

Leseprobe

Chris Rupp, die SOPHISTen

Requirements-Engineering und -Management

Aus der Praxis von klassisch bis agil

ISBN (Buch): 978-3-446-43893-4

ISBN (E-Book): 978-3-446-44313-6

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-43893-4>

sowie im Buchhandel.

Inhaltsverzeichnis

Einleitung	1
Was Sie in diesem Buch erwartet	1
Die SOPHISTen: Alt und Neu.....	3
Neue Erkenntnisse und bewährtes Wissen.....	3
Das Team.....	4
Wissen erarbeiten: Selbsttest und Blended Learning mit ILIAS®	6
Wissen beschreiben: Kapitel für Kapitel zum RE-Leitfaden	6

Teil I – Requirements-Engineering zum Erfolg bringen 7

1 In medias RE	9
1.1 Motivation für eine erfolgreiche Systemanalyse	10
1.2 Der Requirements-Engineer – Mittler zwischen den Welten	11
1.3 Das Requirementsgehirn	12
1.4 Die Disziplin Requirements-Engineering	13
1.5 Die Einteilung von Anforderungen	17
1.5.1 Einteilung von Anforderungen nach ihrer Art.....	17
1.5.2 Einteilung von Anforderungen nach ihrer rechtlichen Verbindlichkeit	18
1.6 Gründe für Dokumentation.....	19
1.6.1 Wissen verfällt bzw. diffundiert.....	19
1.6.2 Detailtiefe und Verständnis fehlt.....	21
1.6.3 Verlust des Gesamtüberblicks	22
1.6.4 Missverständnisse entstehen und bleiben	23
1.6.5 Abweichende Informationen verteilen sich	23
1.7 Typische Probleme in der Anforderungsanalyse.....	24
1.8 Qualitätskriterien im Requirements-Engineering	26
1.8.1 Qualitätskriterien für jede einzelne Anforderung	26
1.8.2 Qualitätskriterien für die Anforderungsspezifikation.....	28
1.8.3 Pragmatische Aspekte von Anforderungen und Anforderungsspezifikationen.....	29
2 Das Bibliothekssystem – wie alles begann	31
3 Von der Idee zur Spezifikation	33
3.1 Von der richtigen Anforderungsmenge.....	34
3.2 Der Zusammenhang zwischen Anforderungen.....	36
3.2.1 Anforderungen und die Architektur.....	36
3.2.2 Anforderungen und deren Verfeinerungen.....	37
3.2.3 Detaillierungsebenen	38
3.3 Die Systemanalyse im Überblick	42

Inhaltsverzeichnis

3.3.1	Anforderungen herleiten.....	46
3.3.2	Anforderungen zum richtigen Zeitpunkt.....	47
3.4	Das Vorgehen in der Projektpraxis	49
4	Agile und andere Vorgehensweisen.....	51
4.1	Konventionelle Vorgehensmodelle und Qualitätsstandards.....	52
4.2	Agile Vorgehensweisen	56
4.2.1	Kanban.....	57
4.2.2	Scrum.....	57
4.3	RE und Scrum	61
4.3.1	Integrationsmöglichkeiten von RE in Scrum	62
4.3.2	Dokumentieren von Anforderungen in Scrum.....	64
4.3.3	RE als Scrum-Projekt.....	65
4.3.4	RE ist immer agil	68
	Teil II – Anforderungen ermitteln.....	71
5	Ziele, Informanten und Fesseln	73
5.1	Die wichtigsten Schritte vor dem Start in die Systemanalyse	75
5.1.1	Anforderungsquellen: Ausgangspunkt und Mittelpunkt	77
5.1.2	Die derzeitige Realität unter die Lupe nehmen	78
5.1.3	Probleme erkunden und Optimierungspotenziale beschreiben.....	78
5.1.4	Ziele definieren und bewerten	78
5.2	Der Stakeholder – das unbekannte Wesen.....	79
5.2.1	Die Notation von Stakeholdern	81
5.2.2	Stakeholder-Relationship-Management – die Pflege von Stakeholdern	83
5.3	Ziele beschreiben	83
5.4	Umfang, Kontext und Grenzen des Systems festlegen	85
5.4.1	Die Kontextabgrenzung.....	85
5.4.2	System- und Kontextgrenzen bestimmen	86
6	Anforderungsermittlung – Hellssehen für Fortgeschrittene.....	90
6.1	Ran an die Kundenwünsche.....	90
6.1.1	Aller Anfang ist schwer	90
6.1.2	Kommunikationsmodelle	91
6.1.3	Repräsentationssysteme der Sprache	93
6.1.4	Die Qual der Wahl	94
6.2	Die entscheidenden Produktfaktoren	94
6.2.1	Basisfaktoren ausgraben.....	95
6.2.2	Leistungsfaktoren abholen.....	96
6.2.3	Begeisterungsfaktoren erarbeiten.....	97
6.3	Ermittlungstechniken.....	98
6.3.1	Kreativitätstechniken	99

6.3.2	Beobachtungstechniken	103
6.3.3	Befragungstechniken	105
6.3.4	Artefaktbasierte Techniken.....	110
6.3.5	Unterstützende Techniken	112
6.4	Anwendung in der Praxis	119
7	Das SOPHIST-REgelwerk – Psychotherapie für Anforderungen.....	123
7.1	Vom Phänomen der Transformation – sprachliche Effekte	124
7.2	Die Wurzeln – das Neurolinguistische Programmieren.....	125
7.2.1	Transformationsprozesse	125
7.2.2	Kategorien der Darstellungstransformation.....	128
7.3	Vom Umgang mit sprachlichen Effekten	131
7.4	Das Vorgehen beim SOPHIST-REgelwerk –Anforderungen auf die Couch gelegt.....	133
7.5	Prüfen der Satzbestandteile	135
7.5.1	Prüfen der Prozesse.....	136
7.5.2	Prüfen von Eigenschaften	144
7.5.3	Prüfen von Mengen und Häufigkeiten.....	148
7.5.4	Prüfen von Begriffen, die Möglichkeiten beschreiben.....	152
7.6	Prüfen des Satzes	154
7.7	Prüfen des Gesamtbilds.....	156
7.8	Anwendung des SOPHIST-REgelwerks.....	161
	Teil III – Anforderungen formulieren	165
8	Grundlagen für die Systemanalyse dokumentieren	167
8.1	Ausgangssituation beschreiben? Ja bitte!	168
8.2	Geschäftsprozessbeschreibung	169
8.2.1	Business-Use-Cases.....	170
8.2.2	Ablaufdiagramme	172
8.2.3	Geschäftsregeln.....	177
8.3	Ziele dokumentieren	180
8.4	Kontextvisualisierung	181
8.4.1	Use-Case-Diagramm zur Kontextvisualisierung	182
8.4.2	Kontextdiagramm der Strukturierten Analyse	182
8.5	Begriffe und Definitionen	183
9	Systemanforderungen dokumentieren – malen oder schreiben?	185
9.1	Dokumentation? Ja bitte!	186
9.2	Anforderungen in Prosa beschreiben	187
9.3	Szenarien	187
9.4	Das System-Use-Case-Diagramm.....	189
9.5	Die Use-Case-Beschreibung	192
9.6	Das Aktivitätsdiagramm	195

Inhaltsverzeichnis

9.7	Das Sequenzdiagramm	197
9.8	Zustandsdiagramm	199
9.9	Das Klassendiagramm als Begriffsmodell.....	201
9.10	Beschreibung von Systemregeln.....	203
9.11	Anforderungen verfeinern	206
9.11.1	Diagramme verfeinern/konkretisieren/detaillieren	206
9.11.2	Tipps zum Thema Detaillierung	208
9.12	Die Wahl der richtigen Dokumentationstechniken	208
9.12.1	Einflussfaktoren auf die Wahl der Dokumentationstechniken.....	210
9.12.2	Auswahlempfehlungen.....	211
9.12.3	Diagramm oder doch lieber natürliche Sprache?	212
10	Anforderungsschablonen – der MASTER-Plan für gute Anforderungen	215
10.1	Linguistische und philosophische Grundlagen	216
10.2	Der schablonenbasierte Ansatz	217
10.3	Schritt für Schritt zur Anforderung.....	219
10.4	Semantische Präzisierung der Anforderungsschablone	225
10.4.1	Rechtliche Verbindlichkeiten	226
10.4.2	Verben – Prozesswörter.....	227
10.4.3	Substantive – Akteure, Rollen, Objekte, Eigenschaften und Abkürzungen.....	228
10.4.4	Bedingungen	229
10.5	Konstruieren in englischer Sprache.....	230
10.5.1	Der Syntaxbauplan im Englischen	230
10.5.2	Semantische Normierung im Englischen	231
10.6	Details für die Konstruktion	232
10.6.1	Präzisierung des Objekts.....	232
10.6.2	Konkretisierung des Prozessworts.....	233
10.6.3	Die Details in englischer Sprache.....	234
10.7	Nicht-funktionale Anforderungen.....	234
10.7.1	Eigenschaften	235
10.7.2	Umgebungen und Kontext	237
10.7.3	Prozesse	238
10.7.4	Konstruieren in englischer Sprache	239
10.8	Bedingungen in Anforderungen	240
10.8.1	Syntax für und Semantik in Bedingungen.....	240
10.8.2	Konstruieren in englischer Sprache	242
10.9	Auf die Sätze, fertig, los!.....	243
11	Dokumentation im agilen Umfeld	247
11.1	Artefakte – eine Übersicht.....	248
11.2	User-Storys	249
11.2.1	Aufbau einer User-Story	249
11.2.2	Das nehm‘ ich dir nicht ab! – Akzeptanzkriterien für User-Storys...	250
11.2.3	Von Use-Cases, User-Storys und Story-Maps.....	252

11.3	Technical Storys	254
11.3.1	Aufbau von Technical Storys	255
11.3.2	Die Priorisierungsproblematik	255
11.4	User-Storys schneiden und verfeinern	256
11.4.1	Das Meta-Pattern	256
11.4.2	Der Minimal-Ansatz und der Reduktions-Ansatz	258
11.5	Wann ist fertig wirklich „fertig“? – Die Definition of Done (DoD) und die Definition of Ready (DoR)	260
11.5.1	Die Definition of Done – weil's gut werden muss	260
11.5.2	Die Definition of Ready – das Quality-Gate für User-Storys	261
11.6	And all together now! – Wann setze ich welche Technik ein?	262
12	Nicht-funktionale Anforderungen – die heimlichen Stars	267
12.1	Definition, Bedeutung und Chancen	268
12.2	Ermitteln und Dokumentieren von NFAs	270
12.2.1	Vorbereitende Tätigkeiten	271
12.2.2	Durchzuführende Tätigkeiten	272
12.2.3	Best Practices	275
12.3	Technologische Anforderungen	277
12.3.1	Inhalte	277
12.3.2	Erfahrungen aus dem Projektalltag	278
12.4	Qualitätsanforderungen	280
12.4.1	Inhalte	281
12.4.2	Erfahrungen aus dem Projektalltag	283
12.5	Anforderungen an die Benutzungsoberfläche	286
12.5.1	Inhalte	287
12.5.2	Dokumentieren von Benutzungsoberflächen	289
12.5.3	Erfahrungen aus dem Projektalltag	292
12.6	Anforderungen an sonstige Lieferbestandteile	292
12.6.1	Inhalte	293
12.6.2	Erfahrungen aus dem Projektalltag	293
12.7	Anforderungen an durchzuführende Tätigkeiten	294
12.7.1	Inhalte	295
12.7.2	Erfahrungen aus dem Projektalltag	295
12.8	Rechtlich-vertragliche Anforderungen	295
12.8.1	Inhalte	296
12.8.2	Erfahrungen aus dem Projektalltag	298
Teil IV – Anforderungen prüfen und bewerten		299
13	Der Qualitätssicherungsprozess – Menetekel oder Wunderheilung?	301
13.1	Qualität ist das, was der Kunde braucht	302
13.1.1	Ziele in der Qualitätssicherung von Anforderungen	303
13.1.2	Konstruktive und analytische Qualitätssicherung von Anforderungen	303

