1. Einführung: Die Schaukel

Das Projekt:

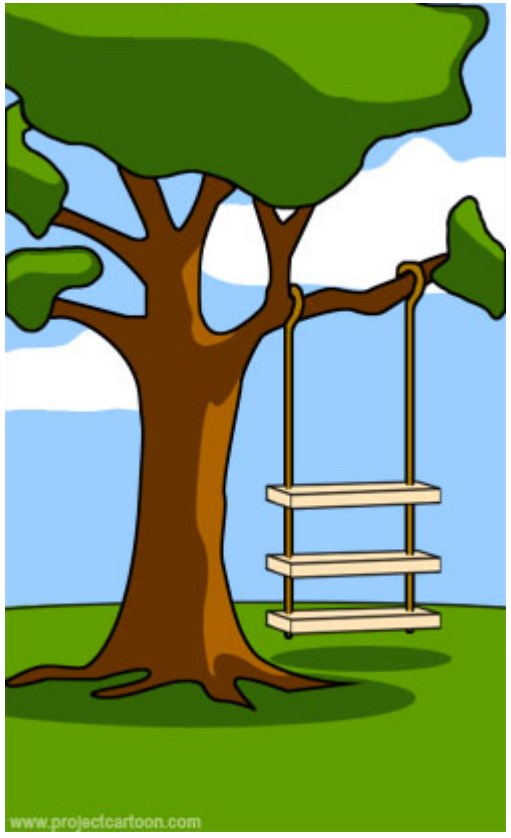
Eine Familie möchte eine Schaukel für ihre beiden Kinder haben.

Die grundlegenden Anforderungen sind:

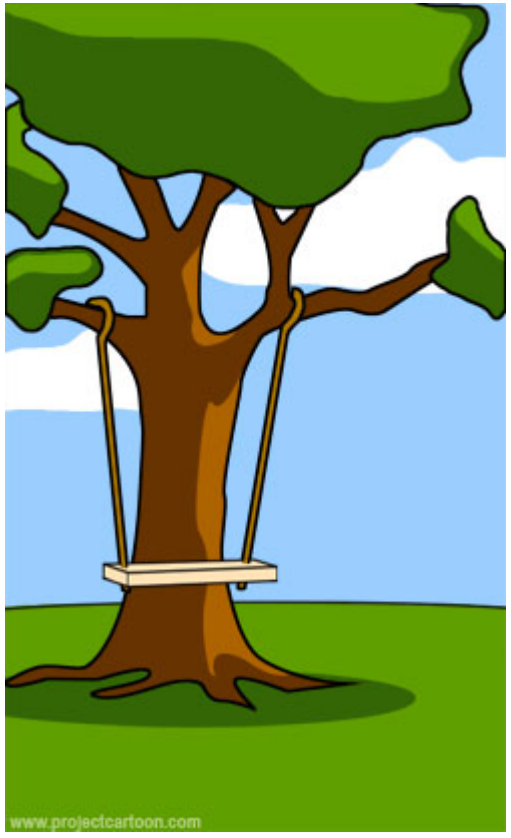
- Die Schaukel kann von einem Kind oder beiden Kindern gleichzeitig genutzt werden.
- Die Schaukel soll robust sein.

> Die folgenden Illustrationen stammen aus:
> Tree Swing Cartoon. http://www.projectcartoon.com/gallery. Nicht mehr verfügbar. Letzter Zugriff: 14. März 2013.
> Siehe auch: https://de.wikipedia.org/wiki/Tree_swing_cartoon

