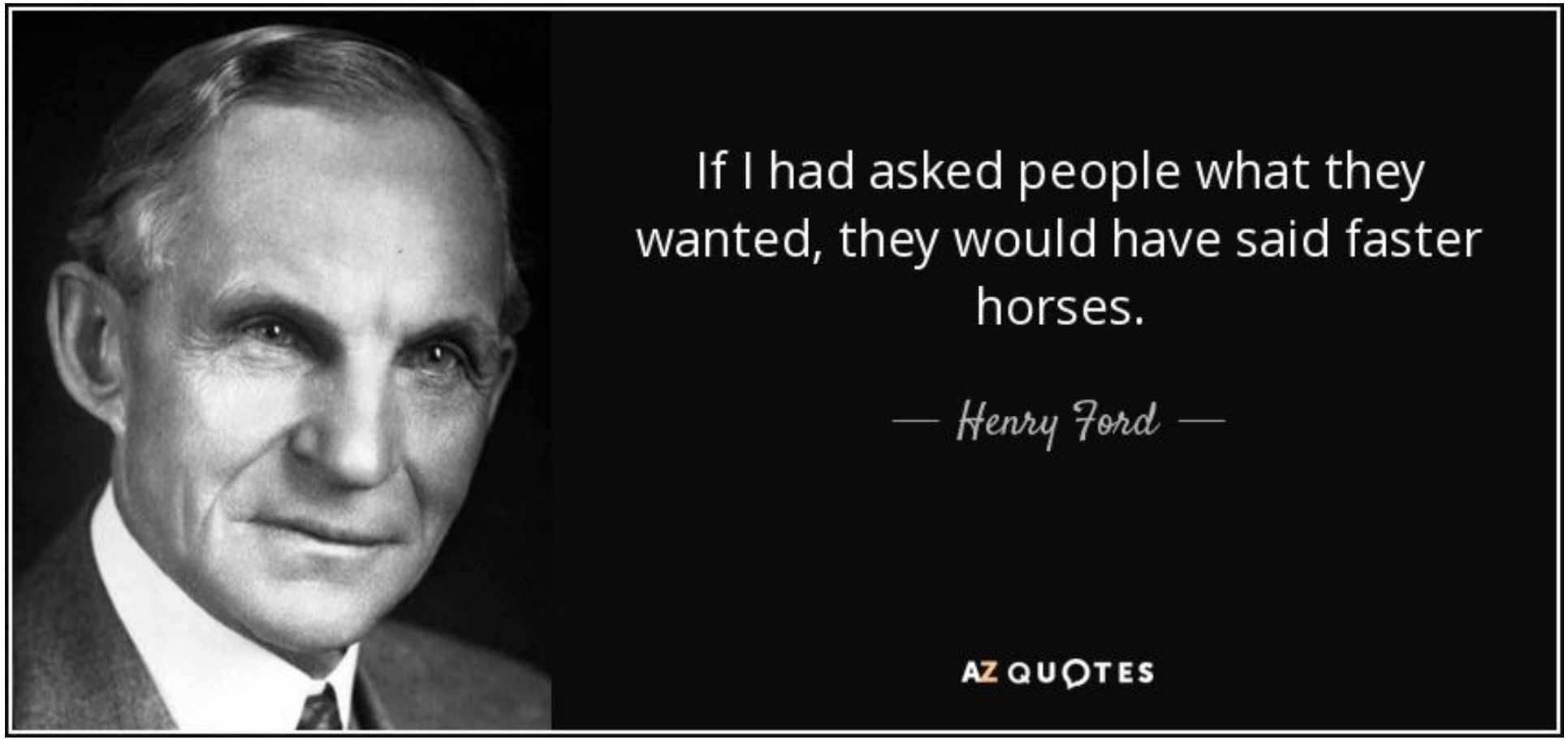
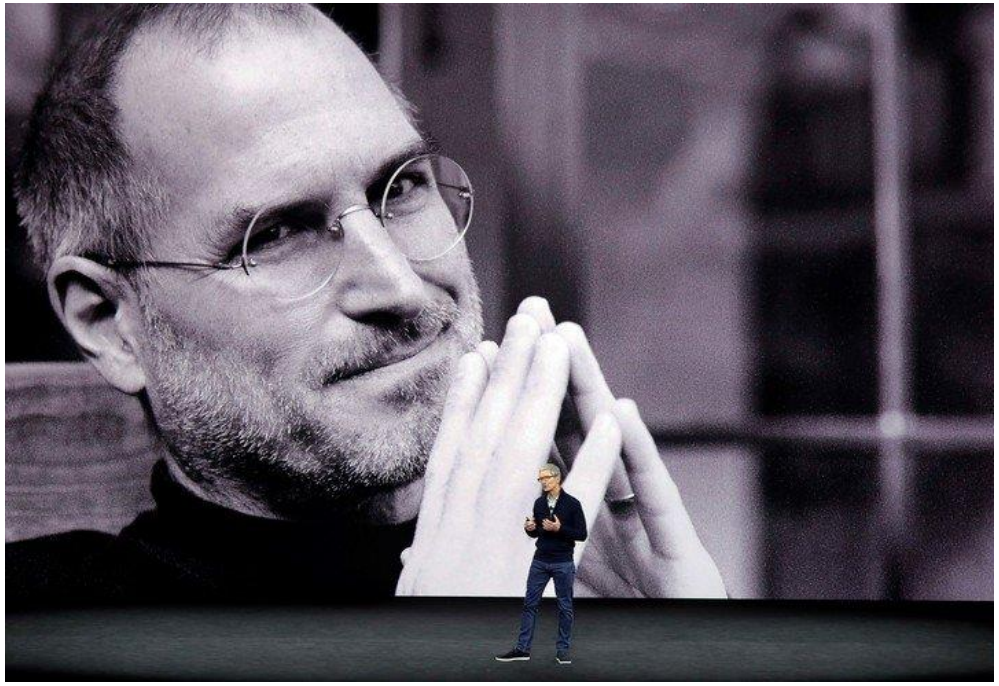
# Requirements engineering