Inhaltsverzeichnis

13.1.3	Vorgehen beim Prüfen von Anforderungen.....	305
13.2	Der Qualitätssicherungsleitfaden – damit Sie loslegen können	306
13.2.1	Qualitätsziele festlegen.....	307
13.2.2	Qualitätssicherungsmethoden auswählen.....	308
13.2.3	Prüfzeitpunkte definieren.....	308
13.2.4	Über die Auswahl geeigneter Prüfer	310
13.3	Plan - Qualitätsprüfung vorbereiten	312
13.3.1	Prüfbarkeit feststellen	312
13.3.2	Prüfgegenstand definieren.....	313
13.3.3	Prüfgegenstand extrahieren und dokumentieren	313
13.4	Do - Qualitätsprüfung durchführen	313
13.4.1	Spezifikationselement bewerten	314
13.4.2	Prüfbericht verfassen.....	314
13.5	Check – Ergebnisse beurteilen.....	314
13.6	Act – Maßnahmen initiieren	315
14	Prüftechniken für Anforderungen – ungeahntes Verbesserungspotenzial	317
14.1	Die Prüftechniken im Detail	318
14.1.1	Reviews	318
14.1.2	Prototyp/Simulationsmodell	322
14.1.3	Testfälle	323
14.1.4	Analysemodell	326
14.1.5	Hilfsmittel bei der Prüfung.....	328
14.2	Vom Durchblick im Dschungel der Prüftechniken.....	330
14.2.1	Einschätzung der Prüftechniken	331
14.2.2	Über die Auswahl geeigneter Prüftechniken.....	331
15	Qualitätsmetriken – drum messe, wer sich ewig bindet.....	333
15.1	Qualitätsmetriken – die Hüter der Anforderungsqualität	334
15.1.1	Qualitätsmetriken für Anforderungen.....	335
15.1.2	Ziele von Qualitätsmetriken – der Blick ins Unbekannte.....	336
15.1.3	Verwendung von Metriken – die erste Herausforderung	337
15.2	Vorbereitung der Messung mit Qualitätsmetriken	337
15.2.1	Qualitätsziele festlegen.....	338
15.2.2	Messleitfaden erweitern	338
15.2.3	Stichprobenumfang definieren.....	338
15.2.4	Stichproben festlegen und dokumentieren	340
15.3	Durchführung.....	342
15.3.1	Qualitätskennzahlen berechnen	343
15.3.2	Messergebnis dokumentieren.....	344
15.3.3	Qualitätskennzahlen beurteilen.....	345
16	Anforderungskonsolidierung – wider den Widerspruch.....	347
16.1	Was ist ein Konflikt?	348
16.2	Konfliktidentifikation	349

Inhaltsverzeichnis

19.1.2	Die Übersicht behalten – Filtern und Sichten bilden	418
19.2	Auswertungen	419
19.3	Traceability	421
19.3.1	Eltern-Kind-Verbindung.....	422
19.3.2	Verbindung von Anforderungen auf gleicher Ebene.....	425
19.3.3	Verbindung zwischen verschiedenen Informationsarten	425
19.3.4	Traces technisch realisieren	426
19.3.5	Definition eines Verfolgbarkeitsmodells.....	429
19.4	Anforderungen strukturieren.....	432
19.4.1	Strukturierung nicht-funktionaler Anforderungen	432
19.4.2	Strukturierung funktionaler Anforderungen	433
19.5	Anforderungen importieren und exportieren.....	442
20	Change- & Release-Management – die stabile Instabilität	445
20.1	Quellen und Typen von Änderungen – es kommt was auf Sie zu	447
20.1.1	Incident-Management – einer für alle und alles auf einmal.....	448
20.1.2	Fachbereich und Produkt-Management	448
20.1.3	Tester	448
20.1.4	Entwickler.....	449
20.1.5	Definitionen der Tickettypen.....	449
20.1.6	Sammeltopf für die Tickets.....	451
20.2	Change-Management.....	451
20.2.1	Priorisierung der Tickets.....	451
20.2.2	Änderung grob beschreiben und entscheiden.....	452
20.3	Tickets einplanen	452
20.4	Release-Management	453
20.4.1	Änderungen durchführen – die Stunde der Traceability	453
20.4.2	Konfigurationen und Basislinien.....	455
20.5	Der Zielspurt – Release ausrollen	456
20.6	Ausnahmesituation – das Emergency Release	458
21	Wiederverwendung – aus alt mach neu	459
21.1	Das Rad nicht immer neu erfinden	460
21.2	Die potenziellen Kandidaten.....	461
21.3	Regelgeleitete Wiederverwendung.....	462
21.3.1	Spezifikationslevel.....	462
21.3.2	Eingeschränkte Produktpalette	463
21.3.3	Einbindung in den Ablauf.....	463
21.3.4	Technologie.....	464
21.3.5	Zwischenfazit.....	464
21.4	Wiederverwendung in der Praxis.....	464
21.5	Auswahl der Vorgehensarten	465
21.5.1	Der Ansatz nach IVENA XT	465
21.5.2	Produktlinien	469

**Teil VI – Spezialfälle meistern: Einführungsprojekte,
Delta Anforderungen, und Usability Engineering ..475**

22	Einführungsstrategien – ein Ratgeber für die organisierte REorganisation	477
22.1	Gründe für eine gute Strategie.....	478
22.1.1	Einführung bedeutet Veränderung.....	478
22.1.2	Nichts ist beständiger als der Wandel.....	479
22.1.3	Veränderung bedeutet Lernen.....	480
22.2	Eine Einführung ist ein Projekt!.....	483
22.2.1	Den Grundstein legen – Erstellung des fachlichen Konzepts	483
22.2.2	Die Umsetzung vorbereiten	485
22.2.3	Umsetzen und anpassen.....	490
22.3	Arbeitspakete einer Einführung.....	491
22.3.1	Marketingkonzept	491
22.3.2	Konzept zur Wissensvermittlung	492
22.3.3	Pilotierungskonzept	498
22.3.4	Migrationskonzept.....	501
22.4	Aufbruch in ein agile(RE)s Leben	504
22.4.1	Vom Wasserfall zur Agilität.....	505
22.4.2	Der hybride Ansatz: klassisches RE & agile Entwicklung	506
22.4.3	Flexibel und adaptiv: agiles RE & agile Entwicklung	508
23	Der Delta-Ansatz – jenseits der grünen Wiese	513
23.1	Delta-Anforderungen – die machen den Unterschied!.....	514
23.2	Das Vorgehen beim Delta-Ansatz.....	515
23.3	Delta-Ansatz oder neue Spezifikation?.....	521
23.4	Delta-Spezifikation im agilen Kontext.....	521
24	Requirements und Usability – wie sich Anforderungen und Benutzerfreundlichkeit ergänzen	534
24.1	Requirements und Usability	524
24.2	Das Persona-Konzept im Requirements-Engineering.....	526
24.2.1	Der Persona-Steckbrief	526
24.2.2	Das Wichtigste zuerst: die Identität	526
24.2.3	Demografische Variablen	527
24.2.4	Verhaltensvariablen.....	527
24.3	Der Persona-Steckbrief als Anforderungsquelle.....	529
24.4	Verifizieren von RE-Artefakten.....	529
24.5	Modellieren aus Benutzersicht.....	532
24.5.1	Der Nutzungsablauf als Modell	532
24.5.2	Begriffe und Zustände	533
24.6	Qualitätssicherung und Übergabe an das Design	533

Inhaltsverzeichnis

Anhang

A – ILIAS® – die neue Art des Lernens	535
B – Literaturverzeichnis.....	537
C – Fotoverzeichnis.....	547
Index.....	549

Einleitung



Liebe Leserin, lieber Leser

Mit dem Ziel, Ihnen ein alltags- und praxistaugliches Kompendium für die Arbeit mit Anforderungen an die Hand zu geben, haben wir unsere Erfahrungen diskutiert, kritisiert, revidiert, und weiter expliziert, und können nun voller Stolz ein „Heureka!“ verkünden:

Vor Ihnen liegt die neue Auflage unseres Standardwerks zum Thema Requirements-Engineering und Requirements-Management.

Was Sie in diesem Buch erwartet

Auch in dieser Auflage haben wir die Buchinhalte in logische Abschnitte eingeteilt, um die verschiedenen Aspekte des Requirements-Engineering zu beschreiben.

Teil I: Requirements-Engineering zum Erfolg bringen

Wieso wird Requirements-Engineering überhaupt betrieben? Ist das sinnvoll? Lohnt sich das? Die Antwort auf diese und ähnliche Fragen finden Sie im ersten Abschnitt. Um die recht theoretischen Ausführungen anschaulicher zu machen, haben wir ein Beispiel mit einer Rahmenhandlung eingebaut, das Sie durch alle Buchkapitel begleiten wird und hier beginnt. Zusätzlich finden Sie in diesem Abschnitt einen Überblick über das Vorgehen bei der Analyse sowie agile und andere Vorgehensweisen in Projekten.

Teil II: Anforderungen ermitteln

Anforderungen fallen (leider) nicht vom Himmel. Sie müssen in harter Arbeit aus einer Vielzahl von Quellen zusammengetragen werden. In diesem Abschnitt stellen wir Ihnen diese Quellen vor, zeigen Probleme auf, die bei der Ermittlung auftreten, und legen dar, wie Anforderungen auf Herz und Nieren geprüft werden können.

Teil III: Anforderungen formulieren

Nach der Ermittlung müssen die Anforderungen auch schriftlich dokumentiert werden. Wir stellen Ihnen Spezifikationsebenen und passende Notationen vor und zeigen Ihnen alte und neue Anwendungsmöglichkeiten für unser bewährtes Template für natürlichsprachliche Anforderungen. Mit Informationen zu agilen Dokumentationsformen wie User-Stories und ihrer Schneidung sind Sie auch für Requirements Engineering im agilen Umfeld gerüstet. Last but not least gehen wir auf die Stiefkinder im Requirements Engineering ein: nicht-funktionale Anforderungen.

Teil IV: Anforderungen prüfen und bewerten

Qualitätssicherung – für manche Leid, für manche Freud. Eine klare Definition von „Qualität“ und ein konsequenter, verständlicher Prozess machen das Leben ein bisschen leichter. Wir zeigen Ihnen die wichtigsten Prüfetechniken und die Anwendung von Qualitätsmetriken für Anforderungen, so dass Sie auch Ihr Management von der Qualität Ihrer Anforderungen überzeugen können. Natürlich lassen wir Sie auch bei der Konsolidierung von Anforderungen nicht alleine und machen Sie mit RE-Konflikten und ihrer Auflösung vertraut.

Teil V: Anforderungen verwalten

Anforderungen sind nicht in Stein gehauen, sondern ändern sich mit der Zeit ganz erheblich und müssen verwaltet werden – eine ganz und gar nicht triviale Aufgabe. Hier finden Sie eine Anleitung, wie Anforderungsmanagement sinnvoll gestaltet wird, welche Zustände Anforderungen während ihrer Lebensdauer durchlaufen, wie Anforderungen freigegeben werden und wie sie, nach Gebrauch, am besten wiederverwendet werden.

Teil VI: Spezialfälle meistern: Einführungsprojekte, Delta-Anforderungen, und Usability Engineering

Um Sie für alle Eventualitäten zu wappnen, ist der abschließende sechste Abschnitt des Buches speziellen, aber essentiellen Fragestellungen gewidmet. Fehlt es an einem Prozess für das

Requirements-Engineering? Schaffen Sie Abhilfe mit einer gelungenen Einführungsstrategie. Ist Ihre Dokumentation eher Flickwerk als Arbeitsgrundlage? Nutzen Sie den Delta-Ansatz, um von einer lückenhaften Dokumentation zu einem soliden Arbeitsstand zu gelangen. Haben Sie Daten aus dem Usability-Engineering? Verwenden Sie Personas zur Integration von Usability-Engineering und Requirements-Engineering.

Die SOPHISTen: Alt und Neu

Die Sophisten, eine Gruppe von Philosophen, lebten in der Zeit um 450 vor Christus in Athen. Sie galten als die Ersten, die auf die von den Vorsokratikern propagierte Naturphilosophie eine *menschenbezogene* Antwort gaben. Protagoras (481– 411) postulierte: „Der Mensch ist das Maß aller Dinge“. Sie gaben auch die entscheidenden Impulse für die Entwicklung vom Mythos zum Logos, das heißt zur Idee eines durch theoretische Vernunft begründeten Weltverständnisses.

Als SOPHISTen der Neuzeit bezeichnen sich die Mitarbeiter der SOPHIST GmbH, der Gesellschaft für innovatives Software-Engineering. Die Ideen und die Werte der alten Sophisten haben wir aufgegriffen und sehen es als Teil unserer Mission, unsere Kunden dazu zu bringen, das Althergebrachte in Frage zu stellen. Seit Jahren begleiten die SOPHISTen namhafte Kunden in unterschiedlichsten Projekten mit Coaching, Training und Auditierung. Dadurch entstand ein umfassender Wissenspool in den Bereichen Requirements-Engineering und -Management und Architektur.



Neue Erkenntnisse und bewährtes Wissen

Seit der letzten Auflage sind fünf Jahre ins Land gegangen - fünf Jahre, die wir genutzt haben, um bewährte Herangehensweisen zu erweitern und zu verbessern und um neues Wissen zu erarbeiten und für unsere Kunden und Freunde nutzbar zu machen. Auch in dieser Auflage verfolgen wir weiter das Ziel, Ihnen die Vorteile und Inhalte von gutem Requirements-Engineering und -Management zu vermitteln, ganz unabhängig davon, ob Sie im agilen oder im traditionellen Projektumfeld tätig sind.

Die grafische Gestaltung hat sich seit der letzten Auflage nicht grundlegend geändert - Warum etwas reparieren, das ganz offensichtlich nicht kaputt ist? Sie werden zu dieser Auflage jedoch mehr Inhalte online finden.

