Was ist Softwaretechnik ?

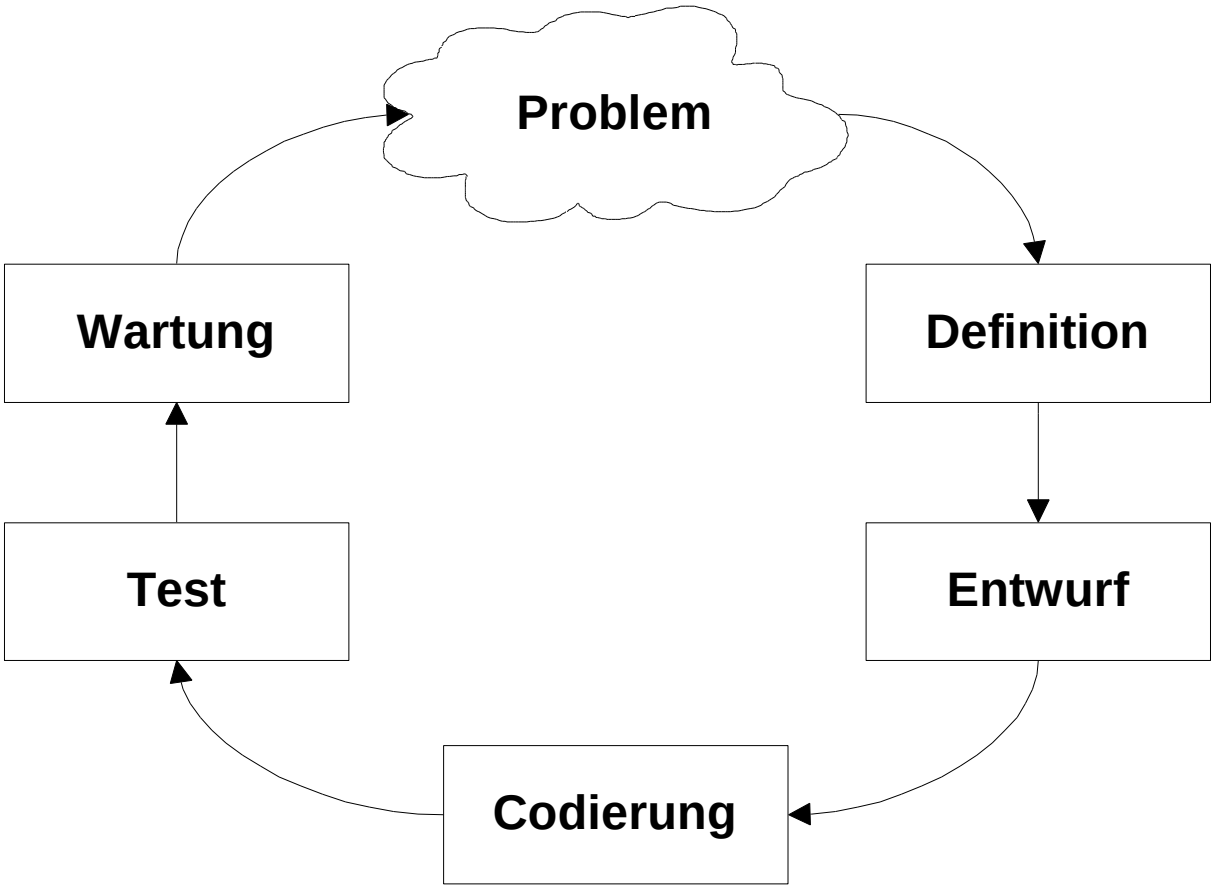
- **Software Engineering: The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them. [Boehm76]**

- **Software-Engineering: das ingenieurmäßige Entwerfen, Herstellen und Implementieren von Software sowie die ingenieurwissenschaftliche Disziplin, die sich mit Methoden und Verfahren zur Lösung der damit verbundenen Problemstellungen befaßt. [Brockhaus]**

