotionalen ist umso stärker, je weniger er den Beteiligten bewusst ist. (Glinz 1988).

12.3.3 Arbeitsumgebung

Da Software von Menschen gemacht wird, haben Ausbildung und Arbeitsumgebung dieser Menschen einen erheblichen Einfluss auf den Produktionsprozess. Im Buch von DeMarco und Lister (1991) werden verschiedene Aspekte dieses Problems betrachtet.

Aufgaben

12.1 Eine Software-Entwicklungsabteilung schreibt rote Zahlen. Anstatt ein besonders sparsames Budget vorzulegen, beantragt die Abteilungsleiterin zum Erstaunen der Geschäftsleitung eine Investition von 100.000 Franken zur Modernisierung der Arbeitsplätze der Entwicklerinnen und Entwickler sowie für Ummöblierungen und Gardinen(!).
Was könnte sich die Frau dabei gedacht haben?

12.2 Ein Unternehmen will die Produktivität seiner Software-Entwicklungsabteilung verbessern. Beschreiben Sie, welche Rolle Werkzeuge dabei spielen können.

Zitierte und weiterführende Literatur

Boehm, B. (1987). Improving Software Productivity. *IEEE Computer* **20**, 9 (Sept. 1987). 43-57.

Brooks, F.P. (1995). *The Mythical Man Month. Essays on Software Engineering*. Anniversary Edition Reading, Mass., etc.: Addison-Wesley. (Neuausgabe des Originals von 1975)

DeMarco, T., T. Lister (1991). *Wien wartet auf Dich! Der Faktor Mensch im DV-Management*. München-Wien: Hanser.

Fairley, R. (1985). *Software Engineering Concepts*. New York, etc.: McGrawHill.

Glinz, M. (1988). Emotionales und Rationales im industriellen Software Engineering. *Technische Rundschau* 20/88, 78-81.

Glinz, M. (1990) Warte nicht auf besere Zeiten - Methoden und Werkzeuge in der Software-Entwicklung. *Technische Rundschau* 35/90, 70-75

IEEE (1992). Themenheft Tools Assesment. *IEEE Software* **9**, 3 (May 1992).

Ludewig, J. (1991). Software Engineering und CASE - Begriffsklärung und Standortbestimmung. *it (Informationstechnik)* **33**, 3. 112-120.

Sackman, H. et al. (1968). Exploratory Experimental Studies Comparing Online and Offline Programming Performance. *Communications of the ACM* **11**, 1 (Jan. 1968).

Weinberg, G.M. (1971). *The Psychology of Computer Programming*. New York: Van Nostrand Reinhold.