Anforderungen suchen und finden

"The hardest single part of building a software system is deciding precisely what to build.

No other part of the conceptual work is so difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems.

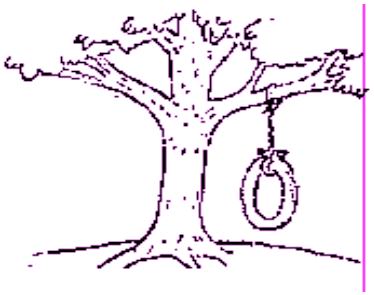
No other part of the work so cripples the resulting system if done wrong.

No other is more difficult to rectify later."

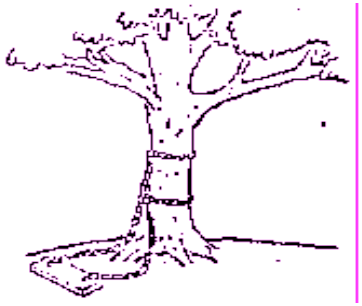
Fred P. Brooks, Jr.: *„The Mythical Man-Month: Essays on Software Engineering“, Jubiläumsausgabe 1995*
(Originalartikel *„No Silver Bullet“* in Proc 10th IFIP World Conference, 1986)

Was sind Anforderungen?

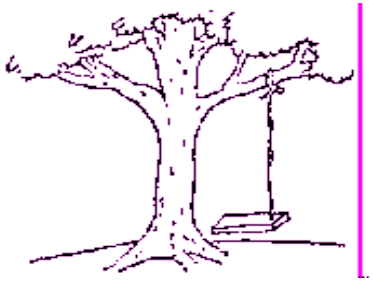
„Ist doch *klar*, was *gemeint* ist...“



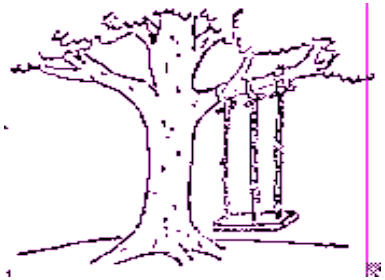
Was die Kinder wollten



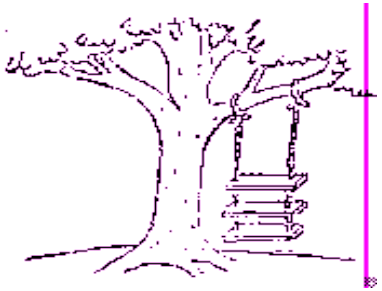
Was der Architekt plante



Entwurf der Baubehörde



Nach Berechnung der Statiker



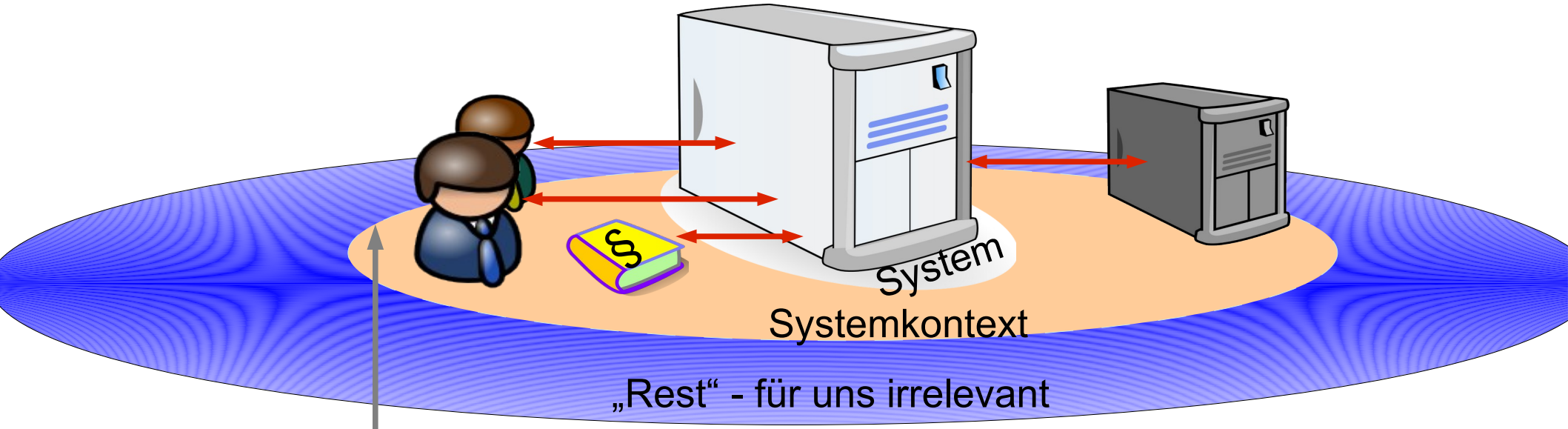
Anpassung an die Bauordnung

Sanierungs-vorschlag

http://www.crazybytes.at/fun/003.htm

Ausgangspunkt: Was gehört zum System...