Das Team

Auf den nächsten beiden Seiten werden Sie die Personen kennenlernen, die an der Erstellung dieses Buches beteiligt waren. Eine detaillierte Vorstellung der Autoren finden Sie auf unserer Homepage. www.sophist.de

Was halten Sie von dem, was wir hier geschrieben haben? Das gesamte Team freut sich auf Ihre Eindrücke und Verbesserungsvorschläge, Ihre Kritik, aber auch Ihr Lob. Treten Sie mit uns in Kontakt. buch@sophist.de

„Die Grenzen meiner Sprache sind die Grenzen meiner Welt.“

L. Wittgenstein (1889 -1951)

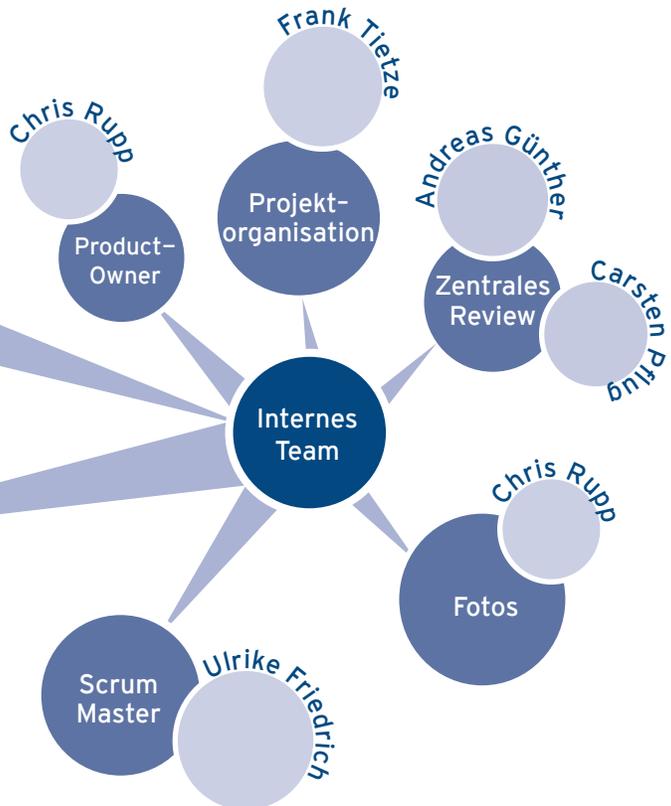
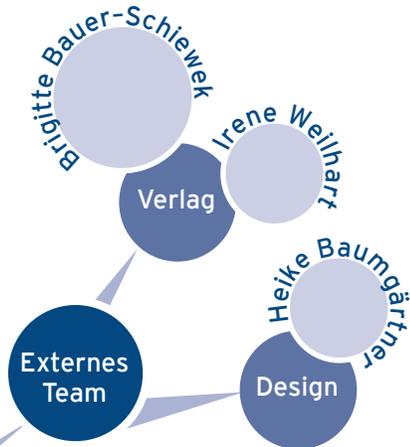
Das Team



Dr. Walter Wintersteiger
Jurgen Appelo
Suzanne Robertson
James Robertson
Karl Wieggers
Alistair Mavin
Ellen Gottesdiener
Mary Gorman
Karol Frühauf
Dr. Helmut Sandmayr
Dr. Matthias Recknagel
Volker Schmidt
Daniel Hopp
Dr. Stefan Walburg
Erik Simmons



Chris Rupp



Wissen erarbeiten: Selbsttest und Blended Learning mit ILIAS®

„ILIAS“:
Integriertes Lern-,
Informations-, und
Arbeitskooperations-
System

Im SOPHIST Learn Management System, basierend auf der Lernplattform **ILIAS®**, finden Sie zu jedem Kapitel des Buches eine Checkliste, mit der Sie interaktiv Ihr Verständnis prüfen können. Verwenden Sie diesen Link, um zur Übersicht der Kapitel-Checklisten zu gelangen:

www.sophist.de/re6/checkliste

Einen tieferen Einstieg in die Materie bieten Blended Learning-Trainings. Als Teilnehmer machen Sie sich in der Onlinephase eigenständig und Ihrem Lernrhythmus entsprechend mit der Theorie vertraut. In der zugehörigen Präsenzveranstaltung werden offene Fragen beantwortet und das neue Wissen vertieft. Zudem haben Sie die Möglichkeit, sich in den Foren der Trainings mit den anderen Teilnehmern auszutauschen. Aktuelle Informationen zu den verfügbaren Blended Learning-Trainings finden Sie auf unserer Homepage unter www.sophist.de/blended-learning/.

Haben wir Ihr Interesse geweckt?

Dann werfen Sie doch einfach mal einen Blick auf unseren Blended Learning-Schnupperkurs unter:

lms.sophist.de

Wissen beschreiben: Kapitel für Kapitel zum RE-Leitfaden

Neudeutsch:
zu pushen

Um die Praxistauglichkeit unseres Kompendiums noch weiter zu **erhöhen**, liefern wir Ihnen auch eine Bauanleitung für Ihren Requirements-Engineering-Leitfaden mit.

Ein Requirements-Engineering-Leitfaden dient nicht nur dazu, Wissen zu festigen, das z. B. in Schulungen und Workshops vermittelt wurde, er kann auch als Nachschlagewerk und als erster Einstieg für neue Mitarbeiter dienen. In den einzelnen Kapiteln unseres Buchs werden wir Sie sowohl im Text als auch in Kommentaren an wichtigen Stellen darauf hinweisen, welche Entscheidungen Sie im Leitfaden dokumentieren müssen. Weitere Informationen zu Leitfäden finden Sie auch in Kapitel 22 „Einführungsstrategien“.

Dokumentation im agilen Umfeld



- Nutzen Sie User-Stories, um die Funktionalitäten Ihres Systems in agil durchgeführten Projekten zu dokumentieren.
- Entdecken Sie das Potenzial von Akzeptanzkriterien und Technical Storys, um nicht-funktionale Aspekte zu erfassen.
- Opfern Sie nicht Ihren Return on Investment bei der User-Story-Zerlegung, sondern schneiden Sie mit Methode!

11 Dokumentation im agilen Umfeld

So viele Dinge hat Herr Büchle inzwischen kennengelernt und hin und wieder schwirrt ihm der Kopf von all den verschiedenen Prozessen, Methoden und Ansätzen, die Ramona ihm vorstellt. Vor allem das, was sie über agile Vorgehensweisen erzählt hat, scheint ihm noch etwas magisch - was genau wird da dokumentiert und wie sieht denn so eine User-Story überhaupt aus? Er hat etwas von Karteikarten gehört, aber mehr als ein paar grobe Stichpunkte passen darauf ja wohl nicht. Wenn es in einem agilen Umfeld keine fein detaillierten Anforderungen gibt, wie weiß man dann überhaupt, wann die Anforderung umgesetzt ist? Herr Büchle sucht Ramonas Rat, um Licht in sein Dunkel der Dokumentation in einem agilen Umfeld zu bringen ...

11.1 Artefakte – eine Übersicht

Die Artefakte, die zur Dokumentation von Anforderungen in einem Projekt verwendet werden, können je nach der eingesetzten Methodik variieren. In diesem Kapitel stellen wir Ihnen User-Stories vor, eine Dokumentationstechnik, die ursprünglich aus dem Extreme Programming [Beck00] hervorgegangen ist, inzwischen aber in allen gängigen agilen Ansätzen sehr populär ist. Zusätzlich lernen Sie Technical Stories, die Definition of Ready und die Definition of Done kennen, die gemeinsam mit den Akzeptanzkriterien der User-Stories maßgeblich dazu beitragen, eine hohe Qualität des zu entwickelnden Systems sicherzustellen.

Eine andere Art der Backlog-Items sind Technical Storys aus Abschnitt 11.3.

User-Stories sind eine Art von Backlog-Items (Einträgen im Backlog). Die Gesamtheit der Backlog-Items für ein System bildet das Product-Backlog (siehe Kapitel 4 „Agile und andere Vorgehensweisen“), das aber keine Anforderungsspezifikation im klassischen Sinne darstellt. Der Product-Owner ist als Hüter des Product-Backlogs dafür verantwortlich, dass die User-Stories für die Sprintplanung priorisiert, mit Akzeptanzkriterien versehen und für die Umsetzung im Sprint ausreichend verfeinert sind (siehe Abschnitt 11.4 User-Stories schneiden und verfeinern).

Beständiger Wandel: das Product-Backlog

Ein Product-Backlog mag auf den ersten Blick einer traditionellen Anforderungsspezifikation ähneln, unterscheidet sich bei näherem Hinsehen aber gravierend davon. Während eine Anforderungsspezifikation basierend auf Analyseergebnissen eine Übersicht über das zu entwickelnde System gibt, sind die Items im Product-Backlog nach Priorität geordnete Arbeitspakete. Somit ist das Backlog eher eine ständig im Wandel befindliche Liste, die als Instrument zur Projektplanung dient, und wird über die gesamte Laufzeit des Projekts weiterentwickelt, verändert und umpriorisiert.

Zu Beginn eines agil durchgeführten Projekts wird meist eine Produktvision entwickelt, aus der im Laufe des Projekts User-Stories abgeleitet werden. Diese User-Stories können anfangs noch sehr grobgranular sein, man spricht hier auch von sogenannten „Epics“. Ein „Epic“ (Deutsch: Epos) ist so vage formuliert, dass es eine Vielzahl an Funktionalitäten umfassen und daher nicht abgeschätzt werden kann. Ein Beispiel-Epic aus dem Bibliothekssystem könnte so lauten:

Als Bibliothekar möchte ich den Bestand verwalten können.

Um in der agilen Systementwicklung Aufwände schätzen und Sprints planen zu können, besteht die Notwendigkeit, solche grobgranularen User-Stories zu verfeinern oder zu

„schneiden“. In Abschnitt 11.4 stellen wir Ihnen zwei Ansätze zur Schneidung von User-Stories vor, die den Return on Investment (RoI) berücksichtigen.

Manchmal hängen mehrere User-Stories inhaltlich zusammen. Diese User-Stories bezeichnet man auch als „Theme“ (Deutsch: Thema), um auszudrücken, dass sie nicht vollkommen unabhängig voneinander umgesetzt werden können. Das Theme bildet eine gedankliche Klammer (fachlich oder technisch) um mehrere User-Stories und kann z. B. über textuelle Referenzen an den einzelnen Stories des Themes abgebildet werden. Häufig bilden beispielsweise die User-Stories, die die CRUD-Operationen abdecken, ein Theme.

Durch die starke Betonung der gewünschten Funktionalität werden nicht-funktionale Aspekte bei der Erstellung von User-Stories oft vernachlässigt. Akzeptanzkriterien (siehe Abschnitt 11.2.2) und Technical Stories (siehe Abschnitt 11.3) stellen zwei Möglichkeiten dar, auch nicht-funktionale Aspekte eines Systems zu erfassen. Systemübergreifende und prozessbezogene Qualitätsanforderungen und Randbedingungen können außerdem einen Teil der Definition of Ready bzw. der Definition of Done (siehe Abschnitt 11.5) bilden.

Akronym für
Create (Erstellen),
Read (Lesen),
Update (Ändern),
Delete (Löschen)

11.2 User-Stories

User-Stories sind Kommunikationsversprechen.

User-Stories dienen als mentale Anker dafür, dass der Kunde eine bestimmte Systemfunktionalität möchte und dass über die Details der Umsetzung dieser Story noch diskutiert werden muss. Sie dienen als Grundlage für die Kommunikation zwischen den Kunden bzw. dem Product-Owner und dem Entwicklungsteam über die Funktionalitäten des Systems, die zu einem Ergebnis führt, mit dem Entwicklungsteam und Kunde einverstanden sind. Somit bilden User-Stories Anforderungen auf grober Ebene (siehe Kapitel 3 „Von der Idee zur Spezifikation“), deren Verfeinerung in der Diskussion erfolgt. Erst mit der Klärung der Details werden User-Stories vollständig – auch wenn die Ergebnisse dieser Klärungen nicht immer dokumentiert werden.

Eine User-Story beschreibt eine Funktionalität, die für den Kunden oder Benutzer eines Produkts oder Systems von Wert ist. Sie besteht aus der schriftlichen Beschreibung der Funktionalität, Gesprächen über die Funktionalität und Akzeptanzkriterien, die Details vermitteln und festlegen, wann eine User-Story vollständig umgesetzt ist [Cohn10].



Die Form von User-Stories ist nicht in Stein gemeißelt – nehmen Sie Anpassungen vor, falls Ihr Team und Ihre Stakeholder weniger oder andere Informationen benötigen, und beschreiben Sie die (neue) Struktur bei den Notationsformen in Ihrem RE-Leitfaden.