- das Problem verstehen... -



DHBW
Duale Hochschule
Baden-Württemberg
Heidenheim



“Jobs believed a truly revolutionary product couldn't depend on customers’ needs and wants. He thought customers could not understand the value of a product until they were actually using it. ”

Real difference between Apple’s products like iMac and all others was the beauty, design and pleasure.

https://www.investopedia.com/articles/personal-finance/042815/story-behind-apples-success.asp

# Requirements Engineering

**Das Richtige tun…**

**<ist wichtiger als>**

**…etwas richtig tun…**

- Effectiveness over efficiency -

Peter Drucker (amerikanischer Ökonom, Pionier der modernen Managementlehre ), 1963:

*„It is fundamentally the confusion between effectiveness and efficiency that stands between doing the right things and doing things right. There is surely nothing quite so useless as doing with great efficiency what should not be done at all.“*

Später wurde es von ihm abgewandelt zu *„Do the **right things**, not things right…“*

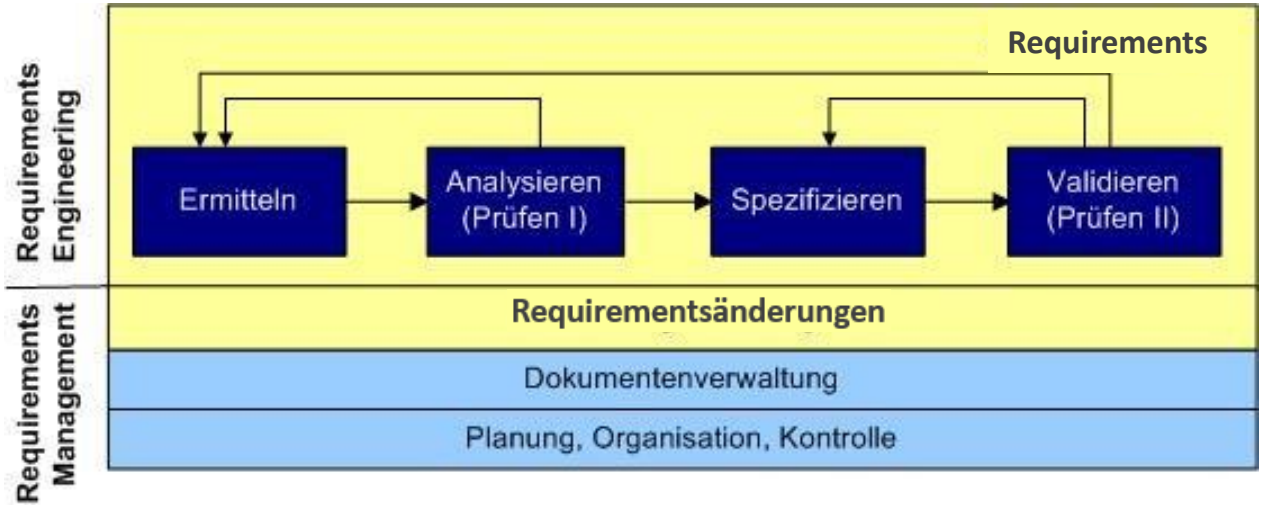
Quelle: Peter Ferdinand Drucker: Managing for Business Effectiveness, Harvard Business Review, Mai-Juni 1963, S. 53–60.

# Requirements Engineering

Requirements Engineering (Anforderungsverarbeitung) umfasst das

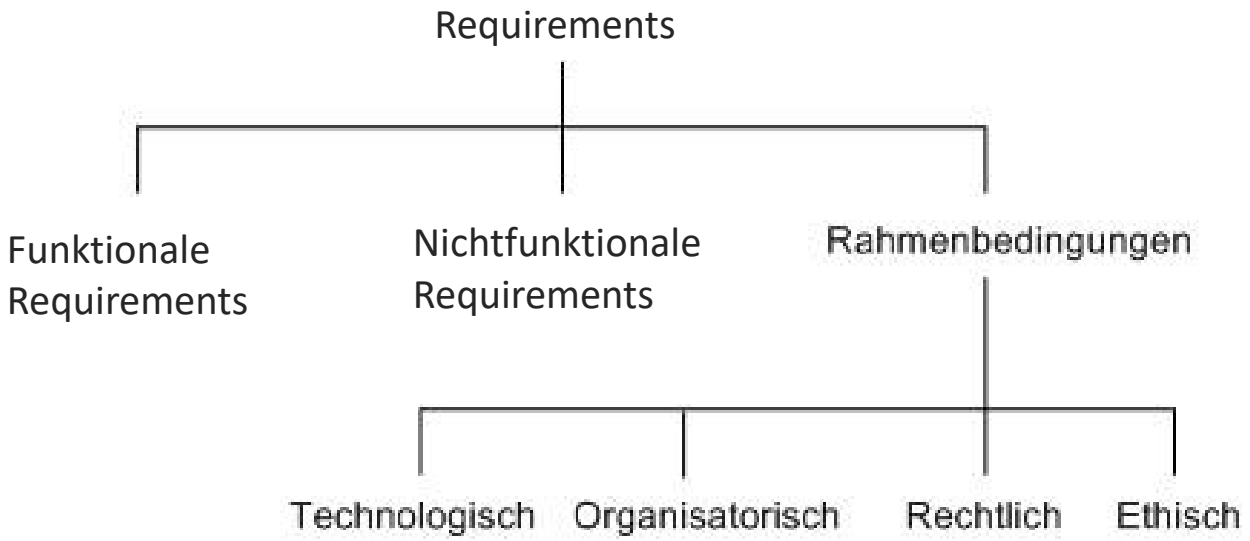
- Ermitteln,
- Analysieren,
- Spezifizieren und
- Validieren

des Kundenproblems und der Eigenschaften und Rahmenbedingungen eines (Software-)Systems, die über seinen gesamten Lebenszyklus gewünscht werden bzw. relevant sind, um dieses Problem zu lösen.