... und was *nicht*?



Systemkontext:

- Personen („Stakeholder“)
- Nachbar-Systeme
- Regelungen, Vorschriften
- Betriebliche Abläufe
- ...

… mit Relevanz für das System

Was ist eine Anforderung?

Eine Anforderung ist:

- (1) Eine Bedingung oder Fähigkeit, die von einem Benutzer (Person oder System) zur **Lösung eines Problems** oder zur **Erreichung eines Ziels** benötigt wird.
- (2) Eine Bedingung oder Fähigkeit, die ein System oder Teilsystem erfüllen muss, um einen **Vertrag**, eine **Norm**, eine **Spezifikation** oder andere, formell gegebene Dokumente zu erfüllen.
- (3) Eine **dokumentierte Repräsentation** einer Bedingung oder Eigenschaft gemäß (1) oder (2)

IEEE Std 610.12-1990

... oder etwas handlicher

Eine Anforderung ist eine **Aussage**

über eine **Eigenschaft oder Leistung**

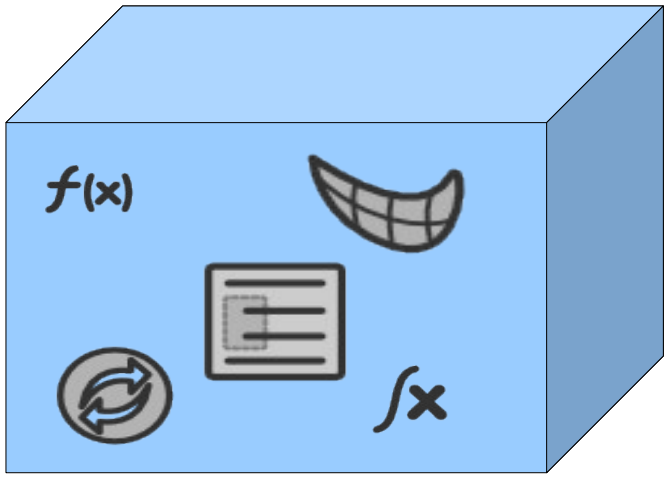
eines **Produktes**, eines **Prozesses** oder

am Prozess beteiligter **Personen**

[Rupp2009]

Funktionale Anforderungen

- **Funktionale Anforderungen („*was*“)**
  - vom System **zu leistende Dienste / Funktionen**
  - **Reaktion** auf bestimmte Eingaben / Fehlersituationen
  - System**verhalten** in bestimmten Situationen
    (ggf. auch Abgrenzung: was soll das System *nicht* tun)
  - **Schnittstellen** zu externen Systemen
  - Ein-/Ausgabe**formate** von **Daten**



Gibt's auch andere Anforderungen?

Ja... die nichtfunktionalen Anforderungen

Qualitätsanforderungen

- Positive Qualitätsanforderungen zu Systemfunktionen:
  „**wie** (gut / viel / schnell / ...)?“

- *Qualität* im ursprünglichen Sinn von „*Beschaffenheit*“

- Typische Beispiele:
  - Usability / Benutzbarkeit
  - Verlässlichkeit, Robustheit
  - Performance, Skalierbarkeit
  - Portabilität (Übertragbarkeit)

Qualitätsanforderungen

- **Bezug** zu funktionalen Anforderungen;

  Qualitätsanforderungen betreffen oft **Querschnittsaspekte**, die bei der Umsetzung der funktionalen Anforderungen und insbesondere bei der **Auswahl von Umsetzungsalternativen** zu beachten sind.

- Forderung „Anforderungs-**Überprüfbarkeit**“ gilt auch hier.

„Das System muss intuitiv zu bedienen sein, schnell rechnen und viele Anfragen verarbeiten können“

Findet der Laie auch. Aber was nützen dem Entwickler solche weichen „Anforderungen“?