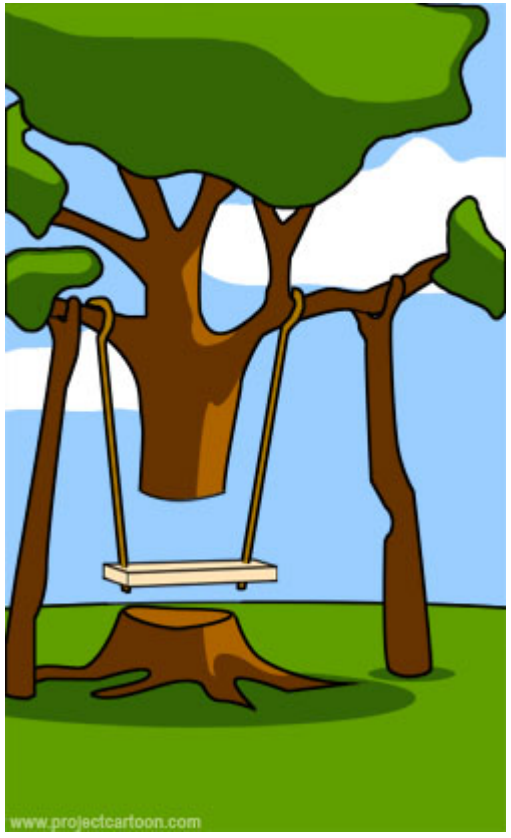
1.1 Was der Kunde erklärte



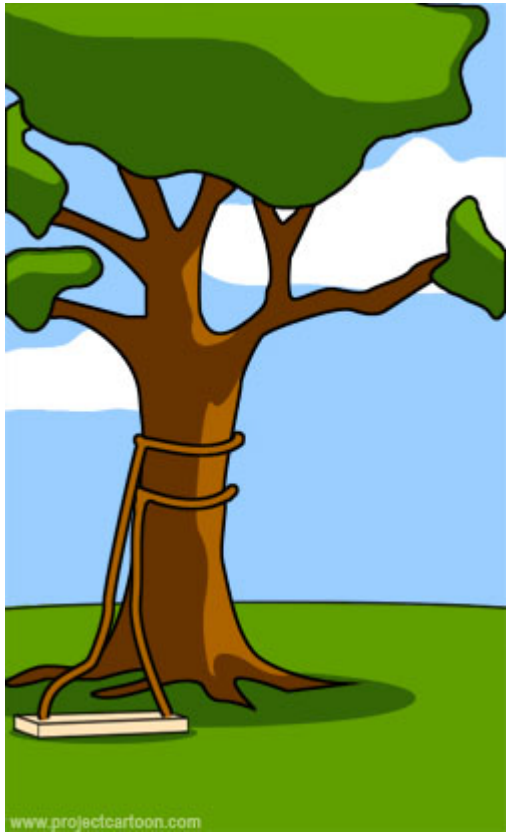
1.2 Was der Projektmanager verstand



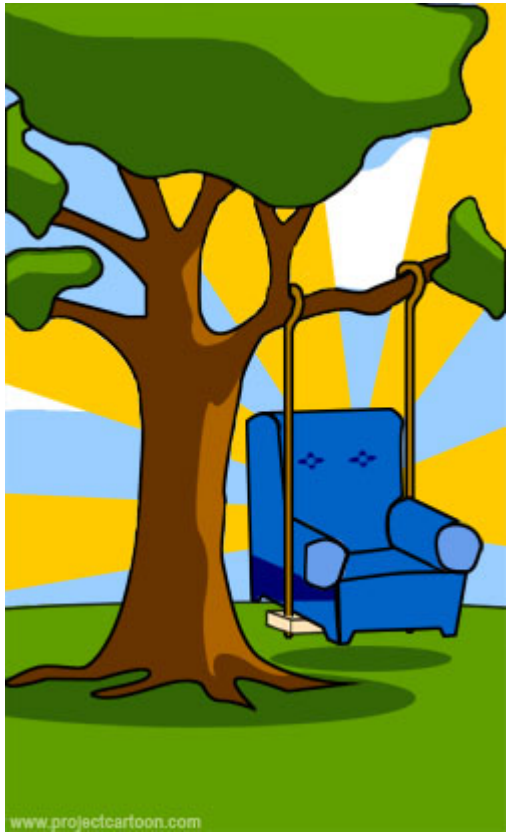
1.3 Was der Analyst interpretierte



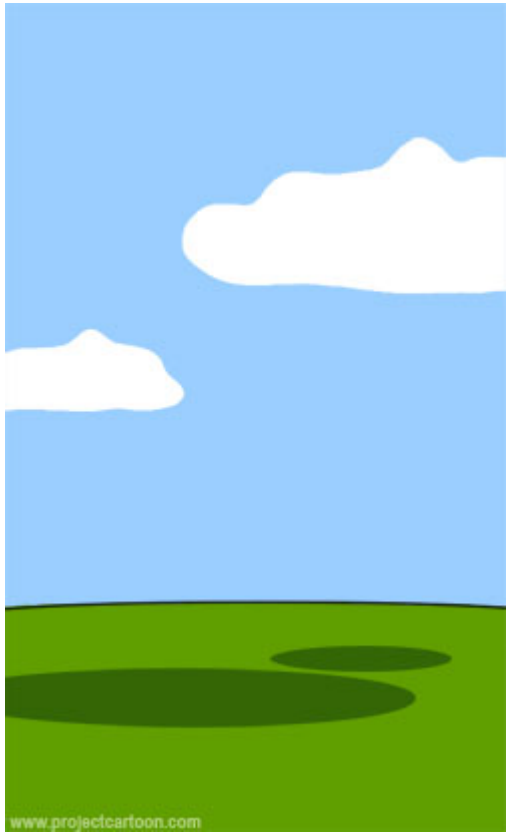
1.4 Was der Programmierer implementierte