11.2.1 Aufbau einer User-Story

User-Stories können prinzipiell in beliebiger Form dokumentiert werden. In der Praxis hat sich jedoch, unter anderem durch Literatur wie Mike Cohns „User Stories“ [Cohn10] und die zunehmende Normierung durch Trainingsangebote und Zertifizierung, eine Standardform herausgebildet (siehe Abbildung 11.1). User-Stories können in einem Tool verwaltet werden, was Ihnen die Verfolgbarkeit zu anderen Artefakten erleichtert (siehe Kapitel 19 „Strukturen und Mengen“). Häufig werden User-Stories aber auch auf Karteikarten oder große Post-it-Zettel geschrieben, die das Backlog z. B. in Form einer Story-Map visualisieren (siehe Abschnitt 11.2.3 „Von Use-Cases, User-Stories und Story-Maps“). Diese Karten können je nach Bearbeitungsstatus und Zusammenhang leicht umorganisiert werden.

11 Dokumentation im agilen Umfeld

Eine englische Variante finden Sie in Kapitel 10 „Anforderungsschablonen“.

Als <Benutzerrolle>
möchte ich <Funktionalität/Systemverhalten>,
so dass <fachlicher Wert für den Benutzer/Kunden bzw. wirtschaftlicher Nutzen>.

Abbildung 11.1: Standardisierte Form einer User-Story

Mehr zu Rollen finden Sie in Kapitel 18 „Versionen und Zustände“.

<**Benutzerrolle**> steht für eine Gruppe von Benutzern oder Kunden, für die eine Funktionalität gefordert wird. Auf diese Weise können z. B. Rollenkonzepte in User-Stories integriert oder zielgruppenspezifische Aspekte berücksichtigt werden, anstatt nur vom Standardanwender auszugehen. Falls vorhanden, können auch Personas anstelle von Benutzerrollen verwendet werden (siehe auch Kapitel 24 „Requirements und Usability“).

<**Funktionalität/Systemverhalten**> enthält den Kern der Anforderung, also eine Beschreibung der Funktionalität, die das System bereitstellen soll, oder einer Eigenschaft, die das System besitzen soll. Details zu der geforderten Funktionalität oder Eigenschaft werden nicht alle im Vorfeld dokumentiert, sondern im Gespräch erarbeitet.

Formulieren Sie wirklich nur eine Begründung - keine weiteren Anforderungen!

Der optionale dritte Teil, die Beschreibung des <**fachlichen Werts für den Benutzer/Kunden bzw. wirtschaftlichen Nutzens**>, liefert die Motivation für die User-Story. Die Motivation kann als Grundlage für die Priorisierung dienen und Hinweise zu weiteren Aspekten liefern, die für die Funktionalität wichtig sind.

Als Nutzer der Bibliothek
will ich den Bestand nach Büchern eines bestimmten Autors
durchsuchen können,
um alle Bücher meines Lieblingsautors zu finden.

Abbildung 11.2: Beispiel für eine User-Story

11.2.2 Das nehm' ich dir nicht ab! – Akzeptanzkriterien für User-Stories

Siehe Kapitel 14 „Prüftechniken für Anforderungen“

Die Frage, wann eine User-Story umgesetzt und „wirklich“ fertig ist, hat großen Einfluss auf die Geschwindigkeit des Entwicklungsteams bei der Umsetzung (Englisch: velocity) und damit auf die Projektlaufzeit, die Projektkosten und den Erfolg der Abnahme. Die Antwort auf diese Frage beeinflusst aber auch die Qualität des zukünftigen Systems für die Benutzer.

Bzw. dem Product-Owner

Im agilen Kontext sind Akzeptanzkriterien nicht zwangsläufig identisch mit Testfällen, sondern können auch verwendet werden, um ergänzende Details zur User-Story zu erfassen. Als mentaler Anker für die Ergebnisse der Diskussion zwischen den Benutzern/Kunden des Systems und dem Entwicklungsteam müssen Akzeptanzkriterien, wie die User-Stories selbst, nicht bis ins letzte Detail ausformuliert werden [Cohn10].

Akzeptanzkriterien legen fest, unter welchen Bedingungen ein Product-Backlog-Eintrag (z. B. eine User-Story) als umgesetzt gilt und erfolgreich abgenommen wird.

Ähnlich wie bei User-Stories gibt es für die Dokumentation von Akzeptanzkriterien kein festes Format. Sie können beispielsweise in Form von Stichpunkten auf der Rückseite einer User-Story-Karte notiert werden (siehe Abbildung 11.3). Gängige Akzeptanzkriterien beschreiben Voraussetzungen für die Funktionalität der User-Story, erwartete (Mindest-) Reaktionen des Systems und nicht-funktionale Aspekte (siehe auch Kapitel 12 „Nicht-funktionale Anforderungen“).

Als Nutzer der Bibliothek
will ich den Bestand nach Büchern eines bestimmten Autors
durchsuchen können.

- Eine Suche nach einem bestimmten Autorennamen (Nachname und/oder Vorname) gibt eine Liste der vorhandenen Bücher zu diesem Namen aus.
- Falls zu einem Autorennamen oder Begriff keine Ergebnisse gefunden werden, bekommt der Nutzer eine passende Meldung angezeigt.
- Es werden maximal 20 Bücher pro Bildschirmseite angezeigt
- Falls mehr als 20 Titel gefunden werden, kann der Nutzer zwischen den Seiten navigieren und die Treffer einschränken.
- Die Suche dauert nicht länger als fünf Sekunden.

Hier wurde der
optionale dritte Teil
(die Intention)
weggelassen, weil
er fast identisch
mit dem Kern der
Anforderung war.

Abbildung 11.3: Eine User-Story mit Akzeptanzkriterien

Als Qualitätsmaßstab für den Inhalt von testbaren Akzeptanzkriterien kann die SMART-Formel [Wake03] verwendet werden. „SMARTe“ Akzeptanzkriterien sind konkret, messbar, machbar, relevant und zeitlich begrenzt. Wie bei allen Anforderungen können natürlich auch die gängigen Qualitätskriterien aus Kapitel 1 „In Medias RE“ dazu beitragen, aussagekräftige Akzeptanzkriterien zu erhalten.

Englisch:
Specific, Measurable,
Achievable, Relevant,
Time-boxed

Um dem Entwicklungsteam und dem Product-Owner eine solide Basis für die Abnahme der Funktionalität zu bieten, sollten testbare Akzeptanzkriterien weiter detailliert werden. Eine geeignete Notation, die auch mit geringem Aufwand als verhaltensgetriebener Testfall eingesetzt werden kann, ist das Given-When-Then-Schema [North06] (Abbildung 11.4):

Angenommen <hier folgen die geltenden Voraussetzungen>,
wenn <hier folgen eine oder mehrere Aktionen des Benutzers>,
dann <hier folgt bzw. folgen die geforderte(n) Reaktion(en) des Systems>.

Englisch: Given

Englisch: When

Englisch: Then

Abbildung 11.4: Das Given-When-Then-Schema

Für jede Aktion des Benutzers bzw. für jede erwartete Systemreaktion werden mit Hilfe des Given-When-Then-Schemas mindestens der Erfolgsfall und der Fehlerfall beschrieben, bei alternativen Fällen je nach Relevanz auch weitere Systemreaktionen (siehe Abbildung 11.5). Es können durchaus mehrere Akzeptanzkriterien in einem Given-When-Then-Schema behandelt werden, wenn man die Akzeptanzkriterien gemeinsam testen kann.

11 Dokumentation im agilen Umfeld

- Eine Suche nach einem bestimmten Autorennamen (Nachname und/oder Vorname) gibt eine Liste der vorhandenen Bücher zu diesem Namen aus.
- Falls zu einem Autorennamen oder Begriff keine Ergebnisse gefunden werden, bekommt der Nutzer eine passende Meldung angezeigt.
- Es werden maximal 20 Bücher pro Bildschirmseite angezeigt.
- Falls mehr als 20 Titel gefunden werden, kann der Nutzer zwischen den Seiten navigieren und die Treffer einschränken.
- Die Suche dauert nicht länger als fünf Sekunden.



User-Story: Bestandssuche nach Autor

Angenommen der Nutzer hat die Suche initiiert UND der Nutzer gibt einen Autorennamen (Vor- und/oder Nachname) ein UND der Nutzer startet die Suche

Wenn keine Bücher mit dem eingegebenen Autorennamen im Bestand sind

Dann zeigt das System nach spätestens 5 Sekunden dem Nutzer die Fehlermeldung „Ihre Suche erzielte keinen Treffer. Bitte verändern Sie die Suchanfrage“ an UND der Nutzer kann die Suche neu initiieren.

Wenn 1-20 Bücher mit dem eingegebenen Autorennamen im Bestand sind

Dann erzeugt das System innerhalb von 5 Sekunden eine Ergebnisliste mit 20 Treffern auf einer Bildschirmseite

UND zeigt dem Nutzer nach spätestens 5 Sekunden alle gefundenen Treffer auf einer Bildschirmseite an.

Wenn mehr als 20 Bücher mit dem eingegebenen Autorennamen im Bestand sind

Dann erzeugt das System innerhalb von 5 Sekunden eine Ergebnisliste mit einem Seitenumbruch nach 20 Treffern UND zeigt dem Benutzer die ersten 20 Treffer der Liste auf der ersten Bildschirmseite an UND der Nutzer kann mittels Navigationselementen zwischen den Bildschirmseiten navigieren.

Abbildung 11.5: Akzeptanzkriterien mit Ergänzung im Given-When-Then-Schema

Wenn Ihre Stakeholder vor allem in Lösungen denken, können Sie mit dem Given-When-Then-Schema auch Detailinformationen sammeln und diese später zu Akzeptanzkriterien abstrahieren (siehe auch Essenzbildung in Kapitel 6 „Anforderungs-ermittlung“).

Behalten Sie die Aufwände für die Detaillierung im Auge - weniger ist manchmal mehr.

Die ergänzende Dokumentation von Akzeptanzkriterien im Given-When-Then-Schema fördert das Hinterfragen von User-Stories. Benutzer/Kunde des Systems und Product-Owner gewinnen ein klareres Bild davon, was sie mit einer User-Story genau fordern. Zudem bietet die Notation im Given-When-Then-Schema den Vorteil, dass sich eine User-Story mit der richtigen Infrastruktur automatisiert testen lässt.

11.2.3 Von Use-Cases, User-Stories und Story-Maps

Use-Cases und User-Stories haben eine ähnliche Ausrichtung: Sie bilden einzelne Systemfunktionalitäten aus der Sicht eines Benutzers ab und liefern in ihrer Gesamtheit eine Übersicht über das System. Die Betrachtung eines Systems mittels Use-Cases erfolgt jedoch üb-

11 Dokumentation im agilen Umfeld

licherweise auf größerer Ebene als bei einer einzelnen User-Story – ein Use-Case umfasst mehrere oder sogar viele User-Stories. Ein Epic wiederum kann in seinem Umfang einem groben Use-Case entsprechen.

Ein Epic kann aber auch viel größer sein als ein Use-Case!

Dennoch können viele Dokumentationstechniken, die im Zusammenhang mit Use-Cases Anwendung finden, auch auf die Arbeit mit User-Stories übertragen werden. Zum Beispiel lassen sich User-Stories mit Aktivitätsdiagrammen oder Zustandsdiagrammen verfeinern und verwendete Begriffe und Benutzerrollen können mit Hilfe von Begriffsmodellen semantisch präzisiert werden. Weitere Informationen zur Verfeinerungen mit Hilfe von Diagrammen finden Sie in Kapitel 9 „Systemanforderungen dokumentieren“. Wie Sie dann diese Diagramme mit Ihren User-Stories verknüpfen können, lernen Sie in Kapitel 19 „Strukturen und Mengen“.

Beziehen Sie unbedingt das Team in die Entscheidung über die Dokumentationsformen mit ein – durch reine Festlegungen schaffen Sie nur zusätzlichen unerwünschten Aufwand und untergraben die Freiheit des Entwicklungsteams, sich selbst organisieren zu dürfen.

Und in Kapitel 4 „Agile und andere Vorgehensweisen“ finden Sie Indikatoren dafür, wie viel Sie in Ihrem agil durchgeführten Projekt dokumentieren sollten.

Die zusätzlichen Informationen in Form von Diagrammen fördern ein umfassenderes Verständnis des Systems bei allen Projektbeteiligten und können, wie auch andere Formen der Visualisierung, in Diskussionen für höhere Transparenz sorgen.

Ein Beispiel für die Visualisierung des Product-Backlogs ist das Story-Mapping [Patton08], siehe Abbildung 11.6: Auf Basis von Szenarien aus Benutzersicht werden die essenziellen Grundfunktionalitäten des Systems festgelegt. Sie bilden in einer horizontalen Reihe angeordnet das „Rückgrat“ (Englisch: backbone) des Systems. Die Ebene des „Rückgrats“ fließt nicht in die Planung der Umsetzung ein. Am „Rückgrat“ werden in der nächsten Reihe darunter die User-Stories des „Walking Skeleton“ aufgehängt. Das „Walking Skeleton“ ist eine Minimalversion des Systems, die einen kleinen End-to-End-Prozess abbildet [Cockburn04]. Unterhalb des Walking Skeleton können nun weitere User-Stories oder Details zu User-Stories hinzugefügt werden. Dabei gilt, dass die Priorität umso niedriger ist, je weiter unten eine User-Story hängt. Das Ergebnis ist eine „Karte“ der User-Stories (Englisch: Story-Map), welche die Umsetzungspriorität und Reihenfolge abbildet:

Das Rückgrat entspricht meist den „Epic“ eines Systems.