Interpretationsspielraum und Klärungs-Faulheit = Grundlage für spätere Streitigkeiten

Beispiele: Überprüfbarkeit

- *intuitiv bedienbar*
- *ausreichend kommentiert*
- *schnelle Namenssuche, auch bei großen Datenmengen*

- *Ein Sachbearbeiter soll das System ohne weitere Schulung nutzen können.*
- *JavaDoc-Kommentare mindestens für alle Klassen und öffentliche Methoden*
- *Bei bis zu 100.000 verwalteten Personal-Datensätzen beträgt Antwortzeit bei Namenssuche unter 5 Sek.*

Dazu: Mengengerüst bestimmen

- Zu erwartende Umfänge abschätzen / bestimmen, z.B.
  - wieviele *Anfragen pro Sekunde* zu bearbeiten sind
  - wie hoch ist das *Datenvolumen* pro Nachricht ist
  - wieviele *Benutzer gleichzeitig* online sind?
  - ...
- Mindest- / Höchstwerte? Durchschnitt / Erwartungswert?
- Dieses „**Mengengerüst**“ hat direkten Einfluss auf...
  - spätere Softwaredesign-Entscheidungen,
  - Algorithmenwahl,
  - Technologieauswahl,
  - ...

Randbedingungen / Einschränkungen

- **Randbedingungen** sind Vorgaben, die von den Projektbeteiligten **nicht beeinflussbar** sind.

- Anders als funktionale und Qualitäts-Anforderungen werden sie **nicht** im System **umgesetzt**, sondern sie **schränken** die **Umsetzungsmöglichkeiten ein**.

- Beispiele:
  - Gesetzliche Regelungen, z.B. zum Datenschutz.
  - *„Das System muss in unserem Rechenzentrum laufen.“*
  - *„Das Projekt-Budget beträgt € 125.000,- (Festpreis).“*
  - *„Die Web-Schnittstelle muss gemäß der unternehmensweiten CI-Vorschriften gestaltet sein.“*

