



Universität
Zürich^{UZH}

Institut für Informatik

Martin Glinz Harald Gall
Software Engineering

Kapitel 19

**Die Rolle der Menschen im
Software Engineering**

19.1 Grundlagen und Gesetzmäßigkeiten

19.2 Software-Psychologie

19.3 Software-Ingenieure im Unternehmen

19.4 Zwei Welten?

19.5 Software-Management und -Kultur

Die Rolle der Menschen im Software Engineering

Software wird von Menschen gemacht.

- ⇒ Die Software-Leute sind ein entscheidender Produktivitätsfaktor
- Können
 - Motivation
 - Arbeitsumfeld

Gesetzmäßigkeiten über Software-Leute – 1

- **Produktivität**
 - Enorme Schwankungsbreiten, bis 20:1
 - Selbst bei Gruppen noch Schwankungsbreiten bis 4:1

- **Disponibilität**
 - Personalbestände nur langsam auf- und abbaubar
 - Zu jedem Aufwand eine optimale Personenzahl
 - Das Aufstocken des Personalbestands in einem verspäteten Projekt führt zu noch mehr Verspätung (Gesetz von Brooks)

Gesetzmäßigkeiten über Software-Leute – 2

○ Arbeitsverteilung

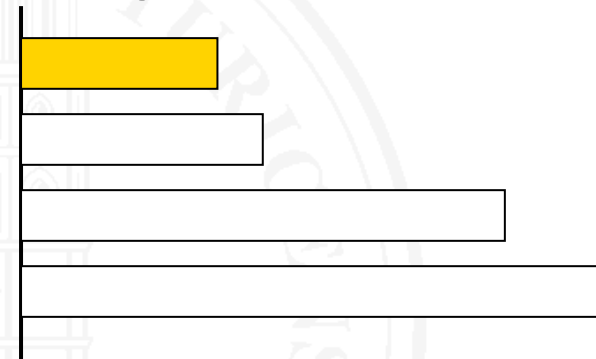
- Programmierer schreiben nicht nur Programme

Programme schreiben

Programme und Dokumente lesen

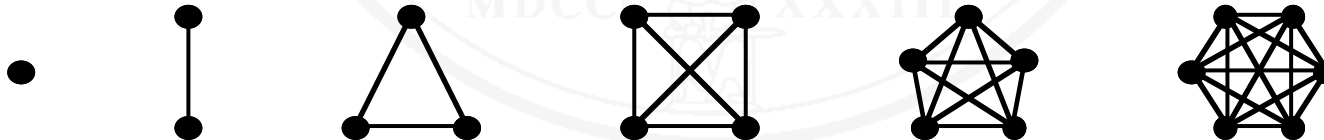
Arbeitsbezogene Kommunikation

Anderes



○ Gruppengröße

- Nicht produktiver Aufwand wächst überproportional mit der Gruppengröße



19.1 Grundlagen und Gesetzmäßigkeiten

19.2 Software-Psychologie

19.3 Software-Ingenieure im Unternehmen

19.4 Zwei Welten?

19.5 Software-Management und -Kultur

Emotionales und Rationales

- In der Praxis des Software Engineerings werden viele Maßnahmen nicht ergriffen, deren Nutzen und Wirksamkeit längst erwiesen ist
- An mangelndem Wissen allein kann das nicht liegen
- Diskrepanz zwischen Wissen und Handeln?
- Faktisch wohl meist eine Diskrepanz zwischen

rationaler Erkenntnis

und emotionaler Befindlichkeit

Emotional vs. Rational

- **Größe** und **Komplexität**
 - Denken im Kleinen vs. Arbeiten im Großen
- **Ego** der Entwickler
 - Individuum vs. Kollektiv
 - Kreativität vs. Systematik
 - Bedrohung durch Überprüfung vs. Qualität durch Prüfung
- **Lust** und **Frust**
 - Programmieren ist selbstmotiviert, dokumentieren nicht
 - Debuggen ist selbstmotiviert, testen nicht
- **Zeiteffekte**
 - Trägheit vs. Fortschritt
 - kurzfristiges Denken vs. langfristige Wirkung von SE-Maßnahmen

Motivation

- Selbstmotivation ausnutzen: **Win-Win** Situationen schaffen
- Explizite **Anreize** für nicht selbstmotivierte Arbeiten
- “**Egoless programming**” ohne Ego-Verlust der Entwickler ermöglichen
- Gute **Arbeitsbedingungen** schaffen
- Leute **ernst nehmen** und **fördern**
- **Selbstwertgefühl** steigern
- Software Ingenieure sollen **stolz** sein können auf ihre Produkte

19.1 Grundlagen und Gesetzmäßigkeiten

19.2 Software-Psychologie

19.3 Software-Ingenieure im Unternehmen

19.4 Zwei Welten?

19.5 Software-Management und -Kultur

Welche Leute haben wir?

