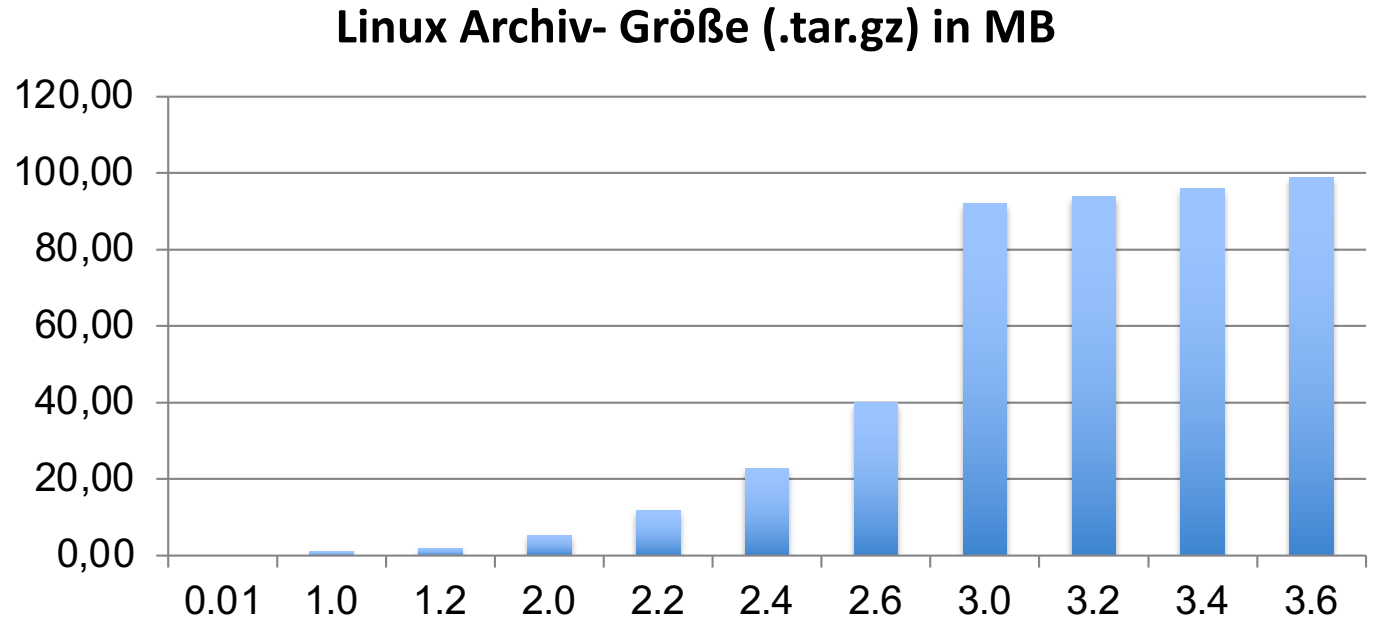
Was sind typische

Fehlerquellen

- Gründe für die schlechte Qualität von Software
  a. Die speziellen Eigenschaften von Software führen schnell zu einer schwer zu beherrschenden Komplexität
  b. Software-Größe steigt stark an



  c. Starke Fluktuation von Mitarbeitern / Mangelnde Qualifikation
  d. Management versucht Erfahrungen aus Produktionsprozessen auf die Softwareentwicklung zu übertragen

Fehlerquellen nach [Hof13]

1. Lexikalische und syntaktische Fehlerquellen
   - Welche Ausgabe liefert die folgende Sequenz?

```
int a = 2;
int b = 1;
int c = a---b;
printf("%i\n", c);
```

   - Gibt es hier ein Problem?

```
y = x/*p /* p ist Pointer auf Divisor */;
```

   - Arbeitet diese Funktion korrekt?

```
float nrm(float x)
{
    while(abs(x)>0,1)
        x = x / 10;
    return x;
}
```

2. Numerische Fehlerquellen

- Welche Ausgabe wird geschrieben?

```
double x = 0.1*3.0;
double y = 3.0/10.0;

if (x == y) {
    System.out.println(x + " gleich " + y);
}
else {
    System.out.println(x + " ungleich " + y);
}
```

- Numerische Überläufe
  - Y2K-Bug
  - Zeitdarstellung im POSIX-Format (Problem am 19.01.2038 um 3:14:08 UTC)
  - ...

3. Semantische Fehlerquellen

- Wo liegt das Problem?

```
String input =
      JOptionPane.showInputDialog("Raumtemperatur (in
Celsius)");
float grad = Float.valueOf(input);
if (Util.isTooHot(grad)){
    JOptionPane.showMessageDialog(null, "Sie haben Hitzefrei!");
} else{
    JOptionPane.showMessageDialog(null, "Sie müssen arbeiten!");
}
```

```
public class Util {
    static boolean isTooHot(float temperature){
        if (temperature > 82.4)
            return true;
        return false;
    }
}
```

4. Parallelität als Fehlerquelle

- Wo liegt das Problem?