Anforderungen... an Anforderungen

verständlich

vollständig

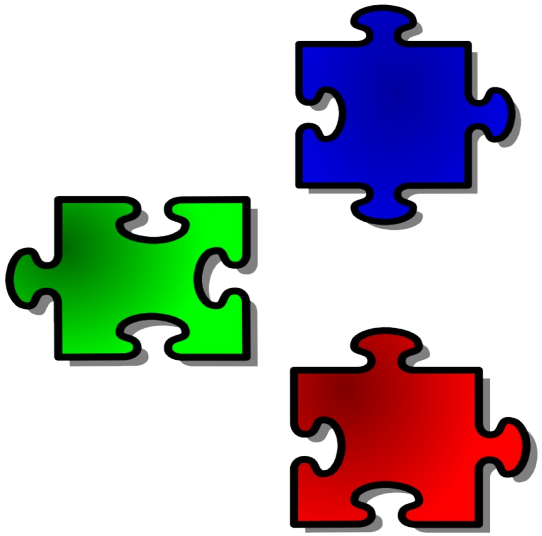
notwendig

eindeutig

korrekt

priorisiert

widerspruchsfrei



abgestimmt

gültig, aktuell

verfolgbar

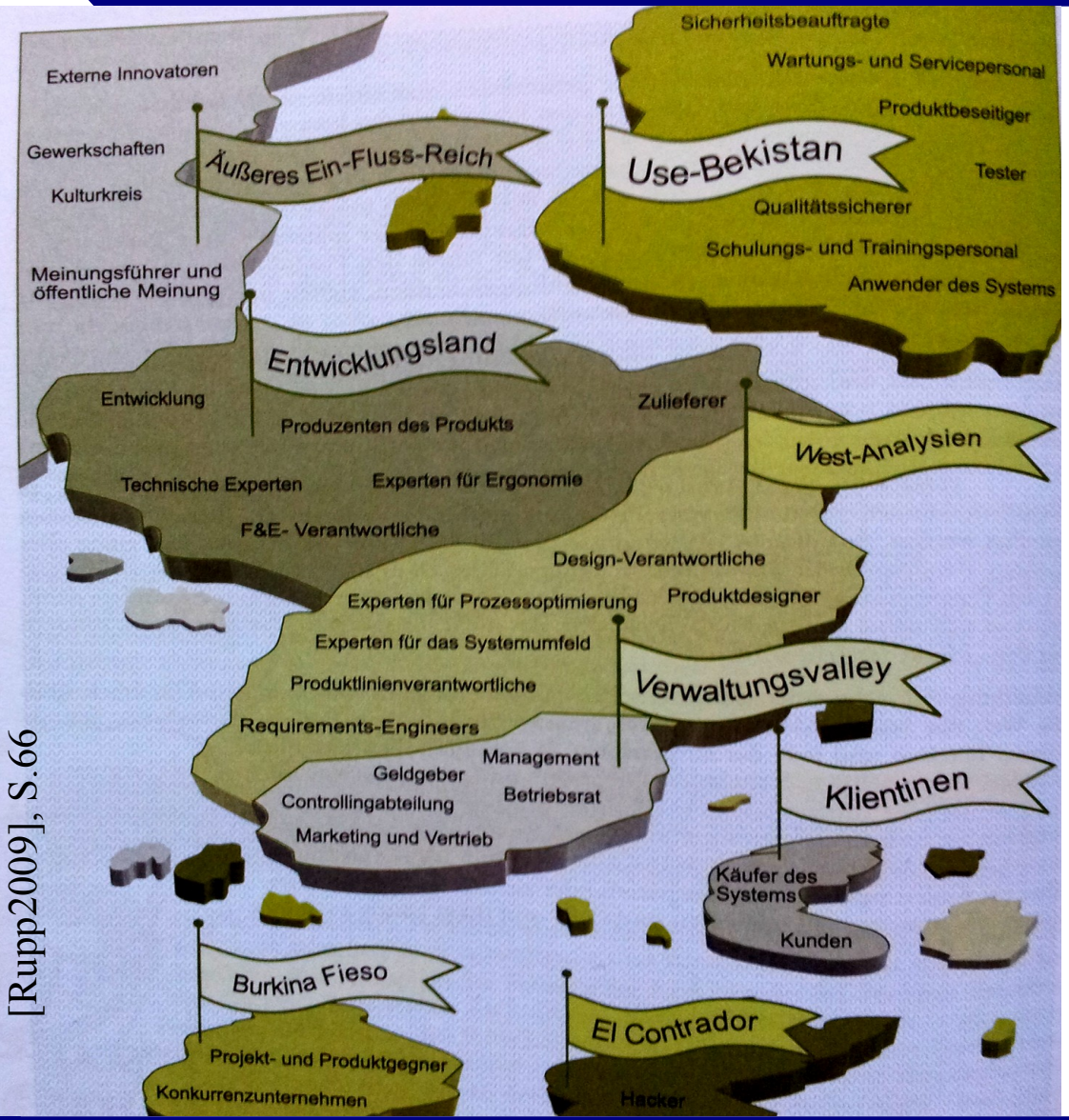
Klassifizierbar
bzgl. rechtlicher Verbindlichkeit

realisierbar

überprüfbar

Woher kommen Anforderungen?

Die Welt der Stakeholder



[Rupp2009], S.66

- **Stakeholder**:
  „Projekt-Betroffener“; Person oder Organisation mit direktem oder indirektem Einfluss auf die Anforderungen an das System

- Sollte man **kennen** und geeignet **berücksichtigen.**

- **Beispiele**:
  - Management
  - Spätere Benutzer und Betreiber (SysAdmins)
  - Entwickler, Tester
  - Marketing, Vertriebspartner
  - Betriebsrat, Gewerkschaften
  - Aufsichtsbehörden
  - ...

Alle wollen das neue System...

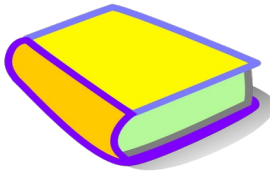
- … wirklich? **Gegenbeispiele**:
  - Unterschiedliche Bewertung des Systems im Management („brauchen wir das wirklich?“)
  - Angst von Abteilungsleitern vor Einflussverlust und Zuständigkeitsverschiebungen im Unternehmen
  - Angst vor Verlust des Arbeitsplatzes
  - Angst vor Statusverlust und Überforderung
  - …

- Projektverantwortliche müssen das rechtzeitig erkennen, sonst läuft man **Gefahr**, dass z.B.
  - … nötige Kooperation oder Infos vorenthalten werden
  - … der Projektfortschritt behindert wird
  - … das Projekt bei ersten Problemen „gekippt“ wird.

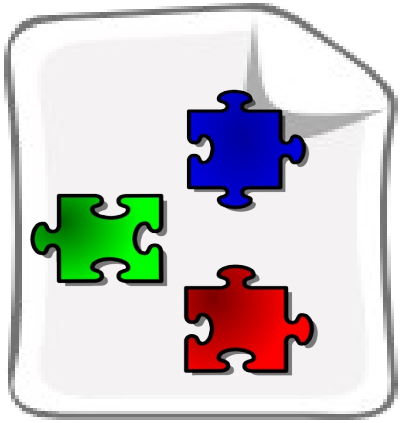
- Zuhören, informieren, frühzeitig einbeziehen.

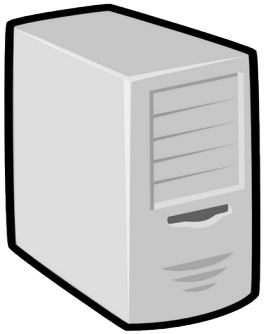
Quellen für Anforderungen

Stakeholder



Dokumente analysieren





abzulösendes Altsystem

Konkurrenz-Systeme

Anforderungen aus Leuten extrahieren

Rollen z.B.