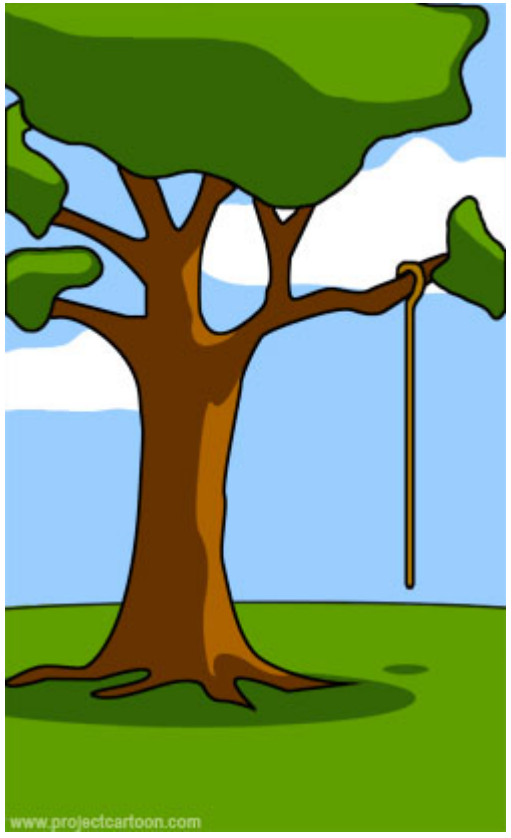
1.5 Was der Berater verkaufte



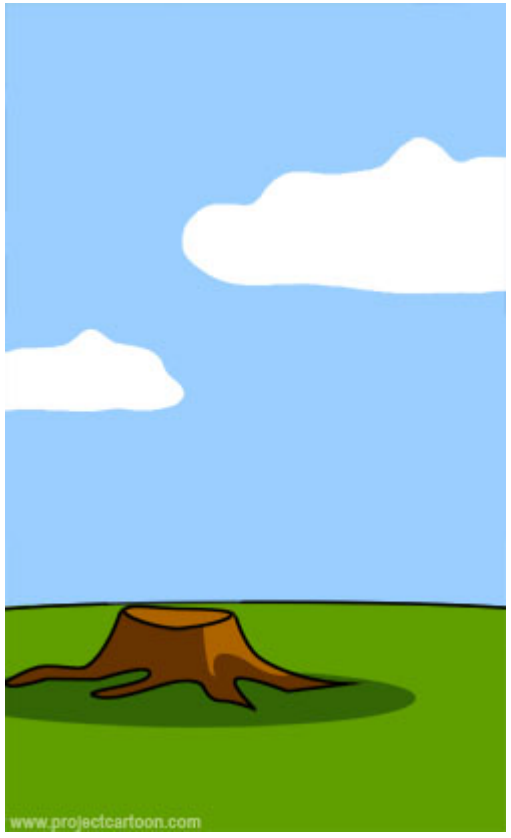
1.6 Was dokumentiert wurde



1.7 Was beim Kunden installiert wurde



1.8 Was der Support leistete



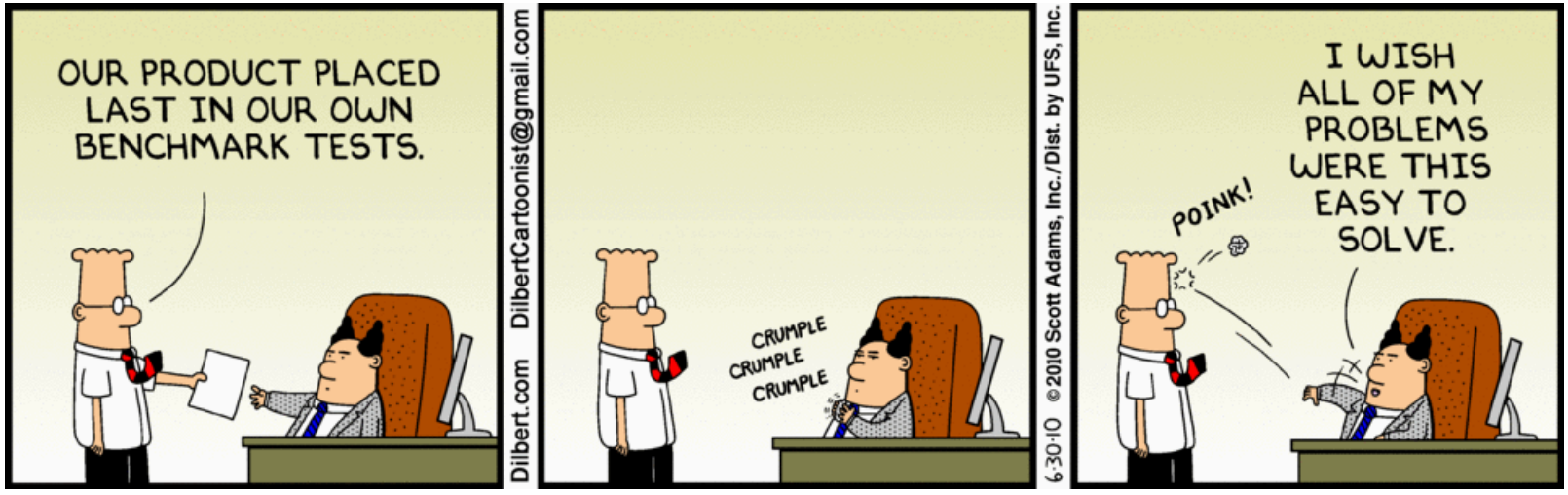
1.9 Was vom Marketing beworben wurde

1.10 Was der Kunde gebraucht hätte

2. Software Engineering ist ...

... jede Aktivität, bei der es um die Erstellung oder Veränderung von Software geht, soweit mit der Software Ziele verfolgt werden, die über die Software selbst hinausgehen.

> Ludewig und Lichter, 2023. *Software Engineering – Grundlagen, Menschen, Prozesse, Techniken.* 4. Auflage, dpunkt.verlag, Heidelberg.



> http://dilbert.com/strips/comic/2010-06-30/ Zugriff: 28. Mai 2022. Nicht mehr verfügbar

2.1 Software Engineering beinhaltet …

Kernprozesse	Unterstützungsprozesse
Anforderungserfassung	Projektmanagement
Softwarearchitektur	Dokumentation
Programmierung	Software-Metrik (Messung)
Modultests	Konfigurationsmanagement
System- und Abnahmetest	Softwareeinführung
…	…

und viele Prozesse mehr: https://de.wikipedia.org/wiki/Softwaretechnik

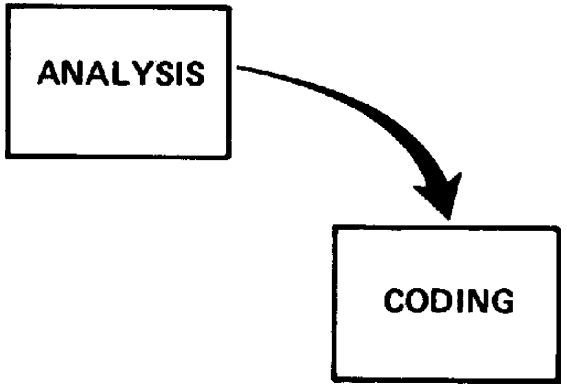
3. Entwicklungsprozess

Der Software-Entwicklungsprozess (SEP) dient der Steuerung der Softwareentwicklung und ist ein Vorgehensmodell oder einfacher ausgedrückt: Der SEP ist der grundlegende Plan.