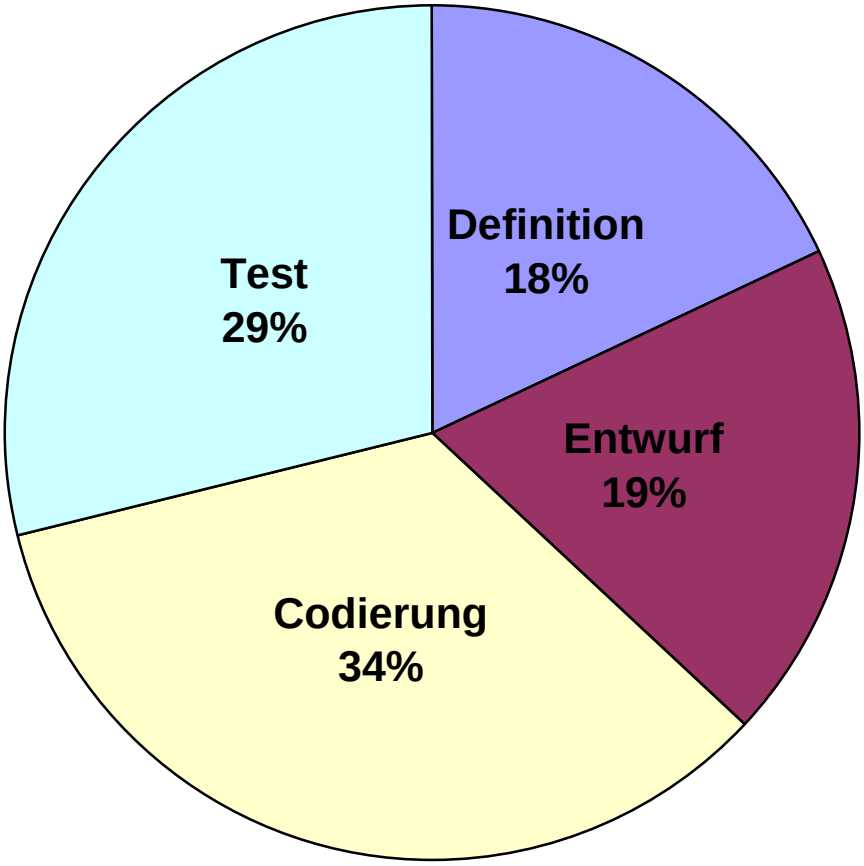
- **Software Engineering ist die praktische Anwendung wissenschaftlicher Erkenntnisse für die wirtschaftliche Herstellung und den wirtschaftlichen Einsatz qualitativ hochwertiger Software. [Pomberger93, S. 3]**

- **Software-Technik: Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden, Konzepten, Notationen und Werkzeugen für die arbeits-teilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Software-Systemen. Zielorientiert bedeutet die Berücksichtigung von z. B. Kosten, Zeit, Qualität. [Balzert96, S. 36]**

Wie entsteht Software?