Analytikerin

sammelt / ordnet / analysiert / dokumentiert Anforderungen

Kunde / Auftraggeber
schildert seine Vorstellungen vom neuen System, kann Auskunft über seine fachlichen Anforderungen geben

Domänenexpertin
verfügt über besondere fachliche Detailkenntnisse zu einem bestimmten Themenbereich

Laptop
... fragt sich, warum ihn alle anstarren :-)

Interview

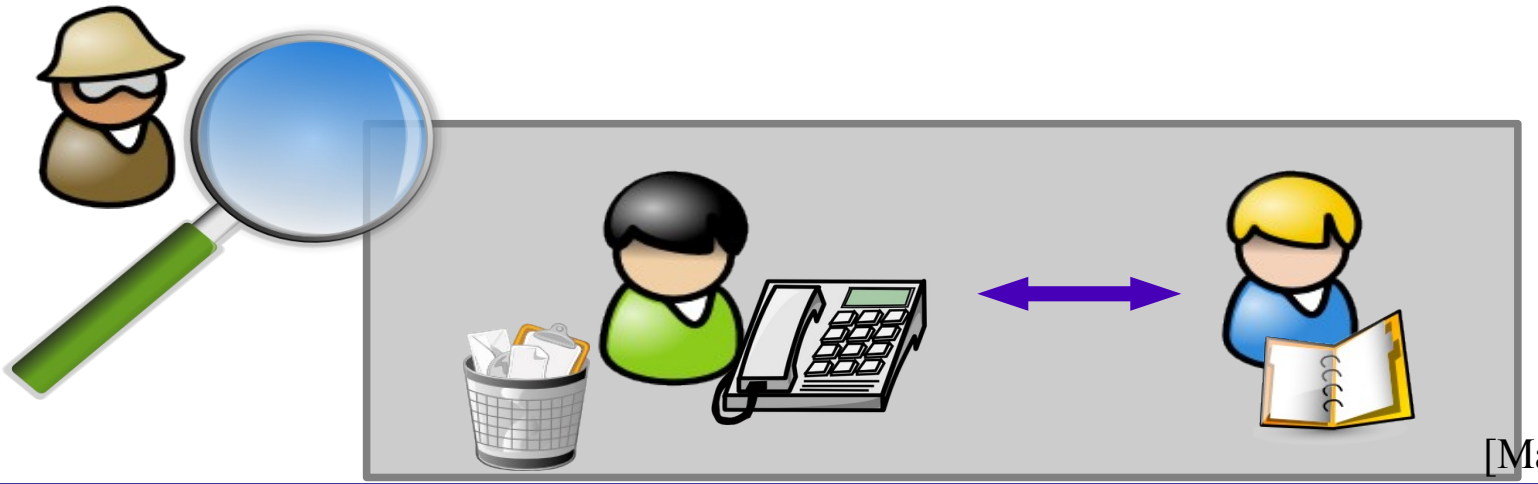
- **Interviews** mit Kunden, Fachexperten, ...
- Interview-**Formen**:
  - strukturiert - Fragen vorbereitet, klare Agenda
  - unstrukturiert - keine vorbereiteten Fragen, mehr Freiraum für Gesprächspartner, führt möglicherweise zu wichtigen neuen (vorher unerwarteten) Aspekten
- Mögliche abzudeckende **Fragebereiche**:
  - Details (was / wer / wann / wo / warum)
  - Zukunftsvorstellungen
  - mögliche Alternativen
  - minimale akzeptierte Lösung des Problems
  - weitere Informationsquellen
  - nützliche Illustrationen (Diagramme, Zeichnungen)

[Maciaszek 2005]