# Requirements

Requirements beschreiben die Eigenschaften, die ein Softwaresystem besitzen muss, sowie Rahmenbedingungen, die für seinen Lebenszyklus (Entwicklung, Betrieb, Wartung) gelten [IEEE Std. 610.12-1990, p. 62; Rupp et al. 2009, S. 17 f.].



# Requirements

***Funktionale Requirements*** geben an, was das Softwaresystem oder einzelne seiner Komponenten tun sollen (und was nicht).
Typischerweise werden diese Anforderungen definiert aus

- der Funktionsperspektive (Systeminput und -output, Fehlersituationen),
- aus der Datenperspektive (Datenstrukturen und Integritätsbedingungen) und
- aus der Verhaltensperspektive (Systemzustände, Zustandsübergänge, Ereignisse) [1].

***Nichtfunktionale Requirements*** geben Kriterien an für die Güte/ Qualität des Softwaresystems oder einzelner Systembestandteile.
Beispiele für solche Anforderungen sind Zuverlässigkeit, Benutzbarkeit und Performance (aus der Sicht der Systembenutzer) sowie Änderbarkeit und Portabilität (aus der Sicht der Entwickler).

[1] Pohl, Klaus: Requirements Engineering: Fundamentals, Principles and Techniques. Berlin et al.: Springer, 2010.

# Rahmenbedingungen

***Rahmenbedingungen*** sind Anforderungen, die die Realisierungsmöglichkeiten für ein Softwaresystem einschränken und nur schwer oder gar nicht geändert werden können.
Nach ihrem Ursprung lassen sich Rahmenbedingungen klassifizieren als:

- ***Technologisch***:
  die vorhandene, technische IT-Infrastruktur, in der das Softwaresystem betrieben und entwickelt werden soll

- ***Organisatorisch***:
  Aufbau- und Ablauforganisation der Einheiten, die die Software nutzen (z. B. Abteilungen mit speziellen Zuständigkeiten) bzw. entwickeln (Vorgehensmodelle für die Entwicklung, Projekttermine)

- ***Rechtlich***: einzuhaltende Gesetze und Richtlinien, z. B. hinsichtlich des Datenschutzes

- ***Ethisch***:
  Sitten des jeweiligen Kulturkreises, denen das System genügen muss, um akzeptiert zu werden
  (z.B. Anredeformen, Farben,…)

# Stakeholder

Erhoben werden die Anforderungen bei den verschiedensten Stakeholdern (auf der Suche nach dem zugrundeliegenden Problem).

***Stakeholder*** sind Personen oder Organisationen mit (positivem oder negativem) Interesse am geplanten Softwaresystem, und/ oder Einfluss auf die Produktentstehung
z. B. Geldgeber, Nutzer, Entwickler, Administratoren, die Marketingabteilung (in der Softwareindustrie) sowie Regulierungsbehörden und Kontrollgremien… bis hin zur Ehefrau des Chefarchitekten.

Bsp. für den Wunsch nach „schnelleren Pferden“:
„Wir brauchen lokal verteilte Redundanz für unser System, damit es stets verfügbar ist“
„Wir müssen uns gegen Datenverluste schützen, und deshalb jede Nacht ein automatisches Backup machen“
„Wir brauchen Loadbalancing, um uns jederzeit an einem der Server anmelden zu können“