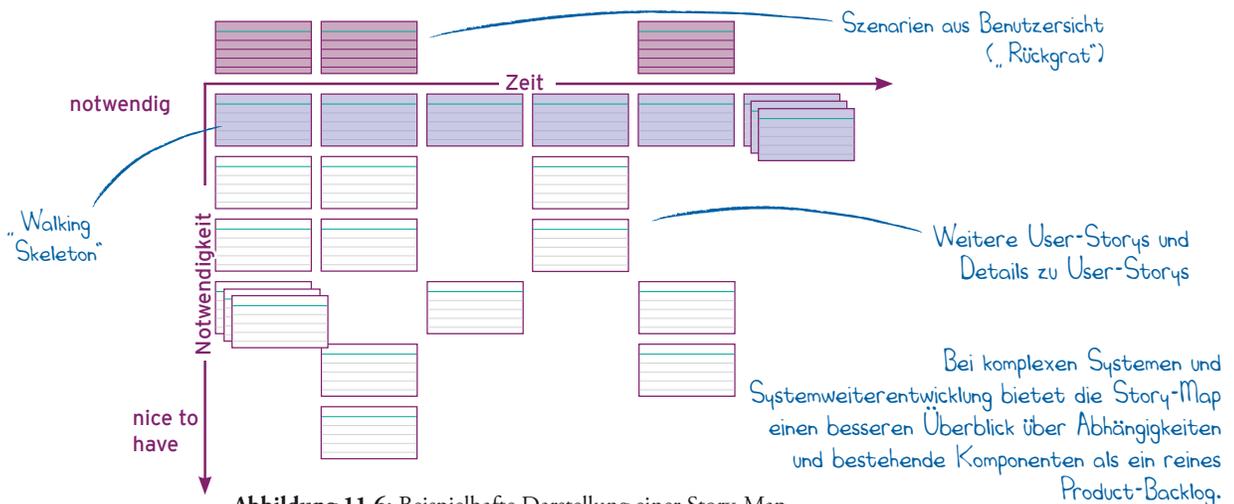


Abbildung 11.6: Beispielhafte Darstellung einer Story-Map

Bei komplexen Systemen und Systemweiterentwicklung bietet die Story-Map einen besseren Überblick über Abhängigkeiten und bestehende Komponenten als ein reines Product-Backlog.

11 Dokumentation im agilen Umfeld

Für verteilt arbeitende Teams können Story-Maps auch in virtueller Form eingesetzt werden.

Die Story-Map lässt sich übrigens auch – ähnlich wie ein Walkthrough – zur Ermittlung und Prüfung von User-Stories einsetzen (siehe Kapitel 14 „Prüftechniken für Anforderungen“). Die entsprechende Technik wird „Walking the map“ (Deutsch: an der Karte entlanggehen) genannt, weil man – vorausgesetzt, dass die Story-Map klassisch in Form von Karteikarten oder Post-its umgesetzt ist – als Requirements-Engineer oder Product-Owner mit dem Stakeholder an der Karte entlangwandern kann. Der Requirements-Engineer bzw. Product-Owner erklärt dabei die Funktionalitäten, wie er sie verstanden hat, und der Stakeholder hat die Möglichkeit, Missverständenes zu korrigieren und die User-Stories bei Bedarf zu ergänzen.

11.3 Technical Storys

Neben User-Stories kann es in einem Product-Backlog auch Storys geben, die keine Funktionalität beschreiben. Diese werden oft als Technical Storys bezeichnet und umfassen technische oder andere nicht-funktionale Aspekte, die nicht über die Implementierung einzelner User-Stories abgedeckt werden.

Technical Storys sind in der agilen Community umstritten. Ein Teil lehnt sie mit der Begründung ab, dass viele Inhalte von Technical Storys korrekterweise Teil der Definition of Done oder des entsprechenden agilen Prozesses sein sollten [Jeffries10] [Wake12]. Technical Storys erzeugen keinen direkten Wert für den Benutzer, sind aber unserer Erfahrung nach gerade für unerfahrene Teams hilfreich: Sie machen die technischen Aufwände hinter User-Stories sichtbar und unterstützen die Teammitglieder somit bei der Planung der Menge an User-Stories, die sie in einem Entwicklungszyklus bewältigen können.

Es wurden auch schon „Technical User-Storys“ und „System-User-Storys“ mit einem System als Benutzer gesichtet.

Es herrscht oftmals Unklarheit darüber, was genau unter einer Technical Story zu verstehen ist. Als synonyme Begriffe für Technical Story werden in den meisten Fällen „Technical Task“ (ursprünglich die Beschreibung einer technischen Aufgabe im eXtreme Programming) oder „Constraint“ (Deutsch: Randbedingung) genannt. Das Synonym „Constraint“ macht deutlich, dass Technical Storys einen starken Bezug zu nicht-funktionalen Aspekten im Projekt haben, seien diese nun organisatorischen oder technischen Ursprungs. Abweichend von der Definition von Randbedingungen nach IREB [IREB] schränken Technical Storys jedoch nicht zwingend den Lösungsraum für die Umsetzung von Funktionalitäten ein.

Neben Akzeptanzkriterien für einzelne User-Stories und der Definition of Done als Qualitätsrichtlinie für den Gesamtprozess und das erzeugte Produkt stellen Technical Storys eine weitere Möglichkeit dar, nicht-funktionale Aspekte des Systems abzubilden.

Je nach den Gegebenheiten im Projekt (z. B. der Qualität des gelebten Prozesses, der Erfahrung des Entwicklungsteams, dem gemeinsamen Verständnis der Definition of Done) können Technical Storys zu ehrlicheren Aufwandsschätzungen und besserer Kommunikation zwischen dem Entwicklungsteam und dem Product-Owner bzw. Kunden beitragen. Allerdings sollte der Unterschied zwischen Technical Storys und User-Stories klar kenntlich gemacht werden, auch um Missverständnissen bezüglich der Priorisierung bei allen Beteiligten von Anfang an vorzubeugen.



11.3.1 Aufbau von Technical Storys

Als Format für eine Technical Story eignet sich eine Taskbeschreibung mit Motivation (siehe Abbildung 11.7):

Die Motivation erleichtert dabei die Diskussion über die Technical Story, ähnlich wie die Informationen zum fachlichen Wert/wirtschaftlichen Nutzen einer User-Story.

Ersetzt magic numbers durch Konstanten, so dass der Code leichter lesbar und wartbar wird.

} Taskbeschreibung
} Motivation

Abbildung 11.7: Technical Story als Taskbeschreibung mit Motivation

Im Kontext von unternehmensinternen Entwicklungsprojekten kann auch mit „internen Kundenrollen“ gearbeitet werden. In diesem Fall können auch interne Rollen als Benutzer verwendet werden. Technical Storys in diesem Format sind bis auf die Rollenbezeichnung identisch mit User-Storys (siehe Abbildung 11.8), können aber z. B. durch Dokumentation in einem eigenen Dokument (einem zweiten Backlog) oder einem eigenen Abschnitt im Product-Backlog von den User-Storys abgegrenzt werden.

Z. B. das Entwicklungsteam, die Systemadministration oder die Wartung

Als Entwickler möchte ich, dass magic numbers durch Konstanten ersetzt werden, so dass der Code leichter lesbar und wartbar wird.

Abbildung 11.8: Technical Story mit interner Kundenrolle

11.3.2 Die Priorisierungsproblematik

Unabhängig davon, welches Format zur Dokumentation von Technical Storys verwendet wird, kann die gleichzeitige Existenz von Technical Storys und User-Storys den Product-Owner vor ein Priorisierungsproblem stellen. User-Storys besitzen einen fachlichen Wert für den Benutzer/Kunden und erzeugen somit automatisch einen Return on Investment (RoI). Technical Storys haben im Gegensatz dazu keinen direkten fachlichen Wert und ein Kunde würde auch nicht ohne Weiteres für die reine Implementierung einer Technical Story bezahlen – besonders, wenn die Umsetzung der Technical Story im aktuellen Sprint mit Verzögerungen in Bezug auf zusätzliche Systemfunktionalitäten verbunden wäre.

Der Benutzer ist bereit, für die Funktionalität zu bezahlen.

Ein Lösungsansatz für dieses Dilemma ist die Zuordnung von Technical Storys zu User-Storys, z. B. in Form von zusätzlichen Tasks für die Umsetzung. Auf diese Weise wird für den Benutzer/Kunden transparenter, weshalb die Umsetzung einer Funktionalität potenziell länger dauert, und der Aufwand kann verhandelt werden.

Im Gegensatz zu „Pauschalauflagen“ bei Schätzungen

Im Fall von Technical Storys, die sich keiner User-Story zuordnen lassen, z. B. im Zusammenhang mit wiederkehrenden Aktivitäten wie Fehlerbehebung oder Refactoring, lohnt es sich, den gelebten Prozess zu untersuchen, denn unter Umständen deuten diese Technical Storys auf eher grundlegende Prozessprobleme hin.

= Umstrukturieren der Architektur oder des Codes

11 Dokumentation im agilen Umfeld

Mehr dazu finden Sie unter www.sophist.de/re6/kapitel9.

Eine weitere Problematik bei der Priorisierung ergibt sich aus der Tatsache, dass ein Product-Owner als vorwiegend fachlicher Domänen- und Produktexperte nicht unbedingt über das notwendige technische Verständnis verfügt, um die Auswirkungen und den potenziellen Aufwand für eine Technical Story realistisch einschätzen zu können. Ein Product-Owner-Team, das auch einen technisch versierten Experten mit einschließt, oder Zusammenarbeit zwischen Entwicklungsteam und Product-Owner bei der Abschätzung von technischen Aspekten können hier Abhilfe schaffen.

Z. B. Epics

11.4 User-Stories schneiden und verfeinern

Schneiden verringert die Anzahl der Funktionalitäten in einer User-Story. Mehr zur Verfeinerung finden Sie in Kapitel 3 „Von der Idee zur Spezifikation“.

Manchmal ist eine User-Story schlicht zu groß, um eine realistische Schätzung abgeben zu können. Ein anderes Mal verbergen sich innerhalb einer User-Story mehrere zusammenhängende User-Stories. Erst nach dem Schneiden bzw. Verfeinern solcher User-Stories ist eine realistische Sprint-Planung möglich.

Wer sich mit dem Schneiden von User-Stories beschäftigt, kommt früher oder später mit dem INVEST-Prinzip von Bill Wake [Wake03] und den „Patterns for Splitting User Stories“ von Richard Lawrence [Lawrence09] in Berührung.

Eine „gute“ User-Story sollte nach dem von Bill Wake entwickelten *INVEST*-Prinzip *unabhängig*, *verhandelbar*, *von Wert*, *schätzbar*, *klein* und *testbar* sein. Diese Eigenschaften sollten auch beim Schneiden von User-Stories als Qualitätskriterien im Hinterkopf behalten werden. Richtige Schneidungsprinzipien stellen sie jedoch nicht dar.

Englisch:
Independent,
Negotiable,
Valuable,
Estimable,
Small,
Testable

Die von Richard Lawrence zusammengetragenen Patterns, z. B. das Schneiden nach Workflowschritten (Pattern #1) oder nach Variationen von verarbeiteten Daten (Pattern #5), bieten im Gegensatz zum reinen INVEST-Prinzip eine praktischere Herangehensweise an die Schneidungsproblematik, lassen aber die Frage, wann welches Schneidungsmuster angewendet werden sollte, weitgehend unberücksichtigt.

Die einzige Ausnahme bildet das Herausbrechen von Spikes (Pattern #9). Immer wenn das Entwicklungsteam nicht genug Kenntnisse hat, um eine User-Story umsetzen zu können, kann es einen sogenannten Spike aus dem Product-Backlog herausbrechen, d. h. eine Story erstellen, deren Inhalt die Analyse des unbekanntes Sachverhalts ist. Die eigentliche User-Story wird frühestens im nächsten Sprint betrachtet und kann dann auf Basis der gewonnenen Erkenntnisse geschätzt werden.

11.4.1 Das Meta-Pattern

Die ersten acht Schneidungsmuster der „Patterns for Splitting User Stories“ sind prinzipiell geeignet, um User-Stories unter Wahrung eines Business-Value zu schneiden. Bei näherer Betrachtung lassen sie sich vier Grundaspekten zuordnen, entlang derer eine Schneiden erfolgen kann: nach Fachlichkeit (d. h. nach „Prozess“ oder nach „Daten“), nach „Qualität“ und nach „Technik“. Diese vier Aspekte stellen die erste der zwei Dimensionen des Meta-Patterns dar.