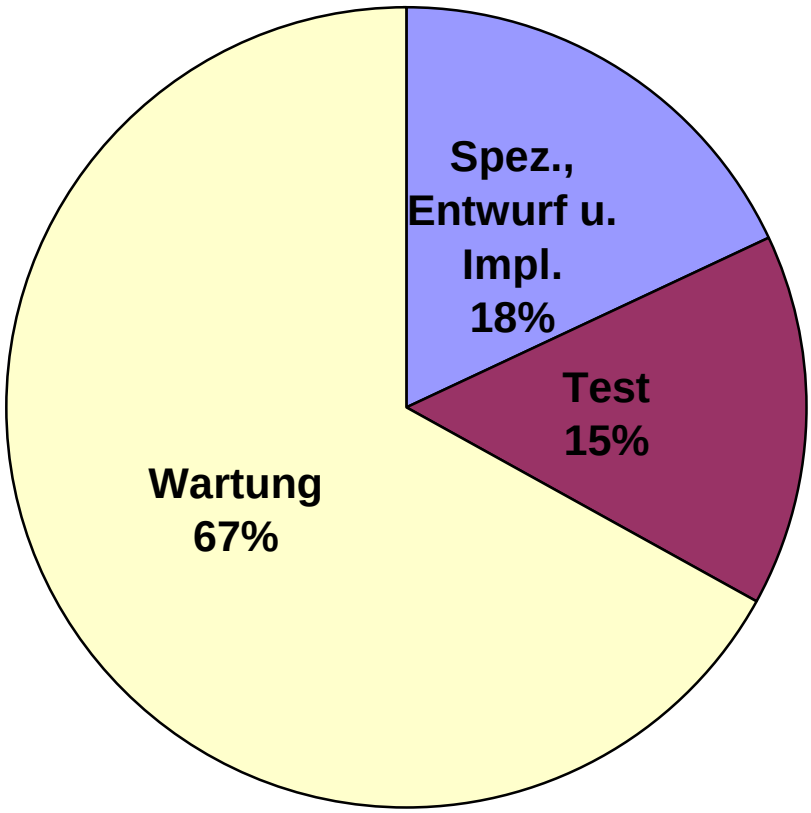


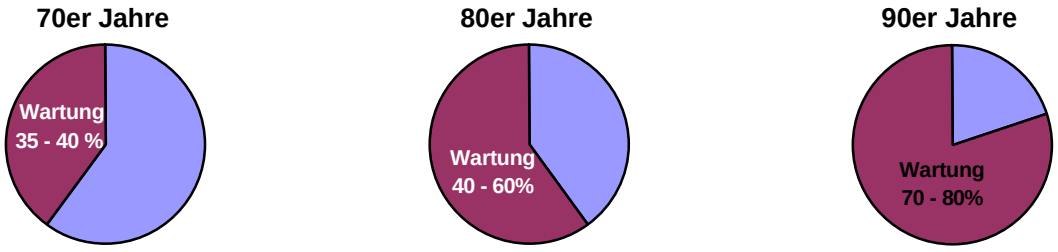
Was kostet Software?



Quelle: [Balzert96, S. 70], [Grady92]

Was kostet Software? (2)





Quelle: [Pomberger93, S. 161], [Arthur88], [Pressmann87]

Wozu guter Programmierstil?

- **Keine eindeutigen Bewertungskriterien für Software**
- **Wartung ist wichtiger und zeitaufwendiger Bestandteil der Software-Entwicklung**
- **Wartung bedeutet, daß alter Code immer wieder gelesen und verstanden werden muß.**
- **Große Software-Systeme bestehen meist aus vielen Einzelteilen. Unter der Annahme, daß ein Teil mit der Wahrscheinlichkeit p korrekt ist und das Gesamtsystem aus n Teilen besteht, dann ist das Gesamtsystem mit der Wahrscheinlichkeit $P = p^n$ korrekt.**

	p		
n	**0,99**	**0,95**	**0,90**
1	**0,99**	**0,95**	**0,90**
10	**0,90**	**0,60**	**0,35**
30	**0,74**	**0,21**	**0,04**
50	**0,61**	**0,08**	**0,01**
100	**0,37**	**0,01**	**0,00**

Was ist guter Programmierstil?

Wichtige Elemente guten Programmierstils sind:

- **Strukturiertheit**
  - **Strukturierung im Großen: Zerlegung eines Programms in mehrere Einzelteile (Module oder Klassen)**
  - **Strukturierung im Kleinen: Nur Bausteine mit einem Eingang und einem Ausgang verwenden: Sequenz, Verzweigung, Wiederholung, Fallunterscheidung, kein GOTO! Bausteine mit einem Eingang und einem Ausgang heißen nach E.W. Dijkstra D-Diagramme.**
- **Äußere Form**
- **Ausdruckskraft**
  - **Namenwahl**
  - **Kommentare**
- **Effizienz**

Quelle: [Pomberger93, S. 137ff]

Guter Programmierstil: Äußere Form

- **In jedem Programmbaustein sollten die Deklarationen deutlich vom Anweisungsteil getrennt werden.**
- **Die Deklarationsteile sollten wenn möglich ein einheitliches Gliederungsschema aufweisen, z. B. durch folgende Reihenfolge: Konstanten, Datentypen, Variablen, Klassen und Module, Methoden und Prozeduren**
- **Die Schnittstellenbeschreibung soll eine Trennung der Eingangs-, Ausgangs- und Übergangsparameter vorsehen.**
- **Kommentare und Programmtext sollten deutlich voneinander getrennt werden.**
- **Die Programmstruktur sollte durch Einrückungen sichtbar gemacht werden.**

Guter Programmierstil: Namenwahl

- **Wähle aussagekräftige Namen, auch wenn sie dadurch lang werden. Der Schreibaufwand lohnt sich spätestens dann, wenn ein Programm nach langer Zeit gewartet werden soll.**

- **Benutze nur allgemein gebräuchliche Abkürzungen, die der Leser eines Programmes ohne weitere Erläuterungen verstehen kann. Verwende Abkürzungen immer im gleichen Sinn (also nicht temp für Temperatur und temporär).**

- **Vergib innerhalb eines Programms nur Namen in einer Sprache (nicht deutsch/englisch).**

- **Benutze Groß-/Kleinschreibung, um verschiedene Arten von Bezeichnern zu unterscheiden (z. B. große Anfangsbuchstaben für Datentypen, Klassen und Module, kleine für Variablen) und um lange Namen besser lesbar zu machen (z. B. CheckInputValue).**