Softwarekosten: 350.000 €

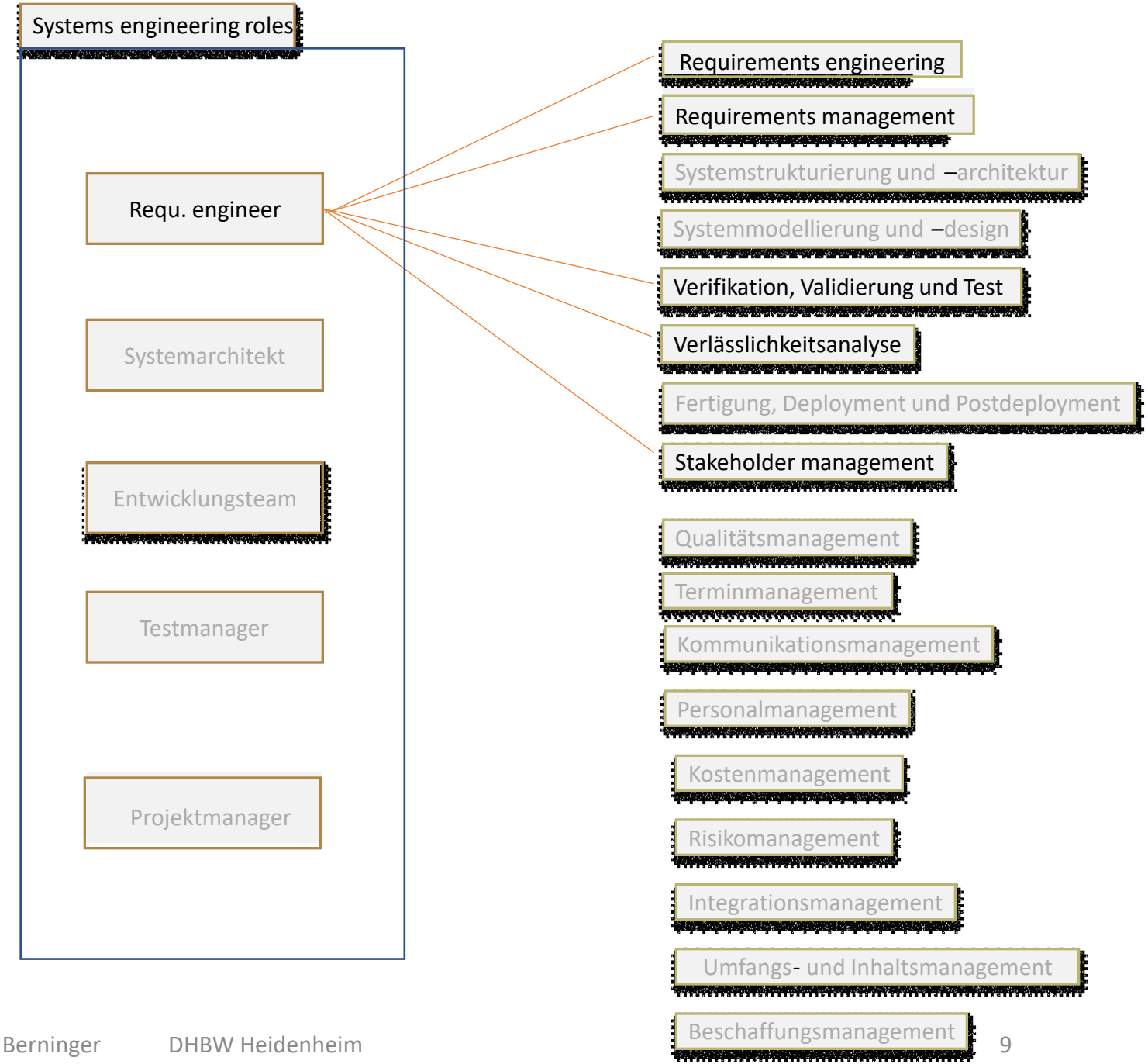
Warum? Warum? Warum?....

**Problem:** Ein instabiles Stromnetz mit häufigen kurzzeitigen Stromausfällen...

Wie wäre es mit einer USV für 50.000 €?

Die Erhebung und Bearbeitung von Requirements ist eine Ingenieurs-Disziplin –

nicht die Aufgabe des Kunden!



# Kernaktivitäten des Requirement engineering

1) ***Ermitteln:*** Herausfinden des Kundenproblems und daraus ableitbarer existierender und als auch potenzieller Anforderungen an die Lösung mittels unterschiedlicher Ermittlungstechniken.

   „Was muss getan werden“, damit ein Projekt erfolgreich ist – um die Bedürfnisse der Stakeholder zu erfüllen, die nicht an der Entwicklung beteiligt sind.

- Brainstorming
- Fokusgruppen
- Interviews (5 x why)
- Beobachtung
- Prototyping
- Anforderungsworkshops
- Umfragen

Requirements engineering: Ingenieure stellen Fragen an **Stakeholder**…

Häufig der größte Schwachpunkt in Ihren wissenschaftlichen Arbeiten!

# 1) Problem und Lösungsanforderungen ermitteln

***Anti-Pattern:***

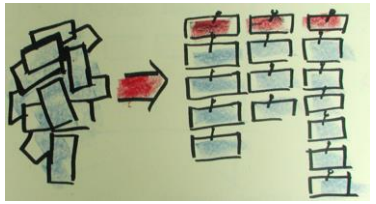
- „Das Problem besteht darin, dass die Firma xyz ihre Datenbanken effizient über einen Kafka-Bus anbinden möchte.“

➢ Lösungsoption statt dem Problem!

Häufig der größte Schwachpunkt in Ihren wissenschaftlichen Arbeiten!

# Kernaktivitäten des Requirement engineering

***2) Analysieren*** (Prüfen I): Klassifizieren und Priorisieren der ermittelten, unstrukturierten Anforderungen in Gruppen eng zusammenhängender Anforderungen - redundante, überlappende oder widersprüchliche Anforderungen werden deutlich.
Auflösung der Probleme durch Verhandlungen zwischen den verschiedenen Stakeholdern und Priorisierung von Anforderungen (*Übereinstimmung)*.



***3) Spezifizieren:*** Überführen der analysierten Anforderungen in eine Standardform (mindestens einheitliche Attribute für jede Anforderung, z.B. Anforderungsschablonen [3]), teilweise auch kompletter Aufbau des Anforderungsdokuments (z. B. IEEE/ANSI 830-1998).


***4) Validieren*** (Prüfen II): Ziel: repräsentieren die spezifizierten Anforderungen tatsächlich das, was die Stakeholder vom Softwaresystem erwarten ? Lösen sie das zugrundeliegende Problem?
-> Auf der Grundlage der Anforderungen soll das ‚richtige' Lösung (z.B. Softwaresystem) entwickelt werden.

Sommerville, Ian: Software Engineering. 9th edition, Harlow et al.: Addison-Wesley, 2011.

# 1) Problembeschreibung/ problem statement

Auch für ‚opportunities' möglich!