Eine Klasse Produzent produziert Waren, die in in einem Lager mit der Kapazität 20 abgelegt werden. Eine Klasse Konsument entnimmt aus dem Lager Waren. Produzent und Konsument werden als Threads gestartet

```
Lager lager = new Lager();

Thread tp = new Produzent(lager);
Thread tk = new Konsument(lager);

tp.start();
tk.start();
```

```
class Lager {
   private int bestand;
   private static final int kapazität = 20;

   public void einlagern() {
      if (bestand+5 <= kapazität) {
         bestand+=5;
         System.out.println("nach Einlagern :" + bestand);
      }
   }

   public void entnehmen() {
      if (bestand-3 >= 0) {
         bestand-=3;
         System.out.println("nach Entnehmen :" + bestand);
      }
   }
}
```

```
class ProduzentThread extends Thread {
   private Lager lager;

   ProduzentThread(Lager lager) {
      this.lager = lager;
   }

   public void run() {
      while(true){
         lager.einlagern();
      }
   }
}
```

Klasse KonsumentThread ist analog implementiert, nur wird hier die Methode lager.entnehmen() aufgerufen

Ein Programmstart führt zur folgenden Ausgabe:

```
...........
nach Entnehmen :  1
nach Einlagern :  6
nach Einlagern :  8
nach Einlagern : 13
nach Einlagern : 18
nach Entnehmen :  3
nach Entnehmen : 15
nach Entnehmen : 12
```

Wie kann das Problem behoben werden?

Race conditions

Deadlocks

5. Spezifikationsfehler
   - Bisher haben wir nur Implementierungsfehler betrachtet
   - Ein Fehler kann aber bereits schon in der Spezifikation enthalten sein
     - Unvollständige Vorgaben
     - Widersprüchliche Vorgaben
     - Ungeeignete Berechnungen
   - Solche Fehler können mit Verfahren, die gegen die Spezifikation testen, nicht entdeckt werden

6. Portabilitätsfehler
   - Es kann sein, dass eine Software auf einem System (Hardware + Betriebssystem + Laufzeitumgebung) fehlerfrei läuft, auf einem anderen System Fehler zum Vorschein kommen
   - Ursachen
     - CPU-Geschwindigkeit
     - Prozess-Scheduler
     - Speicherplatz / Speicherverwaltung
     - Zahlendarstellung

7. Hardwarefehler
   - Nicht immer muss die Software Schuld an einem Fehler sein
   - Ein Hardwarefehler kann zu fehlerhaften Berechnungen führen
     - Falsche Berechnung der Divisionseinheit von Pentium I – Prozessoren (Produktionsjahr 1993)