Wichtige Unterscheidungsmerkmale zwischen SEPs sind:

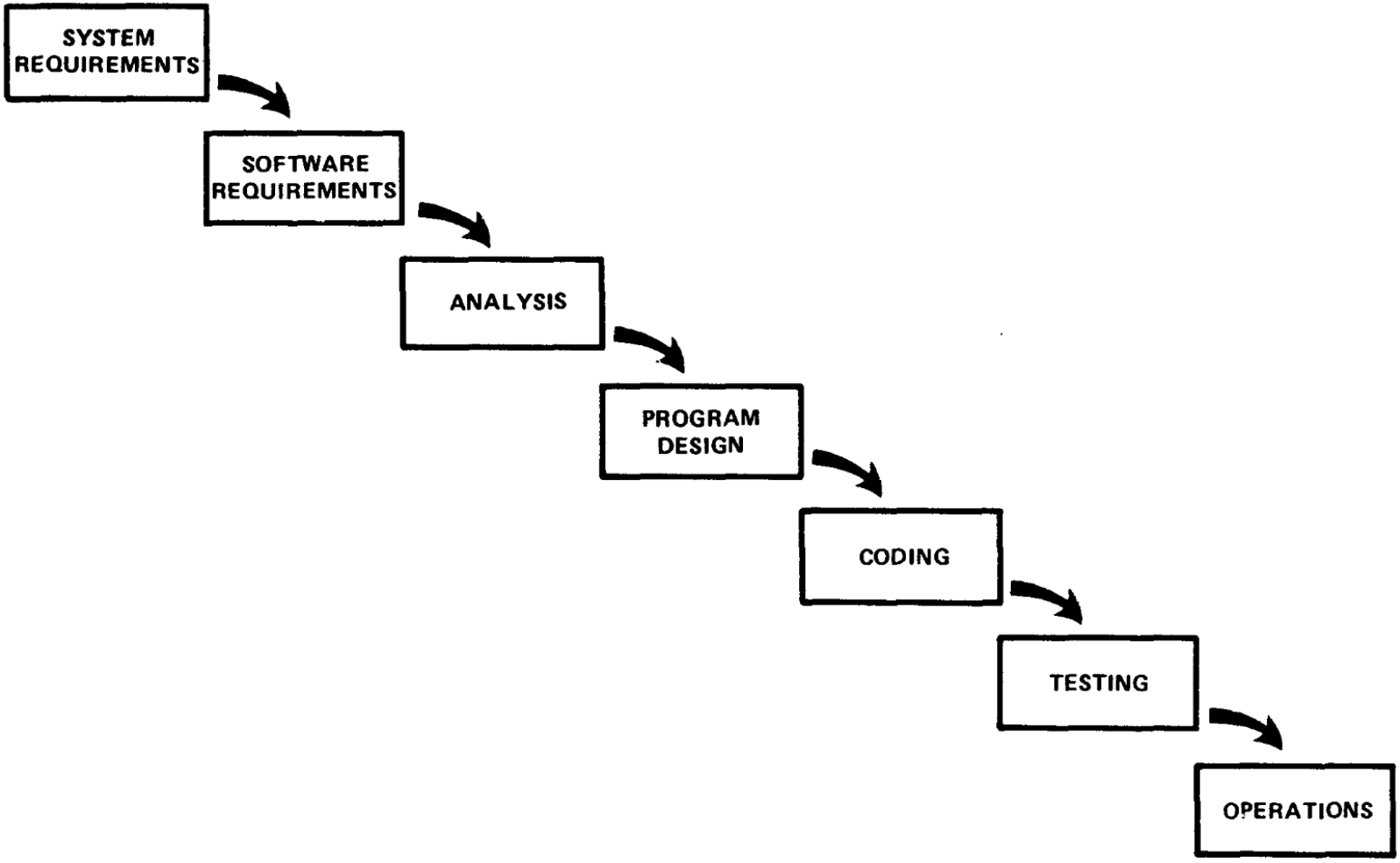
Sequenziell ⇔ Iterativ und Schwergewichtig ⇔ Agil

Das minimale Vorgehensmodell:



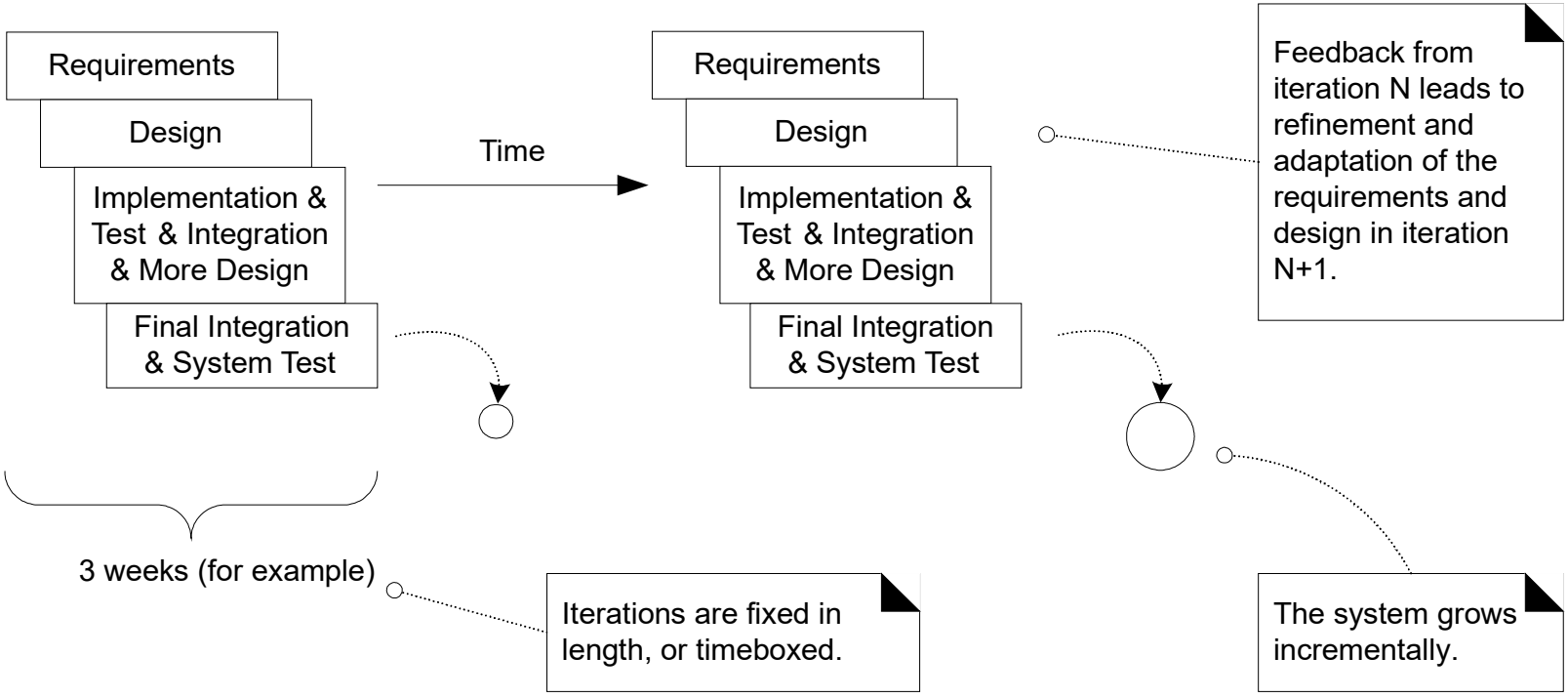
Royce, W. W., 1970: *Managing the Development of large Software Systems.* Proc. IEEE WESCON. August 1970. pp.1-9