## PROBLEM STATEMENT WORKSHEET

| | | |
|---|---|---|
| WHO: Who does the problem impact and involve? | WHAT: What does the problem impact? What are the drivers of the problem? | WHY: Why is solving the problem important to stakeholders and the business? |
| WHERE: Where does the problem reside or have impact? | WHEN: When did the problem begin? When does the problem need to be solved by? | HOW: How was the problem created? How can the problem be solved? |

https://blog.teamairship.com/build-a-better-problem-statement

# 1 )Problembeschreibung/ problem statement

Was häufig getan wird, aber nicht dabei hilft, das Richtige zu tun:

1. Die Firma stellt mir einen konkreten Arbeitsauftrag

2. Ich versuche, diesen über ein passendes Problem zu rechtfertigen.

3. Ich prüfe am Ende nicht, ob ich das Problem wirklich gelöst habe, sondern nur, ob ich den Arbeitsauftrag korrekt erfüllt habe.

Absichern, dass ich das Richtige tue – nicht, dass ich es richtig tue!

# 1) Problembeschreibung/ problem statement

Bessere Alternative:

1. Die Firma stellt mir einen konkreten Arbeitsauftrag
2. Ich suche das wirklich zugrundeliegende Problem für diesen Auftrag.
3. Ich erarbeite mehrere Optionen für die Lösung des Problems (inkl. 1.) – nicht jede muss Software sein.
4. Ich begründe, warum ich 1. weiter untersuche – selbst wenn es nur ein Teilproblem lösen sollte.

# 1) 5 Schritte, eine Lösungsidee eines CEO auf das Problem zurückzuführen

Schritt 1: Klären, ob die Idee ein Problem oder eine Lösung beschreibt

Step 2: Bitten, den Fokus auf das Kundenproblem zu verlagern & es aufzuschreiben

Step 3: Unbekannte Gebiete in der Problembeschreibung finden

Step 4: Mindestens 3–4 Optionen für die Lösung des Problems finden und untersuchen

Step 5: Anbieten, die preferierte Lösung mit einem MVP (minimal viable product) zu testen

# 1) Bsp.: “Wir brauchen eine Knowledge Database!”

Auftrag des CEO:

„Wir haben mehrere Entwicklungsstandorte mit unterschiedlichem Produkt- und technologischem KnowHow.

Stand heute kann ich nicht jeden Entwicklungsauftrag an jeden Standort geben, ohne hohe Risiken für den Projekterfolg einzugehen.

Also: wir brauchen eine standortübergreifende Knowledge-Database!“

# 1) In 5 parallelen Gruppen – 15 min.

## PROBLEM STATEMENT WORKSHEET

| | | |
|---|---|---|
| WHO: Who does the problem impact and involve? | WHAT: What does the problem impact? What are the drivers of the problem? | WHY: Why is solving the problem important to stakeholders and the business? |
| WHERE: Where does the problem reside or have impact? | WHEN: When did the problem begin? When does the problem need to be solved by? | HOW: How was the problem created? How can the problem be solved? |

https://blog.teamairship.com/build-a-better-problem-statement

# 1) Bsp.: “Wir brauchen eine Knowledge Database!”

Wann gilt das Problem als gelöst?

- „Wenn ich jede Entwicklungsaufgabe an einen beliebigen Standort geben kann“
- „Wenn ich nicht gleichzeitige unbearbeitete Aufgaben und irgendwo freie Kapazität habe“

Lösungsoptionen:

-

-

-

-

-

# 3) und 4) Spezifizieren und Validieren von Requirements User stories

Format einer User story (Template):

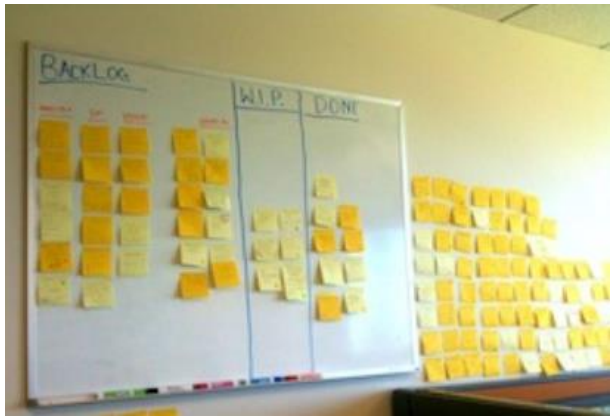

Vorderseite der Story-board Karte:

User Story Template

As a [user role]

I want to [desired feature]

so that [value/benefit]



# 3) und 4) Spezifizieren und Validieren von Requirements
## User stories

Anders als mit **User stories** werden Anforderungen professionell NICHT beschrieben!

(oder aufgelistet)
(oder zitiert)
(oder geclustert)
(oder ins Backlog eingetragen)
(oder …)

# 3, 4) Spezifizieren und Validieren von Requirements

## User stories

Als <Rolle> möchte ich <etwas tun können> so dass < ich … verbessere/ erleichtere/ ...>

Aber welche Qualität erwarte ich?

Sind da

unausgesprochene, implizite, verborgene, nichtfunktionale Requirements (**NFRs**),

die die User story detaillieren?

Unausgesprochen, aber Akzeptanz-/ Qualitäts-/ Abnahmekriterien?