- **Ausbildung:** drei Klassen
 - Gelernte (InformatikerInnen Uni/ETH/FH)
 - Angelernte (Schulungen, Kurse, Lehrgänge besucht)
 - Ungelernte (irgendwie zur Informatik gestoßen, sowie viele Manager)
- **Förderung:** gute Leute schaffen und erhalten
 - Gute Leute fallen nicht vom Himmel
 - Gute Techniker nicht zu schlechten Managern machen
 - Ein(e) Gute(r) ist besser und billiger als drei Schlechte
- **Veränderungen**
 - Mit den Gelernten aufgleisen
 - Die Angelernten gewinnen
 - Den Widerstand der Ungelernten überwinden / ihre Ängste abbauen

Rolle der Infrastruktur

- Arbeitsplatz
 - Platz
 - Ressourcen
 - Störungen
- Arbeitsbedingungen
 - Arbeitszeiten, Überstunden
 - Vollzeit/Teilzeit
 - Betreuung und Kritik
 - Aufstiegschancen
 - Weiterbildung
 - Bezahlung

Der Einfluss der Arbeitsumgebung

- **Ausbildung**
 - Gute Leute einstellen
 - Leute **besser machen**: Fortbildung, geeignete Gruppenprozesse
- **Arbeitsplätze**, an denen gearbeitet werden kann
 - Genug Platz
 - Technisch adäquat ausgerüstet
 - Wenig Störungen

19.1 Grundlagen und Gesetzmäßigkeiten

19.2 Software-Psychologie

19.3 Software-Ingenieure im Unternehmen

19.4 Zwei Welten?

19.5 Software-Management und -Kultur

Technikzentriertes vs. Menschenzentriertes SE – 1

| Technikzentriert | Menschenzentriert |
|--------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| Systeme werden vollständig geplant | Systeme evolvieren, fortlaufende Planung |
| Systeme werden aus Komponenten konstruiert | Systeme wachsen |
| Fokus auf Konstruktion | Fokus auf Verstehen |
| Systeme realisieren/erzwingen Prozesse und Organisationsformen | Systeme helfen Menschen, ihre Arbeit besser/einfacher zu tun |
| Prozesse sind detailliert beschrieben und exakt zu befolgen (→ Workflow) | Prozesse sind Leitplanken / Checklisten |
| Spezifikationen sind vollständig, exakt, möglichst formal und fixiert | Spezifikationen sind partiell und evolutionär (aber wohlorganisiert → Konfigurationsmanagement!) |

Technikzentriertes vs. Menschenzentriertes SE – 2

| Technikzentriert | Menschenzentriert |
|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| Benutzer formulieren Anforderungen und nehmen das fertige Produkt ab | Benutzer sind in den ganzen Entwicklungsprozess involviert |
| Alle Artefakte werden verifiziert; Rückverfolgung Code → Entwurf → Spezifikation | Artefakte werden fortlaufend validiert (erfordert Benutzerbeteiligung!) |
| „Egoless programming“ = Jede(r) ist austauschbar | „Egoless programming“ = Verwenderzentriertes statt erstellerzentriertes Arbeiten |

Software Engineering funktioniert letztlich nur als dynamische Synthese zwischen beiden Welten

19.1 Grundlagen und Gesetzmäßigkeiten

19.2 Software-Psychologie

19.3 Software-Ingenieure im Unternehmen

19.4 Zwei Welten?

19.5 Software-Management und -Kultur

Software-Management

- **Realistische** Planung
- Gegenseitiges **Vertrauen**
- **Informationskultur**



Eine Kultur des Software Engineerings schaffen

- Ziele: **bekannt – anspruchsvoll – realistisch**
- Die **Software-Prostitution** bekämpfen:
Für genug Geld tun sie's auch ohne...
... Sorgfalt, Dokumentation, Test, ...
Sie verachten dabei ihre Arbeit ...
... und schlussendlich sich selbst.
- Wer permanent das **Unmögliche** fordert, erntet **Lüge** und **Resignation**
- Auch Software-Ingenieure brauchen ein **Berufsethos**
- „Hier machen wir das alle so“
→ **Das Richtige zum Selbstverständlichen machen**

Software-Kultur

- Eine Charta des Software Engineerings
 - Wir sind kompetent und schnell, aber wir können nicht alles.
 - Wir orientieren uns an den Bedürfnissen unserer Kunden und geben unser Bestes für sie.
 - Wir akzeptieren keine unrealistischen Aufträge.
 - Wir sind uns der Risiken unserer Arbeit bewusst und stellen uns der Verantwortung.
 - Wir tun die Dinge von Anfang an richtig.
 - Wir prüfen unsere Arbeit und lernen aus unseren Fehlern.
 - Wir haben Spaß am professionellen Arbeiten.

Literatur

Boehm, B.W. and R. Ross (1989). Theory-W Software Project Management: Principles and Examples. *IEEE Transactions on Software Engineering* **15**, 7 (Jul 1989). 902-916.

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.

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

Ludewig, J. (1990). Wie man Informatiker hält. Über Softwareleute und ihre Arbeitsumgebung. *Technische Rundschau Spezial: Schweizerische Marktstatistik für industrielle Software 1990*. 10-13.

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.

Siehe auch Literaturverweise im Kapitel 12 des Skripts.

Im Skript [M. Glinz (2005). *Software Engineering*. Vorlesungsskript, Universität Zürich] lesen Sie Kapitel 12.3.

Im Begleittext zur Vorlesung [S.L. Pfleeger, J. Atlee (2010). *Software Engineering: Theory and Practice*, 4th edition. Upper Saddle River, N.J.: Pearson Education International] lesen Sie Kapitel 13.4 sowie in Kapitel 2.3 die Seite 93 (insb. Abbildung 2.13).