3.1 Das Wasserfallmodell



> Royce, W. W., 1970: *Managing the Development of large Software Systems.* Proc. IEEE WESCON. August 1970. pp.1-9

3.2 Iterativ-inkrementelles Vorgehensmodell

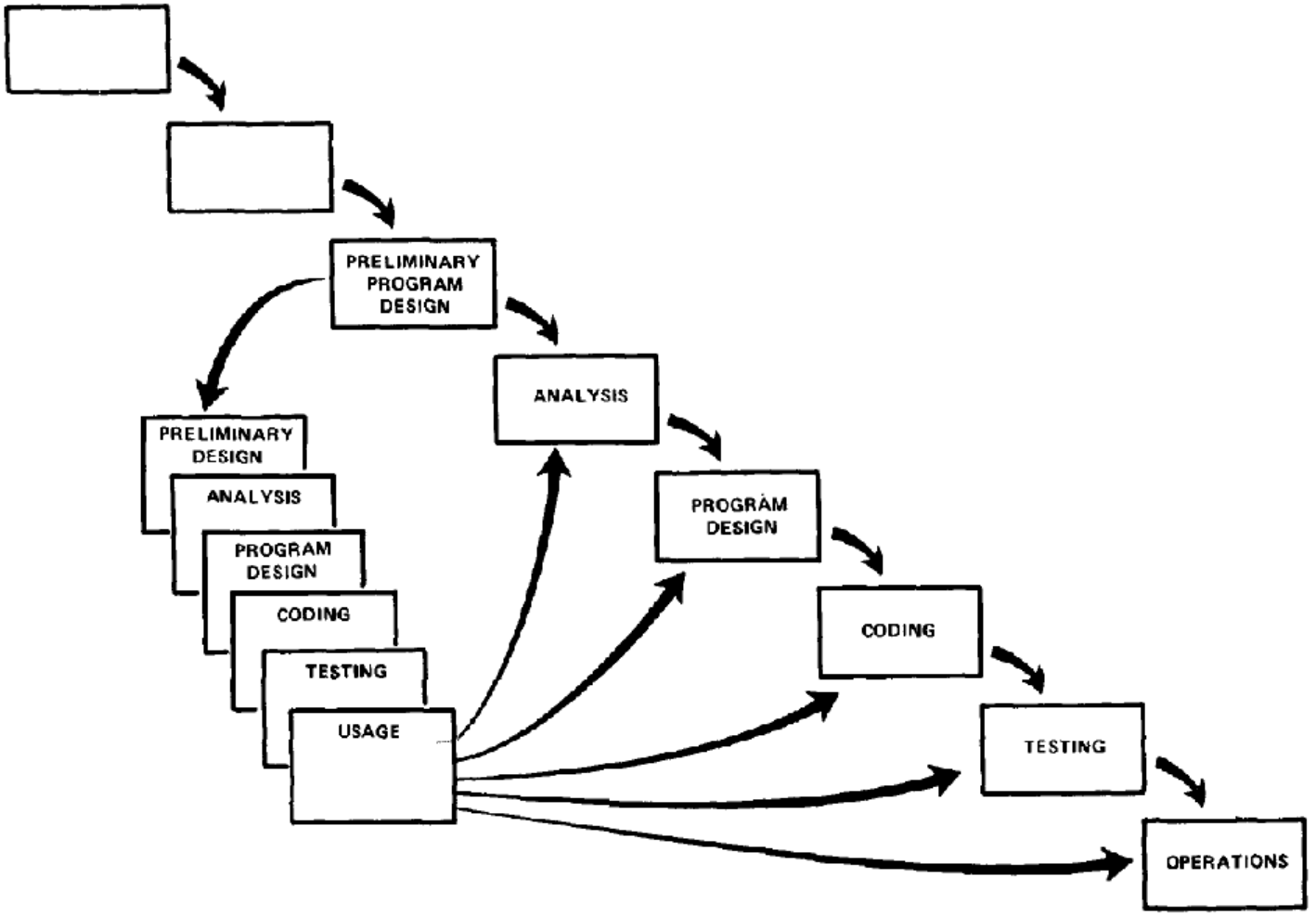


Larman C., 2004. *Applying UML and Patterns*. 3rd ed. Upper Saddle River, NJ: Pearson: Fig. 2.1