# 3,4) Nichtfunktionale Requirements

**Nichtfunktionale Requirements (NFRs):**

- Bestimmen die Qualität des Produkts (CTQs = Criticals to quality) und/ oder
- Bestimmen die Performance des Produkts und seiner Features und/ oder
- Schränken Entwicklungsansätze und Prozesse ein und/ oder
- Sind Architekturtreiber und/ oder
- Listen Betriebs- und Umgebungsbedingungen durch Benennung von zu erfüllenden Normen und Richtlinien

Dean Leffingwell, 2011: „Don‘t forget the ‚ilities‘!“

# 3,4) Klassifikation von NFR

ISO/ IEC 25000 „Software Engineering – Software product quality requirements and evaluation“,
Qualitätsmerkmale in ISO/ IEC 25010

Quality
- **Availability** = Generelle Verfügbarkeit über die Zeit in % (99,9 % = 8h:45‘ /a)
- **Reliability** = Korrektes Antwortverhalten in %
- **Productivity/ Maintainability** = Schwierigkeit, ein Produkt zu erweitern oder Bugs zu fixen
- **Portability** = Möglichkeit des Betriebs auf verschiedenen Plattformen
- **Usability** = User experience bei der Benutzung des Systems
- Security
  - **Safety** = Risikopotential durch das Systemverhalten für die Umgebung
  - **Integrity** = Schutz interner Daten gegen unerwünschte Veränderung
  - **Confidentiality** = Schutz interner Daten gegen unauthorisierten Zugriff

Performance
- **Scalability** = Möglichkeit, die Performance des Systems durch zusätzliche Ressourcen zu steigern
- **Capacity** = Durchsatz in Relation zu bestimmtem Ressourcenausbau

Constraints
- Legal compliance = Einhaltung von Gesetzen
- Domain guidelines = Einhaltung von Regeln und Guidelines
- Schedule & budget constraints = Termin- und Budgetvorgaben

etc. p.p. …

# 3,4) Beschreibung von NFR durch Akzeptanzkriterien

User story…

… mit Akzeptanzkriterien (Rückseite der Story-Board-Karte)

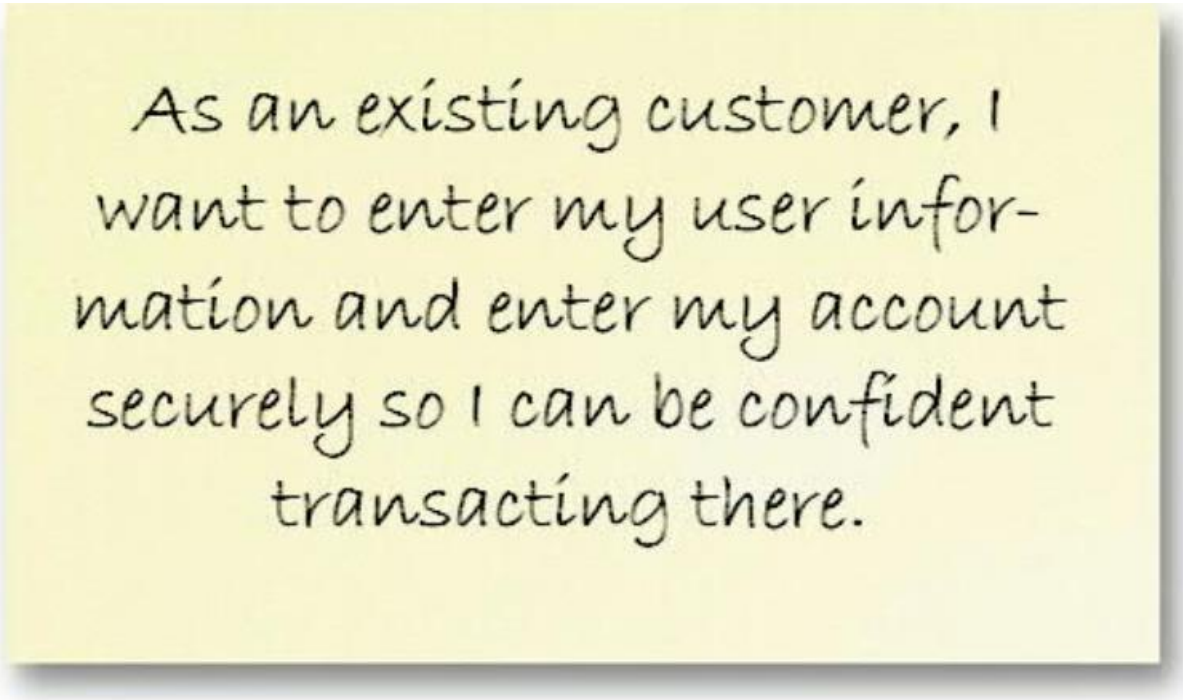
- Gute Akzeptanzkriterien helfen, ein System “Funktioniert wie gewünscht” zu liefern
  statt “Funktioniert wie codiert”

- = Satz von Aussagen, jede mit einem klaren Ergebnis für pass/ fail, die sowohl funktionale (e.g., minimale vermarktbare Functionalität) als auch nichtfunktionale (e.g., minimale Qualität) Requirements beschreiben.