Z. B. Arbeitsschritte oder Workflows

„Prozess“ bezieht sich auf die Einzelvorgänge innerhalb eines abgeschlossenen Sachverhalts (Englisch: end-to-end process), die zusammen eine vollständige Sequenz abbilden. Dabei können Arbeitsschritte, Fehlerfälle, alternative Abläufe, Umwege und Zustandsabhängigkeiten berücksichtigt werden.

11 Dokumentation im agilen Umfeld

Betrachtung nach diesem Aspekt beantwortet die Frage nach dem „Was?“.

„Daten“ bezieht sich auf die Daten, die für den Sachverhalt notwendig sind, der im Rahmen der User-Story gefordert wird. Dabei können zu verarbeitende Datenmengen, Generalisierungen, Gruppierungen, Parameter, Attribute und sonstige Abhängigkeiten der Daten berücksichtigt werden.

Die Betrachtung unter diesem Aspekt beantwortet die Frage nach dem „Womit?“.

„Qualität“ bezieht sich auf die nicht-funktionalen Anforderungen an den Sachverhalt in der User-Story. Dabei können einzelne qualitative Kriterien wie Performance, Benutzbarkeit, Modularität oder Stabilität getrennt von der eigentlichen Funktionalität berücksichtigt werden.

Die Betrachtung beantwortet die Frage nach dem „Wie?“.

„Technik“ umfasst die technischen Aspekte eines Sachverhalts. Die Betrachtung des technischen Aspekts ist problematisch, da er sich zu sehr auf die zugrunde liegende Lösung und nicht auf die Anforderungen an den Sachverhalt bezieht. Zudem werden beim Schneiden nach Technik leicht INVEST-Regeln verletzt, vor allem die Unabhängigkeit von User-Stories sowie ihr Wert für den Kunden. Aus diesem Grund werden wir den technischen Schneidungsaspekt nicht weiter betrachten.

Z. B. die Systemarchitektur bei Softwaresystemen

Die Existenz von Themes zeigt, dass die Unabhängigkeit nicht immer gewährleistet werden kann.

Kompliziertheit bildet die zweite Dimension im Meta-Pattern. Man kann User-Stories so schneiden, dass man zuerst die einfachsten oder zuerst die komplizierten – und damit risikoreichsten – Aspekte betrachtet. „Einfach“ hieße für den Prozessaspekt, nur den minimalen Ablauf zu nehmen, komplizierter wäre die Betrachtung verschiedener Sonder-, Ausnahme- oder Fehlerfälle. Für Daten können Sie sich aufs absolut notwendige Minimum beschränken oder, komplizierter, alle Daten betrachten, die Sie gerne verarbeiten möchten. Bei Qualität könnte „kompliziert“ heißen, dass Sie viel Wert auf Effizienz, Benutzbarkeit, Stabilität und all die anderen Qualitätsaspekte legen – hier wäre „einfach“, dass das System erst einmal tut, was es soll, auch wenn Performanz, Optik oder Benutzbarkeit noch stark zu wünschen übrig lassen. Beide Herangehensweisen haben Konsequenzen (siehe Abbildung 11.9).

	Kompliziert	Einfach
Fachlichkeit (Daten, Prozesse)	Kein „Rauspicken der Rosinen“; früheres Erkennen von Projektrisiken und Wissensdefiziten.	Frühere Erfolgserlebnisse, frühere Rückmeldungen
Qualität	Selbst einfache Funktionalität kommt erst spät.	Potenziell geringere Nutzerakzeptanz
Technik	Stabile, tragfähige Architektur, einfache Erweiterbarkeit; Gefahr des Over-Engineerings	Wahrscheinlich höhere Refactoring-Aufwände; schnellerer Wissensaufbau im Team

Abbildung 11.9: Konsequenzen in Bezug auf Kompliziertheit

Anhand der Tabelle aus Abbildung 11.9 können Sie das passende Vorgehen für Ihre Projektrahmenbedingungen wählen. Haben Sie einen sehr ungeduldigen Kunden, der auf schnelle Ergebnisse drängt? Dann schneiden Sie nach einfacher Fachlichkeit: Bilden Sie zum Beispiel nur den Gut-Fall ohne die Fehler- und Ausnahmefälle ab oder integrieren Sie vorerst

11 Dokumentation im agilen Umfeld

nur die absolut notwendigen Daten. Auf diese Weise können Sie Ihrem Kunden früh erste Ergebnisse präsentieren. Ist Ihrem Kunden, zum Beispiel auf Grund schlechter Erfahrungen in früheren Projekten, eine gute Benutzbarkeit des Systems besonders wichtig – auch wenn das eine längere Wartezeit bedeutet, bis Ergebnisse sichtbar sind? Dann investieren Sie mehr Zeit ins Usability-Engineering und schneiden nach komplizierten Qualitätsaspekten.

11.4.2 Der Minimal-Ansatz und der Reduktions-Ansatz

Wie bleibt nun beim Schneiden von User-Stories der fachliche Wert für den Benutzer/Kunden erhalten? Die Antwort auf diese Frage haben wir in zwei grundlegenden Schneidungsprinzipien herausgearbeitet: dem Minimal-Ansatz (siehe Abbildung 11.10) und dem Reduktions-Ansatz (siehe Abbildung 11.11).

Der Minimal-Ansatz

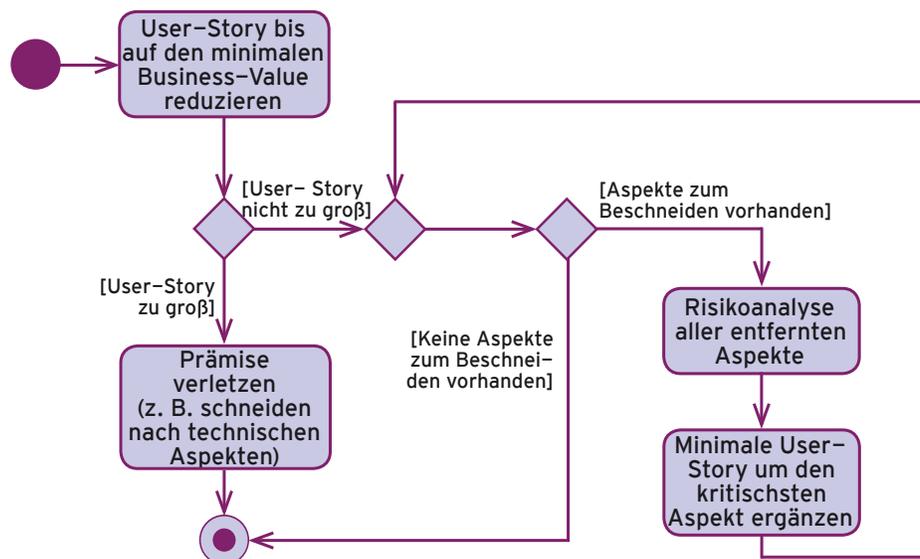


Abbildung 11.10: Der Minimal-Ansatz zur User-Story-Schneidung

Beim Minimal-Ansatz wird ermittelt, welcher Teil des Kerns einer User-Story eine Grundfunktionalität vollständig abbildet. Für den Prozessaspekt wäre das die kürzeste vollständige Abfolge von Einzelvorgängen, also der kleinste in sich geschlossene Prozess, der einen fachlichen Wert für den Kunden bzw. Nutzer erzeugt. Für den Datenaspekt würde nur die für die Funktionalität unerlässliche Datenmenge betrachtet werden und für den Qualitätsaspekt müssten Sie abwägen, ob z. B. für eine Software-Anwendung zunächst auf Schnelligkeit, eine aufwendige Benutzeroberfläche oder anderes verzichtet werden kann. Unterziehen Sie alle gefundenen Aspekte (Prozess, Daten, Qualität) einer vorbereitenden Risikoanalyse und priorisieren Sie sie.

Falls selbst der minimale Sachverhalt zu groß für einen Sprint ist, besteht nur die Möglichkeit, von der Prämisse abzuweichen, dass in jedem Sprint ein Produkt mit fachlichem Wert für den Kunden/Benutzer erzeugt werden muss. So entsteht größerer Spielraum dafür, den Umfang

weiter zu verkleinern, indem zum Beispiel nach technischen Aspekten geschnitten werden kann.

Wenn Sie feststellen, dass der geschätzte Aufwand des minimalen Sachverhalts bequem in einer Iteration zu bewältigen ist, können Sie weitere Prozesse, mehr Daten oder Qualitätsanforderungen hinzunehmen: Erweitern Sie den minimalen Sachverhalt um zusätzliche Teile aus den drei Aspekten, bis der Aufwand dem Umfang entspricht, der in einer Iteration umgesetzt werden kann. Die Erweiterungsaspekte sollten mit Hinblick auf die Ergebnisse der Risikoanalyse gewählt werden.

Erst die riskantesten Aspekte!

Natürlich können Sie auch mehrere User-Stories in einer Iteration bearbeiten.

Der Reduktions-Ansatz

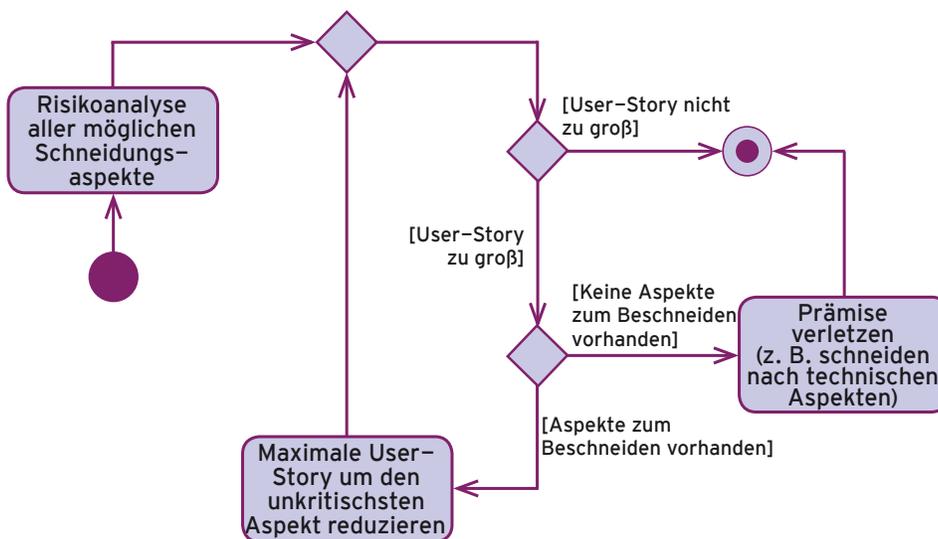


Abbildung 11.11: Der Reduktions-Ansatz zur User-Story-Schneidung

Bei Einsatz des Reduktions-Ansatzes wird zuerst die aufwendigste Form des Sachverhalts ermittelt. Zum Beispiel für den Prozessaspekt die Abfolge von Einzelvorgängen, die auch z. B. Fehlerfälle oder Optionen einschließt, für den Datenaspekt die Datenmenge, die auch nicht zwingend notwendige Zusatzinformationen umfasst, und für den Qualitätsaspekt die jeweils optimale Ausprägung der Kriterien, z. B. eine minimale Reaktionszeit oder ein aufwendiges Benutzeroberflächendesign. Dieser maximale Teilsachverhalt wird danach so lange um die Aufwände reduziert, die als am jeweils wenigsten wichtig eingestuft wurden, bis er dem gewünschten Umfang entspricht.

Bei dieser Betrachtung werden Sie wahrscheinlich feststellen, dass einige der gefundenen Prozesse eine Basis für gute separate User-Stories abgeben können!

Falls keine weiteren Teilsachverhalte zum Beschneiden der User-Story mehr vorhanden sind und der Umfang weiterhin zu groß für einen Sprint ist, kann auch beim Schneiden nach dem

11 Dokumentation im agilen Umfeld

Reduktions-Ansatz die Prämisse verletzt werden, wonach in jedem Sprint ein Produkt mit fachlichem Wert für den Kunden/Benutzer erzeugt werden muss.

11.5 Wann ist fertig wirklich „fertig“? – Die Definition of Done (DoD) und die Definition of Ready (DoR)

Unterschiedliche Rollen im Entwicklungsprozess können ein abweichendes Verständnis davon haben, wann ein Artefakt, sei es nun eine User-Story oder das Inkrement, „fertig“ ist. Dem einen Entwicklungsteam reicht ein Einzeiler als User-Story, andere möchten mindestens drei bis fünf Akzeptanzkriterien, um die Ergebnisse der Diskussion über die Inhalte festzuhalten. Für manche genügt lauffähiger Code, andere sehen möglicherweise die Dokumentation als Teil des Systems, der ebenso erstellt werden muss. Um diese unterschiedlichen Meinungen zu konsolidieren und ein gemeinsames Verständnis von „fertig“ zu schaffen, können im agilen Umfeld zwei Konzepte eingesetzt werden: die „Definition of Done“ und die „Definition of Ready“.