3.2.1 Eigenschaften einer Iteration

- Vorgegebene, feste zeitliche Länge
- Fügt einen definierten Satz von Funktionalitäten hinzu
- Endpunkt ist ein lauffähiges, getestetes System
- Rückkopplungseffekt aus vorhergehender Iteration

3.3 Das Wasserfallmodell – revisited



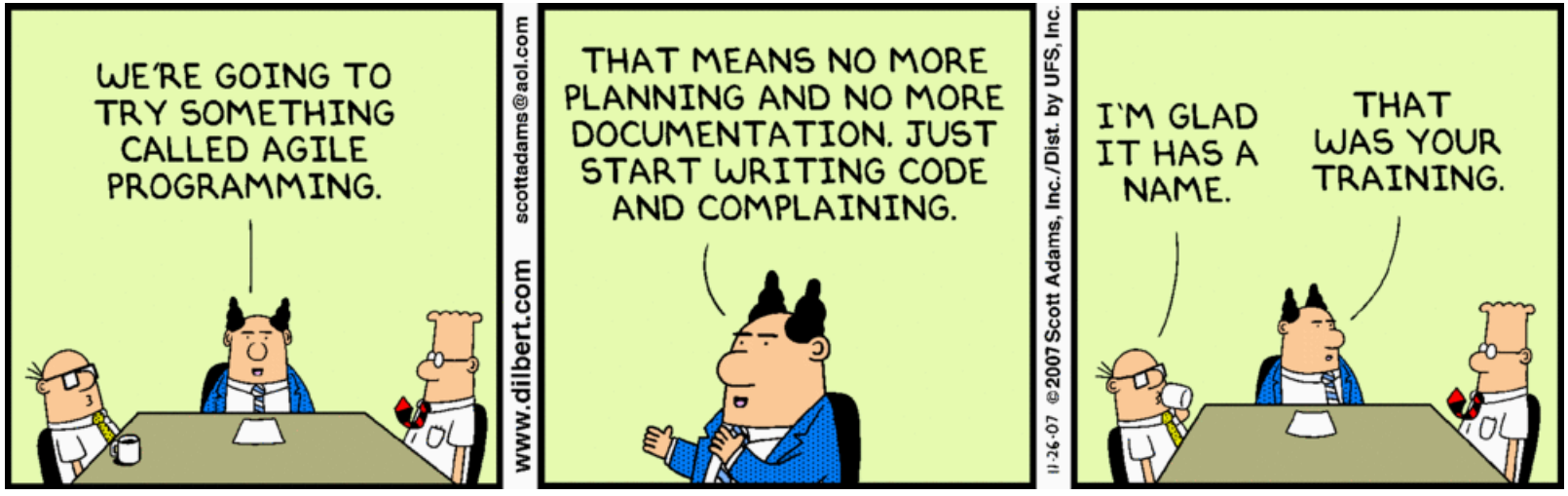
Royce, W. W., 1970: *Managing the Development of large Software Systems.* Proc. IEEE WESCON. August 1970. pp.1-9

3.3.1 Die Agile Softwareentwicklung …

ist eher eine Einstellung als eine Methode:

- Individuen und Interaktion ⇔ Prozesse und Werkzeuge
- Lauffähige Software ⇔ Übermäßige Dokumentation
- Einbinden des Kunden ⇔ Vertragliche Regelungen
- (Positives) Reagieren auf Änderungen ⇔ Planerfüllung

Das agile Vorgehen führt zwangsläufig zu einem iterativ-inkrementellem Vorgehensmodell.