- **Verwende Hauptwörter für Werte, Zeitwörter für Tätigkeiten und Eigenschaftswörter für Bedingungen (z. B. breite, ReadKey, gueltig, Get...., Set....).**

- **Stelle einmal für dich selbst Regeln auf und befolge sie dann konsequent.**

Guter Programmierstil: Namenwahl (2)

- **Ungarische Notation? (Datentyp einer Variable im Namen aufnehmen [Simonyi91])**
  - **ul.....unsigned long**
  - **pul.....pointer to unsigned long**
  - **us.....unsigned short**
  - **pus .....pointer to unsigned short**
  - **puc .....pointer to unsigned char**
  - **psz.....pointer to zero-terminated string**
  - **dw .....double word**
  - **lpi.....far pointer to int**
  - **lpstr .....far pointer to string**
- **Für Laufvariablen in for-Schleifen verwendet man gerne i, j, k, l, ...**
- **Siehe auch [Balzert96, S. 947ff]**

Guter Programmierstil: Kommentare

- **Jede Komponente sollte mit einem ausführlichen Kommentar beginnen:**
  - **Was leistet die Komponente?**
  - **Wozu wird die Komponente verwendet?**
  - **Welche Verfahren, spezielle Methoden, ... werden verwendet?**
  - **Wer ist der Verfasser?**
  - **Wann wurde die Komponente geschrieben?**
  - **Welche Änderungen wurden bereits vorgenommen?**
- **Jede Prozedur bzw. Methode sollte mit einem Kommentar versehen werden, der ihre Aufgabe und Schnittstelle beschreibt.**
- **Die Bedeutung von Variablen sollte kommentiert werden.**
- **Programmteile, die für abgeschlossene Teilaufgaben verantwortlich sind, sollten durch Kommentare gekennzeichnet werden.**
- **Schwer verständliche Anweisungen (z. B. trickreiche Verfahren oder Programmteile, die Besonderheiten eines bestimmten Computers ausnutzen) sollen in Kommentaren so beschrieben werden, daß sie leicht vom Leser verstanden werden können.**

Guter Programmierstil: Kommentare (2)

- **Ein Programmsystem sollte so wenig und so knappe Kommentare wie möglich, aber so viele und so ausführliche Kommentare wie nötig enthalten.**

- **Es ist besonders darauf zu achten, daß Programmänderungen sich nicht nur auf Deklarationen und Anweisungen auswirken, sondern daß auch die Kommentare im Programmsystem immer auf dem laufenden sind (falsche Kommentare sind schlechter als überhaupt kein!).**

- **Die Bedingung, die in einem else-Zweig gilt, wird gerne neben dem else kommentiert.**

Guter Programmierstil: Effizienz

- **Effizienz bedeutet nicht nur, ein Programm hinsichtlich seiner Laufzeit zu optimieren.**

- **Beispiel:**

```
x = 1;
for (i = 0; i < N; i++) {
    a[i] = new int[x];
    x++;
}
```

Die Variable x ist überflüssig. Sie ist immer i + 1. Das Programm kann folgendermaßen geändert werden. Durch diese Änderungen wird es kürzer und verständlicher.

```
for (i = 0; i < N; i++) {
    a[i] = new int[i + 1];
}
```

- **„Erst wenn ein Problem und seine Lösung richtig verstanden wurde, hat es Sinn, die Effizienz zu untersuchen.“ [Pomberger93, S. 138]**

Beispiele

- **Der Variablenname "WPSMHGW" für den Grenzwert in einer Wämepumpensteuerung programmiert von Maier Hans ist keine glückliche Wahl.**

- **Namenwahl:**
  - **Feld1, Feld2, Zaehler**
    - **Messreihe1, Messreihe2, AnzahlZeichen**
  - **P = G2 + Z1 * D**
    - **Praemie = Grundpraemie2 + Zulage1 * Dienstjahre**

- **Kommentare:**
  - **i = i + 1 // i wird um eins erhöht**
    - **Lagermenge = Lagermenge + 1**

- **Strukturierung (zwei Ausgänge, kein D-Diagramm):**

```
/* -- Suche x in einem Array -- */
for (i = 0; i < arr.length; i++) {
   if (arr[i] == x) return i;
}
return 0;
```

Beispiele (2)

- **Äußere Form**

```
int atoi(char *n) {
int r=0;
while (1) {
if (!*n) return(r);
r+=*n+++r+(r<<3)-48;}}
```

- **Korrektheit**

```
while (!eof(f)) {
    c = f.read();
    f.write(c);
}
```

Wozu Testen?