Qualitätssicherung
Qualitätsmanagement

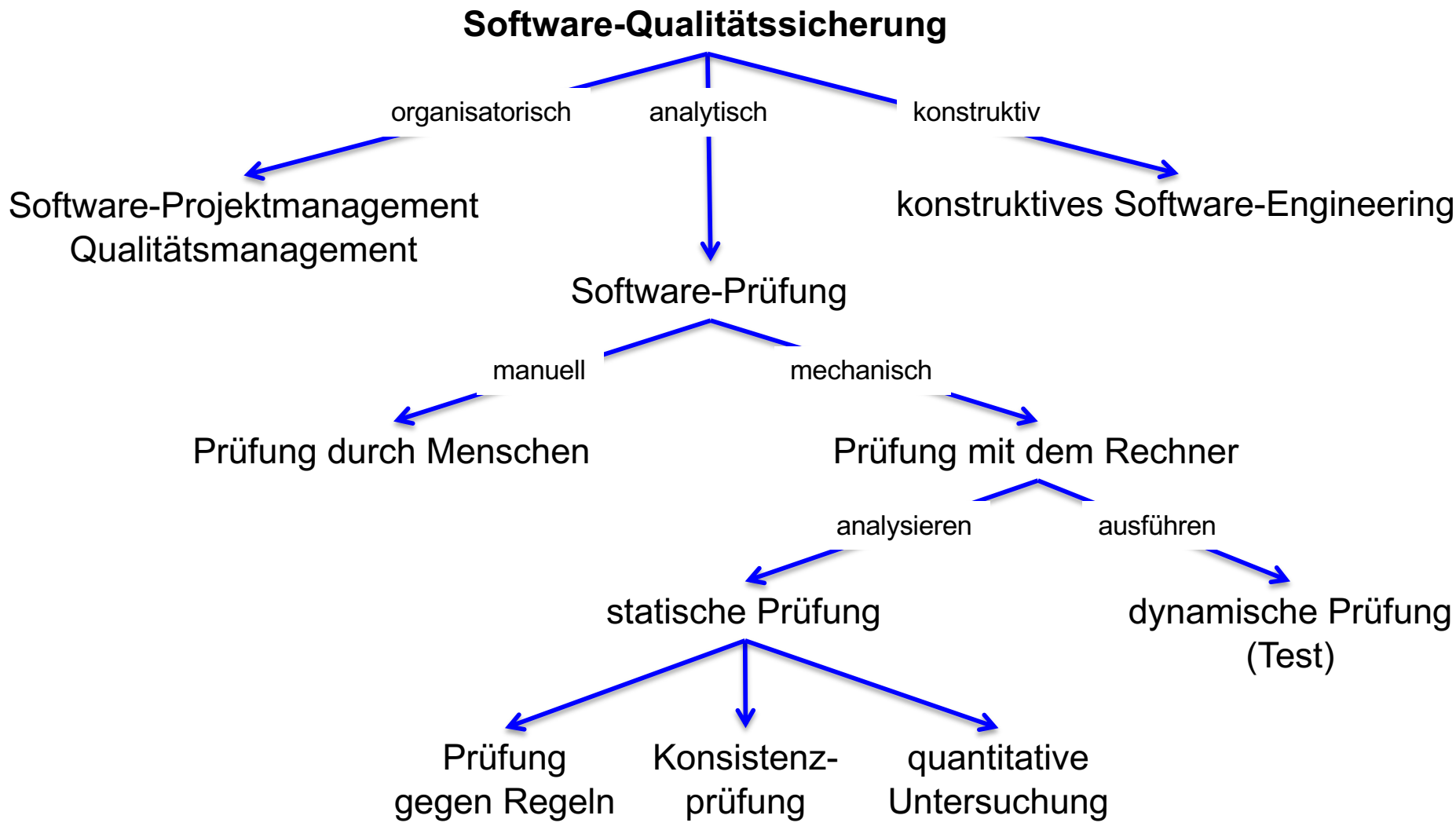
– Es werden drei Maßnahmen zur Software-Qualitätssicherung unterschieden:

1. Organisatorische Maßnahmen
   - Systematische Entwicklung und Qualitätssicherung
   - inkl. Zeitplanung für Prüfung und Korrekturen
   - inkl. Festlegung der Verantwortlichkeiten für Prüfungen
   - Vorgabe von Richtlinien, Standards und Checklisten

2. Konstruktive Maßnahmen
   - „Es kann keine Qualität in ein Produkt hineingeprüft werden“
   - Fehler und schlechte Qualität von Beginn an vermeiden

3. Analytische Maßnahmen
   - Software-Prüfung
   - Fehler und Mängel in den Arbeitsergebnissen erkennen

Organisatorische und konstruktive Maßnahmen sind hilfreicher als analytische Maßnahmen



Gliederung der Software-Qualitätssicherung nach [LL10]

- Die geforderten Qualitätsanforderungen müssen definiert, und die Einhaltung der Qualitätsanforderungen muss überprüft werden
- Es müssen geeignete Rahmenbedingungen geschaffen werden

- **Qualitätsmanagement:**

  Aufeinander abgestimmte Tätigkeiten zum Leiten und Lenken einer Organisation bezüglich Qualität, die üblicherweise das Festlegen der Qualitätspolitik und der Qualitätsziele, die Qualitätsplanung, die Qualitätslenkung, die Qualitätssicherung und die Qualitätsverbesserung umfassen.

Aktivitäten im Qualitätsmanagement

(nach Skript „System- und Softwaresicherung“ von Prof. Dr. Ecke-Schüth)

a) Qualitätsplanung
   - Festlegung von Qualitätsanforderungen an den Prozess und das Produkt in überprüfbarer Form

b) Qualitätslenkung und –sicherung
   - Umsetzung, Steuerung, Überwachung und Korrektur des Entwicklungsprozesses mit dem Ziel, die vorgegebenen Anforderungen zu erfüllen

c) Qualitätsprüfung
   - Durchführung der im Rahmen der Qualitätsplanung festgelegten Maßnahmen
     - Erfassung der Ist-Werte der Qualitätsindikatoren
     - Überwachung der Umsetzung der konstruktiven Maßnahmen
     - Tests, Durchsicht, Reviews, ...

d) Qualitätsverbesserung
   - Auswertung der Qualitätssicherungsergebnisse und der Prozessverbesserungen

– Dokumentation
   - Qualitätssicherungsplan (Prozess-orientiert)
     - Ergebnisse der Qualitätsplanung
   - Prüfplan (Produkt-orientiert)
     - Begleitende Maßnahmen