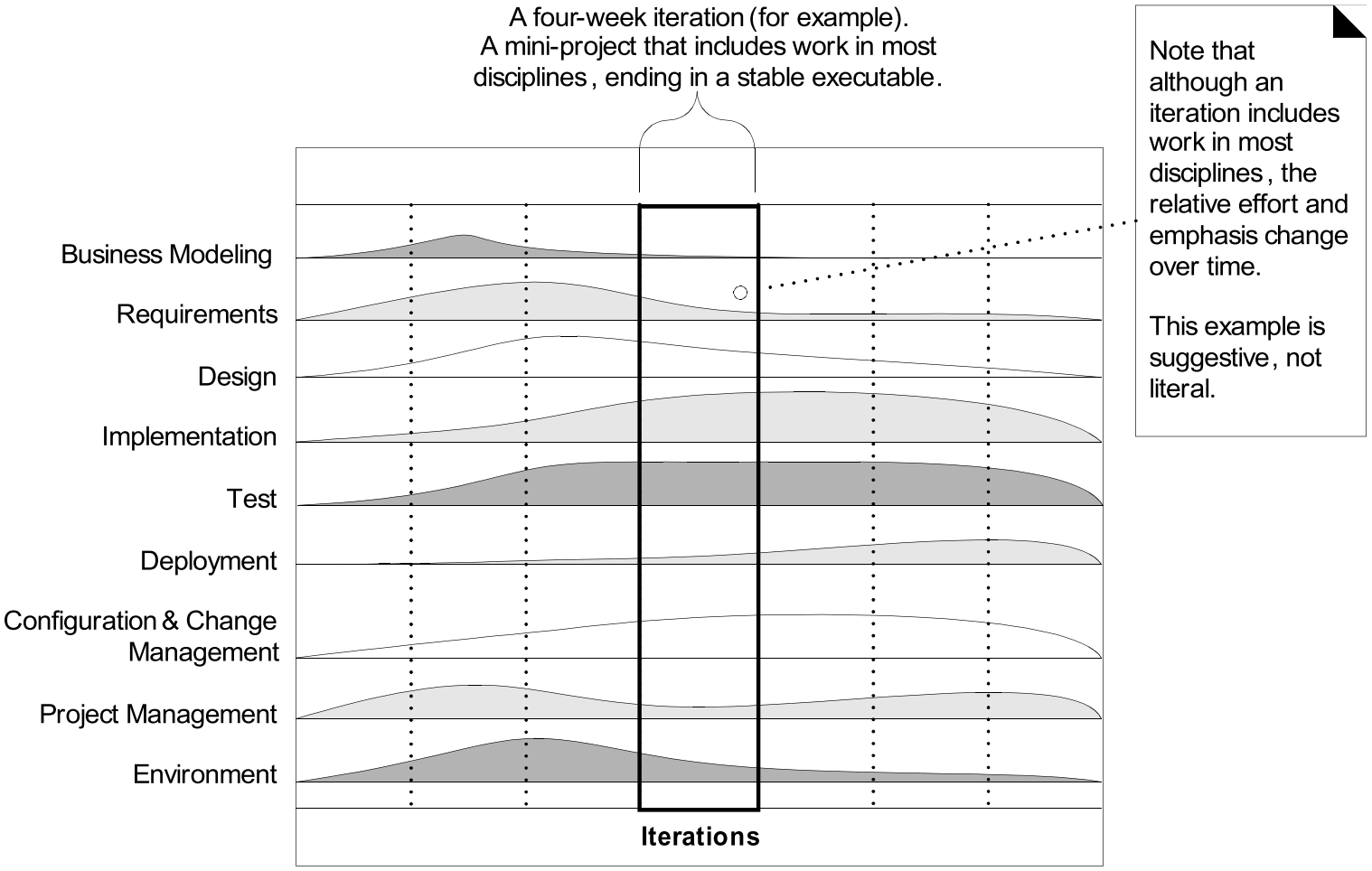
3.4 Welcher Prozess passt hier?

Die gesamte Bearbeitungszeit in unserem Projekt beträgt ca. 9 Wochen, wobei gleichzeitig projektfremde Aufgaben zu erledigen sind.

Es gibt 3 Meilensteine. Realistisch sind 3 Iterationen mit Schwerpunkten:

1. Iteration: Anforderungserfassung
2. Iteration: Architektur
3. Iteration: Implementierung und Testen.

In allen Iterationen werden – mit geringerer Priorität – auch andere Tätigkeiten notwendig sein, bspw. Dokumentation, Prototyping etc. (vgl. Illustration auf der nächsten Seite).



Larman C., 2004. *Applying UML and Patterns*. 3rd ed. Upper Saddle River, NJ: Pearson: Fig. 2.7 (slightly modified)

4. Anforderungen

Es gibt *funktionale* und *nicht-funktionale* Anforderungen.

Use-Case ist eine **textuelle** Ablaufbeschreibung, die

- **Funktionale** Anforderungen im Zusammenhang darstellt,
- die Interaktion zwischen Akteuren betont,
- nicht auf die objektorientierte Entwicklung beschränkt ist.

Ein ausführliches Beispiel für einen Use-Case finden Sie bei Larman, Kapitel 6, S. 61 ff. (s. Moodle).

4.1 Primäre Akteure und Stakeholder

Ein *primärer Akteur*

- nutzt Dienste des Systems um seine Ziele zu erreichen,
- löst in der Regel den Use-Case aus,
- ist immer ein *Stakeholder*.

Ein Stakeholder

- ist ein Mitglied einer Interessensgruppe, die aus allen am System interessierten Personen, Institutionen und anderen Systemen besteht,
- muss kein primärer Akteur sein.

4.1.1 Begriffsklärung: Stakeholder und System

- Der Begriff “Stakeholder” ist schwer zu übersetzen und hat sich (wie “Use-Case” statt “Anwendungsfall”) etabliert. Er wird daher in der Folge weiter verwendet.
- Wenn wir in von “System” sprechen, dann ist – falls nicht explizit anders erwähnt – das System gemeint, das wir erstellen möchten.

4.2 Andere Akteure

Ein *sekundärer Akteur*

- stellt Dienste bereit, die vom System genutzt werden,
- wird auch als *unterstützender Akteur (“supporting actor”)* bezeichnet.

Ein *tertiärer Akteur*

- hat Interesse am System, aber interagiert nicht mit ihm,
- wird auch als *stiller Akteur (“silent actor”)* bezeichnet.

Das System selbst ist ein Akteur, allerdings keiner der obigen, sondern ein eigenständiger Typ.

4.2.1 Finden von Akteuren

Akteure:

- definieren den Nutzen und wichtige Eigenschaften des Systems.
- haben Ziele die sie mit Hilfe des Systems erreichen wollen,
- bestimmen externe Schnittstellen und Protokolle,
- haben Ziele, die sonst in der Analyse leicht übersehen werden können.

Zur Identifikation von Akteuren kann man mit den primären Akteuren beginnen und wechselweise Akteure und Ziele identifizieren und sie in einer Liste notieren (“Actor-Goal-List”).

Vergleiche die Fallstudie für das Point-of-Sale System (POS) bei Larman (Actor-Goal-List auf S. 84).

4.3 Finden von Use-Cases

Die Actor-Goal-List liefert die primären Akteure und deren Ziele.
Ein Use-Case wird von einem primären Akteur ausgelöst.

Definiere Use-Cases, die die Ziele der Akteure erfüllen.

- Benenne die Use-Cases nach den Zielen.
- Definiere einen Use-Case pro Ziel.