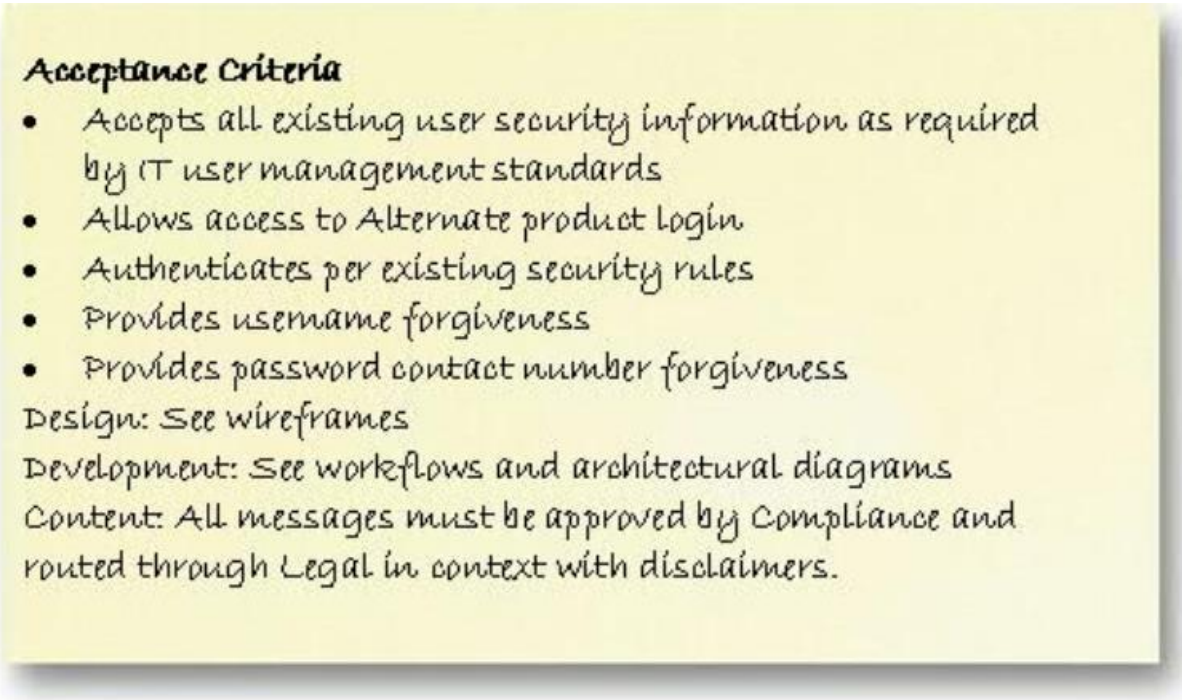
  Diese Requirements vertreten “Zufriedenheitsbedingungen”.

  Es gibt keine partielle Akzeptanz: ein Kriterium ist erfüllt oder nicht.

# 3,4) Beschreibung von Requirements
## User stories




Rückseite der User story - Karte:



# 3,4) Akzeptanzkriterien

Akzeptanzkriterien erfragen:

1. Stichworte in der User story (Verb, Substantiv, Objekt) nutzen
2. Fragen aus einem Fragenkatalog: Wer? Wann? Wie? Wie oft? Wie viele parallel? Wie lang nach…? Wie kontrollieren, ob…? Was soll geschehen, wenn… ? Was könnte verhindern, dass …?
3. Antworten mit dem Team finden, prüfen, diskutieren…
4. Akzeptanzkriterien aus den Antworten generieren
5. Die Basis für Testfälle erstellen

# 3,4) Requirements Engineering Sample: Spendensammlung

Problem:

- 13% der erwachsenen Weltbevölkerung sind Analphabeten.

Lösungsoption/ User story:

- Als Big Spender möchte ich möchte Spenden einsammeln, um den Analphabetismus reduzieren zu können.

Mission:

- Ich möchte 2 Mio € sammeln für ein Alphabetisierungsprogramm
- Ich möchte so viele potentielle Spender erreichen wie möglich

Was kann Leute motivieren, mein Vorhaben mit Spenden zu unterstützen?

1. Sie müssen die Chance bekommen, sichtbar zu werden
2. Sie sollen in Wettbewerb zueinander treten können
3. Die Spendenhöhe soll öffentlich sein
4. Die Teilnahme soll herausfordernd sein und Spaß machen

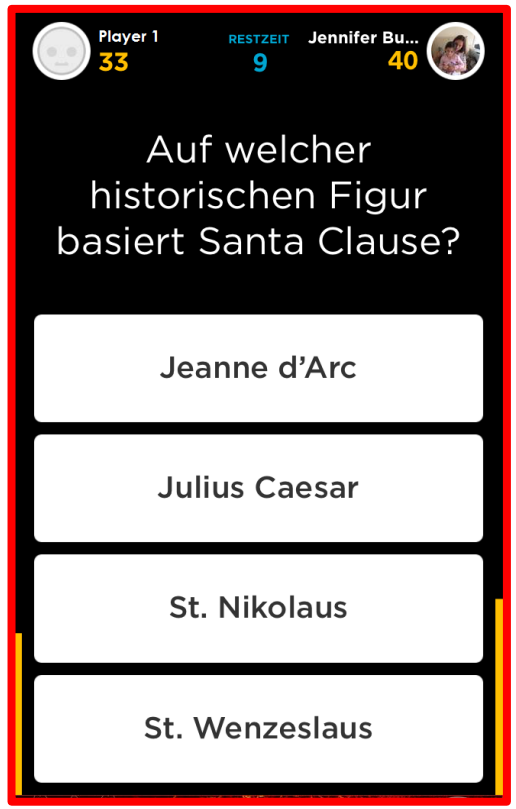
# 3,4) Konkrete Lösungsoptionen analysieren

Brainstorming workshop -> mögliche Lösungsoptionen:

a) Ice bucket challenge
b) Karaoke-Plattform
c) Quiz App

| | a) | b) | c) |
|---|---|---|---|
| Anziehungskraft | ✓ | ✓ | ✓ |
| Sichtbarkeit des Spenders | ✓ | ✓ | ✓ |
| Wettbewerb der Spender | ✓ | ✓ | ✓ |
| Sichtbarkeit der Spende | ✓ | ✓ | ✓ |
| Spaßfaktor | ✓ | ✓ | ✓ |
| Blaming-Risiko | - | - | ✓ |
| Summe | 5 | 5 | 6 |