Menschen machen Fehler:

- **Venussonde**
- **Jagdbomber F18**
- **Hamburger Bahnhof Altona [Möcke95]**
- **Strahlentherapiegerät Therac-25 [Loviscach98]**
- **Denver International Airport [Loviscach98]**
- **Ariane-5 Rakete [Loviscach98]**
- **Lufthansa Airbus A320 in Warschau [Loviscach98]**

Fehlermeldungen zum Schmunzeln [Loviscach98]:

- **"Die Informationsverarbeitung ist auf oberster Ebene fehlgeschlagen."**
- **"Der Arbeitsspeicher reicht nicht aus. Es werden 1,2 MB benötigt. Vorhanden sind aber nur 1,3 MB."**
- **Siehe auch http://www.dasding.de/dialoge/dialog.htm**

Wie testen?

- **Ein Test ist erfolgreich, wenn ein Fehler gefunden wurde!**
- **Testverfahren:**
  - **Verfikation**
  - **Statische Programmanalyse (Code-Inspektion, Komplexitätsanalyse, Strukturanalyse, Datenflußanalyse)**
  - **Dynamisches Testen (Black-Box- und White-Box-Test, Topdown- und Bottomup-Test)**
- **Typische Programmfehler:**
  - **Nichtberücksichtigung von Sonderfällen (z. B. Division durch Null)**
  - **Fehlende Behandlung von Grenzwerten**
  - **Überschreitung von Feldgrenzen**
  - **Iterative Schleifen, die nicht abbrechen**
  - **Nichtinitialisierung von Variablen**
  - **Falsche logische Ausdrücke (besonders in Verbindung mit dem Negationsoperator)**

Zusammenfassung

"Als es noch keine Rechner gab, war auch das Programmieren noch kein Problem, als es dann ein paar leistungsschwache Rechner gab, war das Programmieren ein kleines Problem und nun, wo wir gigantische Rechner haben, ist auch das Programmieren zu einem gigantischen Problem geworden. In diesem Sinne hat die elektronische Industrie kein einziges Problem gelöst, sondern nur neue geschaffen. Sie hat das Problem geschaffen, ihre Produkte zu benutzen." [Dijkstra72]

Bibliographie

[Arthur88] **Arthur L. J., Software Evolution: A Software Maintenance Challenge, John Wiley & Sons, 1988**

[Balzert96] **Helmut Balzert, Lehrbuch der Software-Technik, Heidelberg, Berlin, Oxford, Spektrum Akademischer Verlag, 1996**

[Bleul98] **Bleul Andreas, Loviscach J., Programmieren nach Plan, c't – Magazin für Computertechnik, 19/98, S. 166 - 172**

[Boehm76] **Boehm B.W., Software Engineering in: IEEE Transactions on Computers, Dec. 1976, S. 1226 – 1241**

[Dijkstra72] **Dijkstra E. W., The Humble Programmer, Communication of the ACM, Vol. 15, No. 10, New York, 1972**

[Grady92] **Grady R.B., Practical Software Metrics for Project Management and Process Improvement, Englewood Cliffs, Prentice Hall, 1992**

[Grams90] **Grams T., Denkfallen und Programmiefehler, Berlin: Springer-Verlag, 1990**

[Loviscach98] **Loviscach J., Absturzgefahr – Die Bug-Story, c't – Magazin für Computertechnik, 19/98, S. 156 – 165**

[Möcke95] **Frank Möcke, Alle Räder stehen still - Software-Bug legte Stellwerk lahm, c't – Magazin für Computertechnik 5/95, S. 15**

[Pomberger93] **Pomberger G., Blaschek G., Grundlagen des Software Engineering – Prototyping und objektorientierte Software-Entwicklung, Carl Hanser Verlag, 1993**

[Pressmann87] **Pressmann R. S., Software Engineering. A Practitioner's Approach, McGraw-Hill, 1987**

[Simonyi91] **Simonyi Charles and Heller Martin, The Hungarian Revolution, Byte, August 1991, S. 131 – 138**

[Wallmüller90] **Wallmüller Ernest, Software-Qualitätssicherung in der Praxis, München-Wien: Carl Hanser Verlag, 1990**