Beobachtung

Beobachtung

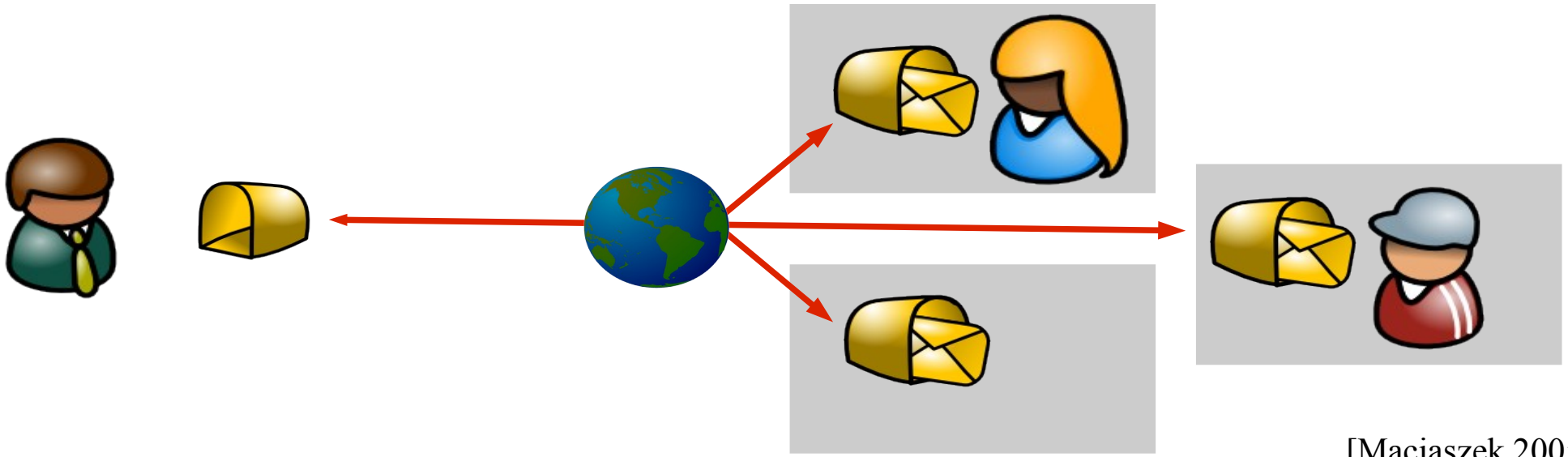
- Lernen durch Beobachtung bestehender Abläufe
- hift Informationen zu finden, die der Interviewpartner nicht formulieren kann oder will
- **Formen:**
  - passiv - Videoaufnahmen, Zuschauen ohne Eingriff
  - erläuternd - Mitarbeiter erläutert Tätigkeit begleitend
  - aktiv - Analyst führt die Tätigkeiten selbst aus, wird Teammitglied



[Maciaszek 2005]

Fragebögen

- **Fragebögen** verschicken
  - zeitsparend, zeitlich entkoppelt
  - anonym (ehrlichere Antworten?)
  - viele Personen „gleichzeitig“ erreichbar
  - oft begleitend zu Interviews eingesetzt
  - passiver als Interviews (keine Rückfragemöglichkeit)



[Maciaszek 2005]

Dokumentenanalyse

- **Bestehende Dokumente** analysieren, z.B.
  - Formulare, Stellenbeschreibungen, Organisationshandbücher, Sitzungsprotokolle ... geben Auskunft darüber, wie das **organisatorische Umfeld** des neuen Systems „funktioniert“
  - Dokumentation bestehender (abzulösender?) **technischer Systeme** enthalten vielleicht (noch immer) wichtige Anforderungen und Begleitinformationen
  - Spezialzeitschriften / -Bücher über das **Tätigkeitsfeld** des Unternehmens enthalten vielleicht interessante **fachliche Zusatzinformationen** / modernere Erkenntnisse

Prototyping

- **Software-Prototyp** für Teilaspekte (z.B. GUI)
  - eingeschränkte Implementierung,
    konzentriert auf zu untersuchenden Aspekt des Systems,
  - vermittelt späteren Anwendern gute Vorstellung,
    wie sich das System z.B. „anfühlen könnte“
  - hilft, früh wertvolles Feedback zu bekommen

Lieber so? *oder so?* *oder so?* *oder so?*

Anforderungserhebung: Probleme

- **Verteiltes Wissen** über Anforderungen
  - Das Wissen kann über mehrer Quellen verteilt sein
  - steckt in den Köpfen (nicht schriftlich festgehalten)
  - Wissensträger können verschiedene Ziele haben
  - ...oder unterschiedliche Wissensstände/Ausdrucksweisen
- **Unbewusstes / implizites "Routinewissen"**
  - wird gerne bei Befragungen vergessen/übersehen,
  - ist oft für die Wissensträger schwer auszudrücken
- **Verfälschungen**
  - Wissensträger *dürfen* nötige Infos nicht herausgeben (oder glauben das; z.B. Unsicherheit wegen Vorgesetzten)
  - Wissensträger *wollen* nötige Infos nicht herausgeben
    - Manipulationsversuch, insb., falls Projektergebnis sie betrifft
    - Bewahrung von „Know-How-Monopolen“ usw.