Fig.: Focus-Area-Matrix, um Lösungs-optionen zu vergleichen



# User stories für die Lösungsoption „Quiz-App“

1. Als *Benutzer* möchte ich *vordefinierte Fragen beantworten*, so dass *ich mein Wissen verbessern kann*.
2. Als *Benutzer* möchte ich*, dass das System meinen Highscore verwaltet*, so dass *meine Ergebnisse sichtbar werden*.
3. Als *Benutzer* möchte ich *das sortierte Ranking aller Spieler sehen*, so dass *ich meine Position prüfen kann.*
4. Als *Benutzer* möchte ich *meinem Profil einen Avatar hinzufügen*, so dass *ich durch ihn repräsentiert werde.*
5. Als *Benutzer* möchte ich *eine 5€-One-Click-Spende machen können*, so dass *ich das Projekt unterstützen kann*.
6. Als *Benutzer* möchte ich *dass sich die Kleidung meines Avatars automatisch der Spendenhöhe anpasst*, so dass *mein Spendenlevel sichtbar wird.*
7. Als *Anbieter* möchte ich *5% an jeder Spende verdienen*, so dass *meine Unkosten gedeckt sind.*
8. …

Wüssten Sie jetzt genug, um die App zu programmieren?

# NFRs aus User stories ermitteln

**User story:** Beschreibt Funktionalität

**Team questions:** Das Team stellt Fragen bezüglich der Qualitätsanforderungen der User story

**Non-functional Acceptance criteria:**

Antworten auf Qualitätsfragen, in Form von Akzeptanzkriterien.

**Test cases:** Für jedes Akzeptanzkriterium mindestens einen Testfall, der enthält:

Vorbedingung

Testschritte

Erwartetes Ergebnis

Benutzte Testdaten

# NFRs aus User stories ermitteln (Beispiel)

**1. User story:** **US 1:** Als *Benutzer* möchte ich *vordefinierte Fragen beantworten*, so dass *ich mein Wissen verbessern kann*.

**Teamfragen:** F1.1: Wie viele Quizfragen soll das System anbieten und verwalten können (Menge)?
F1.2: In welcher Zeit muss der User antworten (zuverlässiger Timeout)?

**Akzeptanzkriterien:** A1.1: Das System soll 10.000 verschiedene Fragen anbieten.
A1.2: Die maximale Zeit, um eine Frage zu beantworten, soll 30 Sekunden sein.

**Testfälle:** T1.2: Das Timeout für die Antwortzeit messen:

| | |
|---|---|
| Vorbedingung: | Die App läuft, der User ist angemeldet. |
| Testschritte: | 1. Starte Timer, stelle Frage<br>2. Stoppe den Timer, sobald die Zeit für die Beantwortung abgelaufen ist |
| Erwartetes Ergebnis: | Die gemessene Zeit liegt zwischen 30…31 Sekunden |
| Benutzte Testdaten: | User: Mustermann, Password: xxx |

# NFRs aus User stories ermitteln (Beispiel)

1. **User story:** **US2:** Als *Benutzer* möchte ich*, dass das System meinen Highscore verwaltet*, so dass *meine Ergebnisse sichtbar werden*.

**Teamfragen:**
F2.1: Nach welcher Zeit soll der Highscore aktualisiert sein (Response time?)
F2.2: Wer darf meinen Highscore sehen (Security)?

**Akzeptkriterien:**
A2.1: Der Highscore soll innerhalb von 2 Sekunden aktualisiert sein.
A2.2: Nur angemeldete Spieler sollen meinen Highscore sehen können, und nur meinen Alias-Namen.

**Testfälle:**
T2.1: Zugangsprüfung zu den Highscorewerten
Vorbedingung: Die App läuft, User ist angemeldet.
Testschritte: 1. Ein anderer User verbessert seinen Highscore – automatisch Timer starten.
2. Timer stoppen bei Einblendung des Ergebnisses.
Erwartetes Ergebnis: Timerwert ist < 2 Sekunden.

T2.2: Zugangsprüfung zu den Highscorewerten
Vorbedingung: Die App läuft, Benutzer ist angemeldet.
Testschritte: 1. Drei andere Benutzer verbessern ihren Highscore.
Erwartetes Ergebnis: Highscore der 3 Benutzer ist nur unter ihren Alias-Namen sichtbar.

# Requirements (backlog) management

Verantwortliche(n) Stakeholder für die Entscheidung über das Ranking festlegen

Stakeholder bei Backlogänderungen informieren, zu regelmäßigen Meetings einladen

Beziehungen zwischen Requirements, Produktbacklog, Releasebacklog, Sprint backlog festlegen

Verhältnis zwischen Requirements, Bugs und Technical debt festlegen

Wie kommt man von den Requirements zum Scope eines Releases (Scope management)?

# Requirement backlog management

Agile Methoden:

Time-boxed Ansatz für die Deadline der Projektlieferung

**plus**

Wiederholte Implementierungssprints mit Neubewertung des Backlogrankings und kurzfristiger Nominierung der Requirements für den nächsten Sprint

Wasserfallansatz :

Frühe Vereinbarung des Projektumfangs (Requirements) **und** Liefertermin

# Einbeziehung der Stakeholder ins RM

CCB (Change Control board) etablieren, mit relevanten Stakeholdern

Regelmäßige Meetings vereinbaren

Alle anstehenden Änderungswünsche am Projektumfang diskutieren (auch Behebung von Bugs sind Änderungen!), um das Backlog ranking anzupassen