Qualitätsplanung ➔ Qualitätssicherungsplan

a) Festlegung der Aufgaben: Was ist zu tun?
   - Identifizierung der zu sichernden Produkte
   - Identifizierung der relevanten Qualitätsmerkmale
     - Relative Bedeutung
     - Quantifizierung in Form von Metriken

a) Festlegung der Vorgaben und Hilfsmittel: Wie ist es zu tun?
   - Auswahl der zur Datenerfassung und Qualitätsprüfung geeigneten Techniken und Methoden
   - Konstruktive Vorgaben (Richtlinien, Vorlagen, Werkzeuge, ...)
   - Analytische Vorgaben (Verfahren, Werkzeuge, ...)

c) Festlegung der Termine: (Bis) wann ist es zu tun?
   - Festlegung des zeitlichen Verlaufs der Datenerfassung/Dokumentation über den Entwicklungsprozess
   - Abstimmung des Prüfplans mit dem Projektplan

d) Festlegung der Verantwortlichkeiten: Wer hat es zu tun?
   - Festlegung der Verantwortlichkeiten für die Qualitätsprüfung und –lenkung
   - Definition und Besetzung von Rollen (Qualitätsmanager, Prüfer, Autor, Gutachter)

Qualitätslenkung

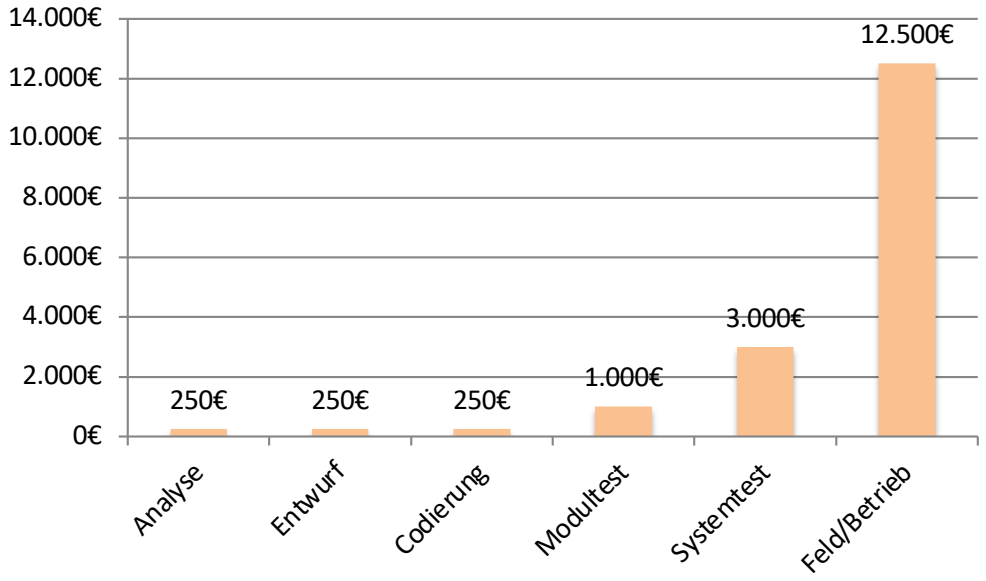
a) Reguläre Aktivitäten
   - Durch entwicklungsbegleitende Qualitätsprüfungen sind die Anforderungen sicherzustellen

b) Besondere Aktivitäten

c) Finale Aktivitäten
   - Wenn alle Qualitätsziele für ein (Teil-)Produkt erfüllt sind, kann eine formale Abnahme erfolgen (Produktzertifikat)

Wann sollte mit der Qualitätssicherung begonnen werden?

- Nach Untersuchungen von Boehm führt eine späte Fehlerentdeckung zu einem exponentiellen Kostenanstieg
- Das Zahlenmaterial wurde von Boehm 1981 veröffentlicht. Ein überproportionaler Kostenanstieg wird aber auch in der heutigen Zeit als realistisch angesehen (siehe zu diesem Thema [Bal08])
- In [Bal08] sind Daten zu den Kosten pro Fehlerkorrektur angegeben, die von Möller im Jahr 1996 publiziert wurden:



Für uns sind die relativen Kostenangaben von Interesse

- Fehler, die früh in der Entwicklung entstehen und für einen längeren Zeitraum nicht entdeckt wurden, können zu Summationseffekten in späteren Phasen führen

  Je früher ein Fehler entdeckt wird, desto kostengünstiger kann er behoben werden

- Die beiden folgenden Ziele sollten in der angegebenen Reihenfolge verfolgt werden:
  1. keine Fehler machen (z.B. durch geeignete konstruktive Maßnahmen)
  2. Fehler, die dennoch gemacht wurden, möglichst früh entdecken und beseitigen

Qualität und Projekt

- Qualität gibt es nicht geschenkt
  - Qualität, Funktionalität, Kosten und Zeit sind über die Produktivität gekoppelt
  - Im Laufe eines Projekts ist die Produktivität konstant