11.5.1 Die Definition of Done – weil's gut werden muss

Die Definition of Done bezieht sich immer auf das Ergebnis eines Entwicklungszyklus (nicht auf eine einzelne Story!). Sie kann bei Bedarf über den Projektverlauf angepasst werden.

Der Scrum-Guide [SCRUMGUIDE] definiert die Definition of Done als ein Artefakt, das bei allen Projektbeteiligten ein gemeinsames Verständnis dafür erzeugt, welche formalen Kriterien erfüllt sein müssen, damit die Arbeit an einem System- oder Produktinkrement als abgeschlossen gilt.

Scrum setzt zudem voraus, dass ein fertiges Produktinkrement potenziell auslieferbar (Englisch: shippable) sein muss. Streng genommen bedeutet das, dass der Product-Owner am Ende des Sprints ein Produkt erhält, das er sofort, ohne weitere „Abschlussarbeiten“ an den Kunden ausliefern könnte.

Die Definition of Done muss jedes Unternehmen und sogar jedes Projekt für sich definieren. Sie kann genauso im Großausdruck bei den Entwicklern an der Wand hängen wie auch elektronisch allen zugänglich abgelegt sein.

Üblicherweise enthält eine Definition of Done diese minimalen Forderungen:

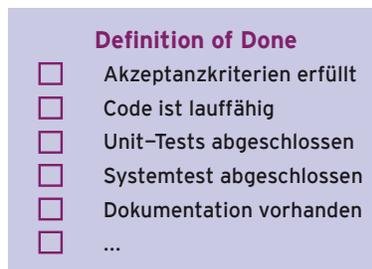


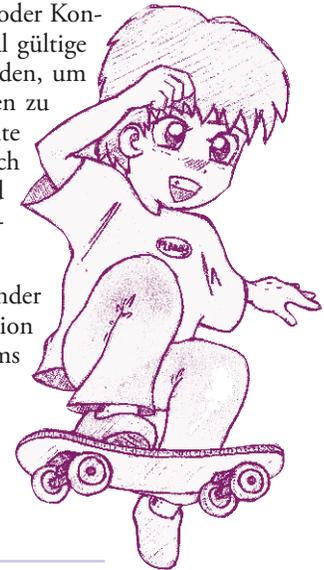
Abbildung 11.12: Beispiel für eine Definition of Done

Wenn man diese Voraussetzungen ernst nimmt, ist Scrum als Methodik für manche Projekte nicht unbedingt geeignet, z. B. für Hardware-entwicklung.

Neben der Erfüllung aller Akzeptanzkriterien, lauffähigem Code und vorhandener Dokumentation kann z. B. noch eine Rechtschreibprüfung der erzeugten Dokumente oder Konformität zu Unternehmensrichtlinien wie Styleguides gefordert sein. Auch global gültige nicht-funktionale Anforderungen können in der Definition of Done erfasst werden, um sie nicht redundant auf feinerer Ebene (z. B. als Akzeptanzkriterien) beschreiben zu müssen. Beispiele hierfür sind globale Performance-Anforderungen an jedes erstellte Inkrement (Abfragen, Berechnungen ...) oder wiederkehrende Tätigkeiten, die nach jedem Sprint erfolgt sein müssen. Diese können in Abhängigkeit der Domäne und der eingesetzten Technologien sehr unterschiedlich sein und lassen sich nicht verallgemeinern.

Die Definition of Done ist entwicklungsteamspezifisch und umfasst mit zunehmender Erfahrung des Entwicklungsteams immer mehr Aktivitäten. Daher wird die Definition of Done auch als Maß für die Reife (Englisch: maturity) eines Entwicklungsteams und des gelebten Prozesses angesehen.

Erfahrene, gut eingespielte Teams sind produktiver in Sprints.



11.5.2 Die Definition of Ready – das Quality-Gate für User-Stories

Die Definition of Ready wird verwendet, um zu prüfen, ob User-Stories klar und konkret genug sind, um vom Entwicklungsteam im Sprint umgesetzt werden zu können. Eine User-Story ist nicht „Ready“, solange der Product-Owner sie nicht so erklären kann, dass das Team genau versteht, was zu tun ist. Das Entwicklungsteam kann während der Sprint-Planung sogar User-Stories zurückweisen, die nicht Ready sind, ist aber im Gegenzug daran gebunden, die Definition of Done für akzeptierte User-Stories zu erfüllen.

Die Product-Backlog-Einträge, mit denen sich das Entwicklungsteam im kommenden Sprint beschäftigen soll, werden so weit verfeinert, dass jeder von ihnen innerhalb der Time Box des Sprints auf „Done“ gebracht werden kann. Die Product-Backlog-Einträge, für die das der Fall ist, werden als „Ready“ angesehen – bereit für die Auswahl durch das Entwicklungsteam in einem Sprint Planning. [SCRUMGUIDE]

Mit Hilfe der Definition of Ready sollen vor allem unnötige Verzögerungen und Demotivation durch unklare Aufgabenstellungen oder nicht identifizierte Abhängigkeiten vermieden werden. Außerdem verhindert der zusätzliche Kontrollschritt, dass User-Stories unreflektiert in den Entwicklungsprozess gelangen.

Eine Definition of Ready kann Sie davor schützen, erst mitten im Sprint festzustellen, dass die Beteiligten sehr unterschiedliche Vorstellungen vom Inhalt der User-Story hatten.

Der Product-Owner ist dafür verantwortlich, dass User-Stories ausreichend verfeinert werden, und die verantwortlichen Stakeholder müssen sich gezielt Gedanken über ihre Forderungen machen. Unausgegrenzte, vage Ideen sind bei konsequentem Einsatz der Definition of Ready nicht mehr zulässig, wodurch die Chancen des Entwicklungsteams steigen, das Inkrement so fertigzustellen, dass es der Definition of Done genügt.

Bei zu vagen User-Stories lohnt es sich, die Definition of Ready und die Akzeptanzkriterien zu überprüfen – werden genug Details erhoben und dokumentiert?

Machen Sie sich die Regeln des SOPHIST-Regelwerks aus Kapitel 7 zu Nutze!

11 Dokumentation im agilen Umfeld

Die Definition ist auch perfekt als Checkliste für unerfahrene Product-Owner geeignet!

Die folgenden Punkte könnten in einer Definition of Ready stehen:

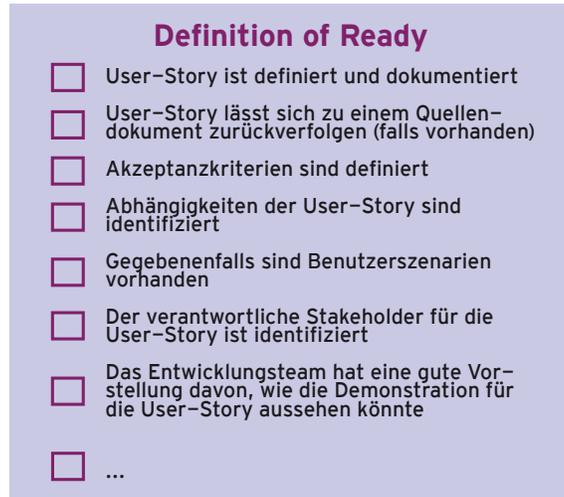


Abbildung 11.13: Beispielhafte Definition of Ready

Die Inhalte der Definition of Ready werden wie die Inhalte der Definition of Done gemeinsam von Entwicklungsteam und Product-Owner festgelegt und sind von diversen Faktoren abhängig, beispielsweise von der Arbeitsweise des Entwicklungsteams, der Vertrautheit mit der fachlichen und technischen Domäne oder der Häufigkeit und Qualität der Kommunikation zwischen Entwicklungsteam und Product-Owner. Genau wie die Definition of Done kann auch die Definition of Ready über den Projektverlauf hinweg angepasst werden.

11.6 And all together now! – Wann setze ich welche Technik ein?

Die Verwendung des User-Story-Formats für Anforderungen ist vor allem Geschmackssache – Sie können problemlos auch in einem agilen Projekt mit „traditionellen“ Anforderungen arbeiten. Die anderen vorgestellten Dokumentationstechniken können allerdings in Abhängigkeit von den Rahmenbedingungen Ihr Projekt positiv beeinflussen. Somit stellt sich die Frage:

Was verwende ich wann?

Abbildung 11.14 ist eine Übersicht unserer gesammelten (und eifrig diskutierten) Erfahrungen mit der Verwendung der vorgestellten Dokumentationstechniken. Gehen Sie folgendermaßen vor, um zu ermitteln, von welchen Techniken Ihr Projekt profitieren könnte:

1. Prüfen Sie, welche Randbedingungen auf Ihr Projekt zutreffen. Konzentrieren Sie sich dabei auf die markantesten drei bis vier.
2. Sehen Sie sich die Empfehlungen in Bezug auf Ihre ausgewählten Randbedingungen an.

3. Wägen Sie ab, welche Techniken Sie einsetzen können und möchten. Je stärker die entsprechenden Techniken empfohlen sind, desto eher sollten Sie sie in die engere Wahl nehmen.

++ sehr empfohlen		0 neutral	Technical Stories	Definition of Done	Definition of Ready	Akzeptanzkriterien	Detaillierung mit Given-When-Then-Schema	Story-Map
+ empfohlen		- nicht empfohlen						
Menschliche Einflussfaktoren								
Neues/unerfahrenes Entwicklungsteam			0	++	++	++	+	++
Neuer/unerfahrener Product-Owner			+	++	++	++	0	++
Geringes Abstraktionsvermögen (Stakeholder)			0	+	+	+	+	++
Geringe Verfügbarkeit des Product-Owners			0	++	++	++	+	+
Ungenauere Schätzung von Aufwänden			+	++	++	++	0	0
Organisatorische Einflussfaktoren								
Niedrige Transparenz von Entwicklungsaufwänden			++	0	++	+	0	0
Unklarheiten bei User-Stories im Sprint			0	++	++	++	+	0
Knappe Ressourcen, enger Zeitrahmen			0	+	+	+	-	+
Fachlich-inhaltliche Einflussfaktoren								
Niedrige Qualität der User-Stories			0	++	+	++	+	+
Große User-Stories			0	++	++	++	+	0
Widersprüchliche User-Stories			0	+	+	+	++	++
Niedrige Qualität der Produktinkremente			0	+	++	++	++	0
Komplexer Sachverhalt			0	++	+	+	++	++
Verwendung komplexer/neuer Technologien			++	+	+	++	0	0
Hohe Kritikalität des Sachverhalts			+	++	++	++	++	+

Das vorher erworbene Wissen macht sich später bezahlt.

Abbildung 11.14: Dokumentationstechniken und Rahmenbedingungen

Nehmen wir an, Sie haben ein Projekt mit einem unerfahrenen Entwicklungsteam, knappen Ressourcen bzw. einem engen Zeitrahmen und einem komplexen Sachverhalt, der umgesetzt werden soll.

Unser herzliches Beileid!

Zusammengenommen könnte sich daraus folgende Auslegung ergeben:

Sie verzichten vorerst auf Technical Stories, um zu prüfen, ob der Zusatzaufwand für die Verwaltung und Priorisierung gerechtfertigt ist. Bei User-Stories muss allerdings in Bezug auf Inhalt, Interpretation und Aufwand Klarheit herrschen. Um nicht unnötig mit der Umsetzung in Verzug zu geraten (z. B. weil die Storys noch nicht ausreichend spezifiziert sind), wird eine Definition of Ready definiert. Aufgrund der fehlenden Erfahrung beim Vorgehen legen Sie zusammen mit dem Entwicklungsteam genau fest, welche Aktivitäten für jedes Produktinkrement durchgeführt werden müssen (Definition of Done), und erklären auch Akzeptanzkriterien für verpflichtend, damit die gegenseitigen Erwartungen dokumentiert werden.

11 Dokumentation im agilen Umfeld

Die Detaillierung der Akzeptanzkriterien mit dem Given-When-Then-Schema würde vor allem weiteren Aufwand erzeugen – Sie machen daher die Entscheidung davon abhängig, ob die Tester mit diesem Format umgehen können, so dass die Detaillierung die Basis für Testfälle bilden könnte. Eine Story-Map hätte Vorteile wegen der Komplexität des Sachverhalts, scheidet aber aufgrund der begrenzten Ressourcen/Zeit und des höheren Verwaltungsaufwands im Vergleich zum Standard-Product-Backlog aus.

Agile Requirements: Beyond User Stories

By Ellen Gottesdiener and Mary Gorman

Many agile teams rely on user stories to communicate product needs. While stories are an excellent way to generate ideas, they are insufficient to address all product requirements. Successful agile teams utilize structured conversations to continually discover high-value requirements across the 7 Product Dimensions, considering three planning horizons. Let's explore each of these concepts—the 7 Product Dimensions, the structured conversation, and the three planning horizons—in more detail.