Exkurs

Überprüfung von Anforderungen automatisieren

Unit Testing mit JUnit und eclipse

- JUnit ist ein freies Unit-Testing-Framework für Java
- „Quasi-Standard“ in Java-Welt: JUnit
  - **Testfall-Klasse** enthält eine Reihe von
  - **@Test**-Methoden. Der Methodenname sollte auf das Testziel hinweisen.
  - Mit @DisplayName("...") kann sprechendere Testbezeichnung (für Testübersicht/-protokoll) angegeben werden
  - Ausführung der Tests mit einem **TestRunner** (oft aus IDE startbar)
  - Ab **JUnit 5**: Stärkerer Gebrauch von Java8-Features (Lambda-Ausdrücke)
  - Jede Testmethode wird **isoliert** ausgeführt, Reihenfolge **nicht** garantiert (Ausführung jeder Test-Methode mit „frischer“ Instanz der Testklasse)

```
@Test
@DisplayName("optimalZahl() muss den optimalen Wert berechnen")
public void testeEtwas() {
    ...
}
```

Assert

- „Asserts“ beschreiben erwartete Zustände/Werte innerhalb eines Tests.
- Das Framework stellt in `org.junit.jupiter.api.Assertions` verschiedene statische „**`assert...`**“- Methoden bereit, um **Test-Bedingungen** zu formulieren
  - **`assertTrue(`*`ausdruck`*`), assertFalse(`*`ausdruck`*`)`**
  - **`assertEquals(`*`sollwert`*`, `*`istwert`*`)`**
  - **`assertSame(`*`objref1`*`, `*`objref2`*`), ...`**
  - **`assertNull(`*`ausdruck`*`), assertNotNull(`*`ausdruck`*`)`**
  - **`assertThrows(`*`exception`*`, `*`executable`*`)`**
  - `...`
- Optionaler Info-String als letzten Parameter hinzufügbar

```
@Test
@DisplayName("optimalZahl() muss den optimalen Wert berechnen")
public void testeEtwas() {
    ...
    Assertions.assertEquals(17, meinObj.optimalZahl(), "17 erwartet" );
}
```

Beispiel: Mittelwertberechnung

- Es sollen Summe und arithmetisches Mittel zu einer (beliebig langen) Eingabefolge von int-Zahlen bestimmt werden.

  z.B. für 1, 3, 17, 9: Summe = 30, Mittel = 30 / 4 = 7.5

- Schnittstelle der zu realisierenden Klasse:
  - **`add()`** - zum Hinzufügen einer `int`-Zahl
  - **`getSumme()`** - zum Abrufen der Summe der Zahlen
  - **`getMittel()`** - zum Abrufen des arith. Mittels

Ein Testfall

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions;
public class DatenSammlerTest {
  DatenSammler dasa=new DatenSammler();
  @Test public void testVieleWerte() {
    int i, s = 0;
    for (i=0; i < 20; i++) {
        dasa.add(i);
        s += i;
        Assertions.assertEquals(s,dasa.getSumme(),"Summenabruf");
    }
  }