Bestimme die *Systemgrenze*:

- Welche Akteure gehören zum System?
- Welche Use-Cases gehören zum System?

4.3.1 Prüfung: Handelt es sich um einen Use-Case?

- “Boss Test”
  - Die Ausführung des Use-Case macht den Boss glücklich?
  - Erzeugt der Use-Case einen messbaren Mehrwert?
- “Size Test”: Use-Cases sollen nicht zu kurz sein. Gegenbeispiel: “Eingabe einer Artikelnummer”.
- “EBP Test” (*Elementary Business Process*): Bevorzuge Use-Cases, die grundlegende Geschäftsprozesse abbilden (vgl. Larman S. 88). Beispiel: ‘"Retoure abwickeln"’.
- Ein Use-Case je Ziel eines Akteurs
  - Ausnahme: CRUD-Operationen (Create, Read, Update, Delete), bspw. Benutzerdaten anlegen, auslesen, ändern und löschen im Use-Case “Benutzer verwalten”.

4.3.2 Systemgrenze

Die Systemgrenze grenzt das zu implementierende System vom Rest der Welt ab und hat daher hat fundamentalen Einfluss auf die funktionalen und nicht-funktionalen Anforderungen.

Systemgrenze für POS-Fallstudie:

- POS-System ist das zu implementierende System
- Kassierer, Kunde, elektronisches Bezahlsystem, System zur Steuerberechnung, System zur Verkaufsanalyse, etc. liegen außerhalb.

4.3.3 Schreiben von Use-Cases: Kurzformat (“brief format”)

Der eigentliche Use-Case ist eine textuelle Ablaufbeschreibung.

Verkauf abwickeln: Der Kunde kommt mit Artikeln in den Kassenbereich. Der Kassierer erfasst jeden Artikel mit dem POS-System. Das System zeigt den aktuellen Gesamtbetrag und Detailinformation zum jeweiligen Artikel an. Der Kunde bezahlt den Gesamtbetrag. Das System erfasst und validiert die Zahlung. Das System aktualisiert den Warenbestand. Der Kunde erhält eine Quittung vom System und verlässt den Kassenbereich mit den Artikeln.

4.3.4 Stilfragen

Schreiben Sie immer in *essenziellen Stil*:

- Formuliere die Absicht und die Verantwortlichkeiten des jeweiligen Akteurs.
- Jeder Schritt beginnt mit einem Akteur und einem Verb und ist aktiv formuliert.
- Formuliere das Was und nicht das Wie (“black-box style”).
- Fasse Dich kurz!

4.3.5 Ausführliches Format (“fully dressed”) – Hauptzweig

Use Case: **Verkauf abwickeln**

Hauptzweig:

1. Kunde kommt mit Artikeln in den Kassenbereich.
2. Kassierer beginnt neuen Verkauf.
3. Kassierer gibt Artikelnummer ein.
4. System zeichnet den Einzelposten auf und zeigt Beschreibung, Preis und aktuelle Gesamtrechnung an. Preis wird auf Basis eines Regelsatzes berechnet.

Kassierer und System wiederholen 3-4 für alle Artikel.

5. System zeigt die Gesamtrechnung.

…

Vgl. Larman, S. 67 ff. und die bereitgestellte Vorlage

4.3.6 Ausführliches Format – Erweiterungen

Use Case: **Verkauf Abwickeln**

Erweiterungen:

1a. Kunde oder Manager will ausgesetzten Verkauf fortsetzen.

1. Kassierer wählt “Fortsetzen” und gibt ID des Verkaufs ein.
2. System zeigt aktualisierte Gesamtrechnung.
   2a. Verkauf nicht gefunden.
      1. System signalisiert dem Kassierer Fehler.
      2. Kassierer gibt alle Artikel neu ein.
3. Kassierer setzt Verkauf fort.

…

4.3.7 Aufbau der Erweiterung

- Die Erweiterung beginnt mit der zweiteiligen Nummerierung: Die Nummer im Hauptzweig, auf die sich die Erweiterung bezieht, gefolgt von einem Buchstaben, der die Erweiterungen für diese Nummer im Hauptzweig durchzählt (a, b, c, ...).
- Direkt hinter der Nummerierung steht die Bedingung, durch die diese Erweiterung ausgelöst wird.
- Die Behandlung der Bedingung (d. h. die Erweiterung selbst) wird wieder schrittweise durchgezählt. Weitere Verzweigungen in der Erweiterung werden nach dem Schema <Nummer-in-der-Erweiterung.Buchstabe> innerhalb der Erweiterung behandelt.

→ Ausführlich in Larman, S. 69 ff.

4.3.8 Ausführliches Format – Allgemeine Erweiterungen

Use Case: **Verkauf Abwickeln**

Erweiterungen:

∗a. Manager verlangt administrative Aktion

1. System wechselt in administrativen Modus.
2. Manager oder Kassierer führt Aktion aus.
3. System kehrt in normalen Modus zurück.

…

Diese Erweiterung gilt für **alle** Nummern im Hauptzweig, daher steht hier statt der Nummer das Zeichen ∗, gefolgt vom Buchstaben, der die allgemeinen Erweiterungen durchzählt.

4.4 Andere Anforderungen

Im FURPS+ Modell steht F(unctional) für die *funktionalen Anforderungen*, die durch die Use-Cases beschrieben werden. Die *nicht funktionalen* Anforderungen werden auch als Qualitätsattribute bezeichnet:

- Usability
- Reliability
- Performance
- Supportability
- + Implementierung, Recht, etc.

werden in der Regel einfach in einer Liste aufgeführt. Sie müssen quantifiziert und damit überprüfbar sein. Beispiel: Reaktionszeit des Systems in Sekunden.