  @Test public void testArithmittel() {
    dasa.add(1);
    dasa.add(2);
    Assertions.assertEquals(1.5, dasa.getMittel(), "Mittelwert");
  }
}
```

Kommt die Implementierung mit „vielen“ Werten klar?

Test für Berechnung des arith. Mittels

Implementierung

zu testen

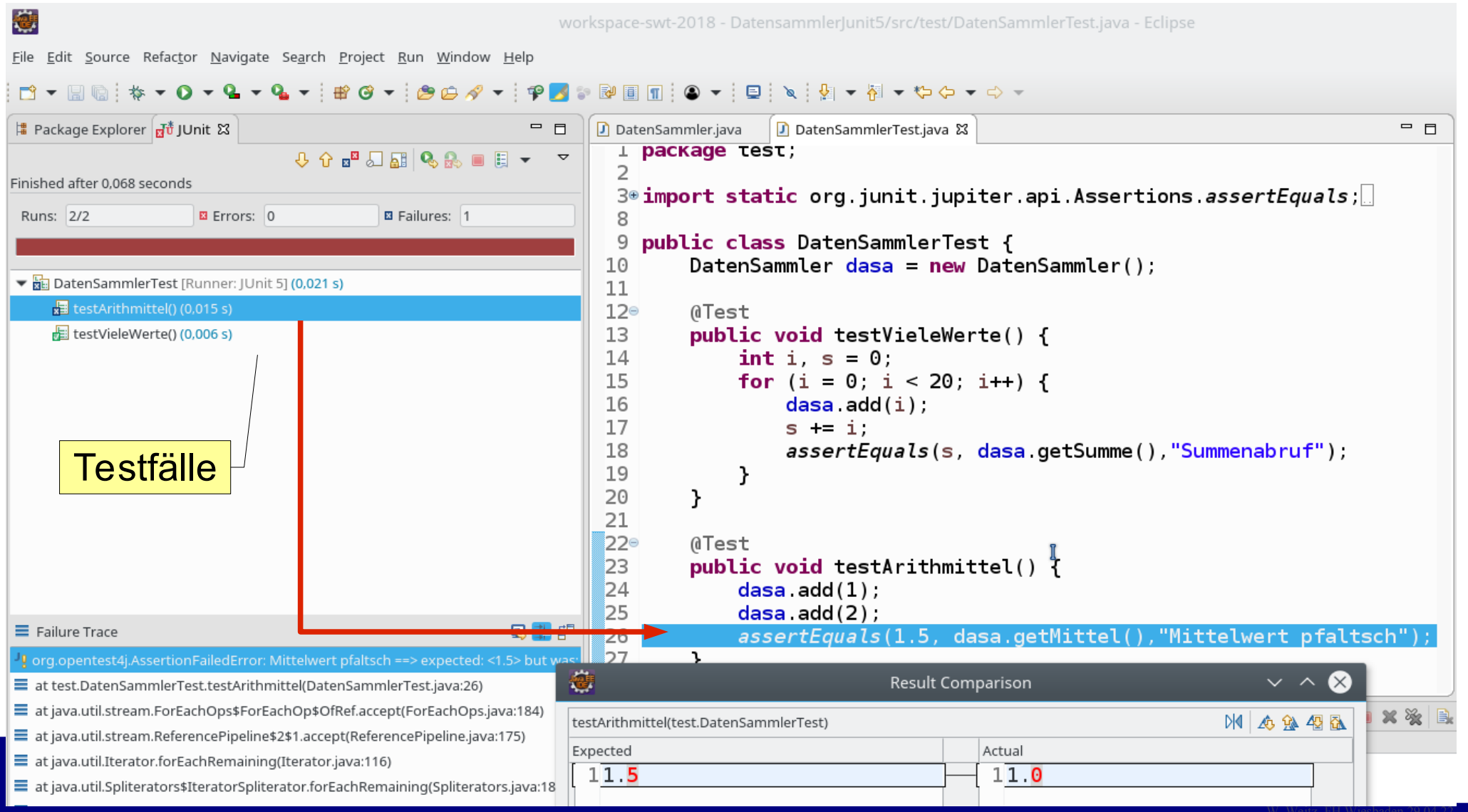
```
public class DatenSammler {
    private int sum = 0;
    private int zaehler = 0;

    public void add(int n) {
        zaehler++;
        sum += n;
    }

    public double getMittel() {
        return sum / zaehler;
    }

    public int getSumme() { return sum; }
}
```

JUnit-Unterstützung in Eclipse



Erwartete Exception testen


```
class .... {
 ...
   @Test
   @DisplayName("Test zur Ausloesung einer Exception")
   public void testExc() {
      // Dividiert durch Null, da noch keine Werte
       assertThrows(ArithmeticException.class, () -> {
          double a = dasa.getMittel();
       }, "Division durch Null bei leerem Datensammler erwartet");
   }
 ...
}
```

- Test schlägt fehl, wenn die erwartete angegebene Exception im Lambda-Block *nicht* auftritt

Timeouts vorsehen

```
import static java.time.Duration.ofSeconds;
class .... {
  @Test
  @DisplayName("Ausfuehrungsdauer max 10 Sekunden")
  public void testEndlos() {
    assertTimeoutPreemptively(ofSeconds(10), () -> {
        for (int i=1; true; i++) {
          m = dasa.getMittel();
        }
      }, "Sollte nicht mehr als 10 Sekunden brauchen");
  }
}
```

- Test wird automatisch nach Timeout mit Fehler abgebrochen

Testumgebung einrichten

```
class .... {
   @BeforeEach public void setUp() {
      // Initialisierungen vor jedem Test
     ...
   }
   @AfterEach public void tearDown() {
      // Aufräumen nach jedem Test
      ...
   }
  ...
```

- **@BeforeEach**-Methode wird vor jedem einzelnen Test,
  @AfterEach-Methode nach jedem Test ausgeführt
- Erlaubt z.B. Herstellen von für jeden Test benötigten Objektstrukturen oder andere Vorbereitungen

Allgemeine Vorbereitungen

```
class .... {
  @BeforeAll public void initialisierung() {
     // Initialisierungen einmal vor erstem Test
     ...
  }
  @AfterAll public void aufraeumen() {
     // Aufräumen einmal nach Ende des letzten Tests
     ...
  }
 ...
```

- „Globale“ Initialisierung / Aufräumen vor/nach Ausführung der Einzeltests in der Testklasse
- Mit Vorsicht zu genießen, Notwendigkeit hinterfragen