The 7 Product Dimensions

A successful product provides value to its stakeholders, who work together as product partners to discover and deliver that product. These product partners include internal and external customers, business and technical people, and others, such as regulators and suppliers. Each of the product partners brings different ideas of success—unique perspectives that can be expressed across 7 Product Dimensions: user, interface, action, data, control, environment, and quality attribute.

						
User	Interface	Action	Data	Control	Environment	Quality Attribute
Users interact with the product	The product connects to users, systems, and devices	The product provides capabilities for users	The product includes a repository of data and useful information	The product enforces constraints	The product conforms to physical properties and technology platforms	The product has certain properties that qualify its operation and development

Figure 1: The 7 Product Dimensions

Image Source: Discover to Deliver: Agile Product Planning and Analysis, Gottesdiener and Gorman, 2012

Together, the 7 Product Dimensions give the product partners a holistic, comprehensive understanding of the product. No single dimension, by itself, is sufficient. User stories, while valuable, typically only address 2 of the 7 product dimensions: user and action. In other words, a story states a user's goal for interacting with the product and the capabilities (actions) the product provides for the user. But what about the other essential dimensions? How does the product interface with users, systems, and devices? What data

will the product receive or supply? What controls must the product enforce? What is the environment the product must conform to? What are the quality attributes that qualify the product's operation and development? All of these aspects must be considered as well. An effective way to explore the totality of the 7 Product Dimensions is through the structured conversation.

The Structured Conversation

The structured conversation is a metaphor for the ongoing, systematic, and collaborative discovery of product options, within and across all 7 Product Dimensions. (An option represents a potential product need—one possible way to fulfill the product's vision, goals, and objectives.) Many agile teams refer to the structured conversation as discovery or building and refining (or grooming) the product backlog. Whatever you call it, the structured conversation is a lightweight framework that guides the partners as they learn about the product's possibilities and decide what to deliver.

The structured conversation involves three key activities: explore, evaluate, and confirm. Unlike simply writing user stories, which typically focuses only on one or two dimensions, the structured conversation allows product partners to explore a myriad of options across the 7 Product Dimensions. Then, they evaluate each possible solution, considering its benefits, risks, and dependencies, to determine a cohesive chunk of high-value options. Because the product's value must be transparent and measurable, the product partners must also confirm the candidate solution.

On agile projects, the confirmation activities of verification and validation are intertwined; in short delivery cycles the partners verify that the solution was built correctly (using acceptance criteria or conditions of satisfaction) and validate that it was right thing to build (using quantifiable measures).

Three Planning Horizons

Through structured conversations, the product options deliberately evolve, often transitioning through three planning horizons: the Big-View, the Pre-View, and the Now-View. The goal is to discover the candidate solution. On an agile project, the candidate solution is the smallest possible set of options that delivers value. The smallest set may be defined as a Minimum Viable Product, a feature, a Minimum Marketable Feature, chunky or sliced stories, and others, depending on the planning horizon.

In terms of scope, the Big-View encompasses the generalized product options the partners anticipate will help realize the long-term product vision. In the Pre-View the partners allocate options needed for delivery in the next release. And in the Now-View the product options required for immediate development and potential delivery are thinly sliced and fine-grained. In each view, all three partners realms (customer, business and technology) participate to plan and analyze the product needs, addressing all 7 Product Dimensions.

Conclusion

There is much more to agile requirements than user stories. User stories are just one of many tools that the product partners can use to explore and evaluate product options. Other helpful aids include analysis models, use cases, prototypes, sketches, examples, and tests. Regardless of how product options are represented, successful products result when people collaborate powerfully using structured conversations to continually discover and deliver high-value product needs. This is the essence of agile product planning and analysis.

Ellen Gottesdiener, Founder and Principal of EBG Consulting, helps people discover and deliver the right software products at the right time. Ellen focuses her client around the collaborative convergence of requirements, product, and project management. She presents globally, and provides leadership for the IREB®, IIBA® and PMI® communities. Ellen is co-author of Discover to Deliver: Agile Product Planning and Analysis and author of two other acclaimed books: Requirements by Collaboration and The Software Requirements Memory Jogger.

Mary Gorman, a leader in business analysis and requirements, is Vice President of Quality & Delivery at EBG Consulting. Mary coaches product teams, facilitates discovery workshops, and trains stakeholders in collaborative practices essential for defining high-value products. She speaks at conferences, writes, and plays a leadership role in developing guidelines for requirements and business analysis for IIBA® and PMI® communities. Mary is co-author of Discover to Deliver: Agile Product Planning and Analysis.

Nun ist Herrn Büchle vieles klarer: „Das heißt also, dass in einem agilen Umfeld ohne ganz viel Kommunikation gar nicht entwickelt werden kann, richtig?“ Ramona lächelt. „Eigentlich sollte in jedem Projekt, egal nach welchem Vorgehensmodell es durchgeführt wird, viel kommuniziert werden. Aber manchmal glaubt man, dass Prozesse und definierte Artefakte die Kommunikation ersetzen ...“.

Index

A

- Abstimmungs- und Weisungsmethode 359
- Adaptive Software Development 39, 56
- Agiles Manifest 56
- Agilität
 - agiles RE/agile Entwicklung 508
 - Crystal 39, 57
 - eXtreme Programming 34
 - Kanban 57
 - klassisches RE/agile Entwicklung 506
 - Scrum 39, 57
 - Umstieg auf agile Vorgehen 505
- Aktivitätsdiagramm 195, 532
- Akzeptanzkriterium 59, 67, 275
- Analyse 45
 - siehe* Systemanalyse
- Analysemodell 326
- Änderbarkeit (Qualitätsanforderung) 281
- Anforderung
 - Änderungsrate 35
 - Architektur, Zusammenhang 36
 - Bedingung 240
 - Definition 13
 - Detaillierungsebenen 38
 - herleiten 46
 - Hierarchie 36
 - Historie 403
 - Import/Export 442
 - Inhalt 225
 - Kategorien 17
 - Lebenszyklus 388
 - nicht-funktional 147, 234, 268
 - Präzisierung 38
 - Qualitätsaspekte 303
 - Quelle 77
 - rechtliche Verbindlichkeit 18
 - Semantik 225
 - strukturieren 432
 - Vereinzelung 142
 - Verfeinerung 36, 423
 - Version 404
 - Zeitpunkt der Erzeugung 47
 - Zusammenhang zwischen 36
 - Zustand 390
 - Zweck 16
- Anforderungen erahnen 118
- Anforderungsermittlung 89
 - Einflussfaktoren 119
 - Erfolgsfaktoren 91
 - Ermittlungstechniken 98
- Anforderungskonsolidierung 347
 - Dokumentation 363
 - Kommunikationsmodell Schulz von Thun 357
 - Konfliktsteckbrief 351
 - Verhaltensstrategien 358
- Anforderungsquelle 77
 - Dokument 77
 - Stakeholder 77, 79
 - System in Betrieb 77
- Anforderungsschablone 215
 - Bedingung 240, 241
 - Definition 218
 - Details 232
 - Eigenschaften 235
 - englisch 230
 - funktional 219
 - Inhalt 225
 - Kontext 237
 - mit Bedingung 220, 224
 - nicht-funktional 234
 - Prozesse 238
 - Semantik 225
 - Syntax 220
 - Überblick 219
- Annäherungsmethode 359
- Anwendungsfall
 - siehe* Use-Case
- Apprenticing 104
- Arbeitsgedächtnis nach Baddeley 21

- Artefaktbasierte Techniken 110
- ASD
 - siehe* Adaptive Software Development
- Attribut 409
 - Kombinationen 410
 - Pflichtattribut 417
 - Vererbung 423
- Attributierungsschema 415
 - Definition 415
- Attributtyp 409
- Attributwert 409
- Audio-/Videoaufzeichnungen 115
- Auffälligkeit in Anforderungen 303
- Ausnahmen in Anforderungen 156
- Auswahlempfehlung 119
- Auswertung
 - Fortschrittsauswertung 419
- Automotive SPICE 55
- B**
- Backlog-Item 59
- Basisfaktoren 95
- Basislinie 455
- Bedienelemente 288
- Bedienkonzepte 287
- Bedingung 229, 240
 - Semantik 240
 - Syntax 240
- Bedingungen in Anforderungen 158
- BedingungsMASTeR 241
- Befragungstechniken 105
- Begeisterungsfaktoren 97
- Begriffsmodell 201
- Bennungskonflikt 355
- Benutzbarkeit (Qualitätsanforderung) 281
- Benutzerfreundlichkeit 524
- Benutzungsoberfläche 286
- Beobachtungstechniken 103
- Betrachtungsgegenstand 36
- Bewusstes Wissen 96
- Beziehungskonflikt 356
- Bootstrap 55
- BPMN
 - siehe* Business Process Modell & Notation
- Brainstorming 99
- Bug 449
- Business Process Modell & Notation 174
- C**
- Capability Maturity Model (CMM) 486
- CCB
 - siehe* Change Control Board
- Certified Professional for Requirements Engineering 11
- Change Control Board 452
- Change-Management 43, 451
 - Definition 446
- Checklisten 329
- Cliffhanger-Effekt 20
- CMMI 55
- ConditionMASTeR 242
- CPRE
 - siehe* Certified Professional for Requirements Engineering
- D**
- Daily-Scrum 60
- Darstellungstransformationen 127
- Defect 450
- Definition
 - Begriff 229
 - Prozesswort 228
- Definition of Done 59, 67, 260, 275
- Definition of Ready 261
- Delta-Anforderung 514
- Delta-Ansatz 513, 521
 - funktionale Anforderungen 517
 - nicht-funktionale Anforderungen 520
- Delta-Spezifikation 513
- Detailed FunctionalMASTeR 234
- Detaillierter FunktionsMASTeR 232
- Detaillierungsebenen 38
- Dokumentation
 - Gründe 19
 - Ziel 84
- Dokumentationstechnik 186
 - Auswahlempfehlung 211
 - Einflussfaktoren 211
- Dreyfus Model of Skill Acquisition 480

E

Eigenschaften 144
 EigenschaftsMASTER 235
 Einführung Requirements Engineering 478
 Lernen 480
 Einführungsprojekt 483
 Einführungsstrategie 478
 Agilität 504
 Coachingkonzept 495
 fachliches Konzept 483
 Konzept zur Wissensvermittlung 492
 Marketingkonzept 491
 Methode 488
 Migrationskonzept 501
 Mitarbeiter 487
 Pilotierungskonzept 498
 Prozess 488
 Schulungskonzept 493
 Umsetzung 490
 Vorbereitung 485
 Werkzeug 489
 Widerstand 479
 Emergency Release 458
 Entwicklungsteam 58
 EnvironmentMASTER 239
 Epic 59, 248
 EPK
 siehe Ereignisgesteuerte Prozessketten
 Ereignisgesteuerte Prozessketten 172
 Ermittlungstechniken
 Anforderungen erahnen 118
 Apprenticing 104
 Artefaktbasierte Techniken 110
 Audio-/Videoaufzeichnungen 115
 Auswahlempfehlung 119
 Befragungstechniken 105
 Beobachtungstechniken 103
 Brainstorming 99
 Essenzbildung 117
 Feldbeobachtung 104
 Fragebogen 106
 Interview 106
 Kreativitätstechniken 99
 Methode 6-3-5 100
 Mind Mapping 114

SOPHIST-REgelwerk 113
 Systemarchäologie 110
 Szenarien 116
 Unterstützende Techniken 112
 Use-Case-Modellierung 116
 Wechsel der Perspektive 101
 Wiederverwendung 111
 Workshop 113
 Ermittlungstechnikenmatrix 119
 Essenzbildung 117

F

Feature Tree 470
 Feldbeobachtung 104
 Filtern 418
 Fragebogen 106
 FunctionalMASTER 230
 Detailed 234
 Funktionale Anforderung
 Definition 17
 Funktionalität 282
 FunktionsMASTER 220, 230
 detailliert 232
 mit Bedingung 220, 224
 Funktionsverbgefüge 140

G

Generalisierung 129
 Geschäftsprozess
 Ablaufdiagramme 172
 Definition 169
 Dokumentation 169
 Geschäftsregeln 177
 Given-When-Then-Schema 251
 Gliederungsstruktur 379
 Glossar 183
 Graphical-User-Interface 286, 524
 GUI
 siehe Graphical-User-Interface

H

HMI 286
 siehe Human-Machine-Interface (HMI)
 Homonym 227
 Human-Machine-Interface 286

I

IEEE 29148 380
Implizite Annahmen 159
Import/Export von Anforderungen 442
Incident-Management 43, 448
Informationsart 377
Informationsmodell 377
Inkrement 59
Innovation
 Definition 450
Innovations-Management 44
Inspektion 320
Interessenkonflikt 355
International Requirements Engineering
 Board 11
Interview 106
IREB
 siehe International Requirements
 Engineering Board
ISO 9000 55
ISO 9001 55
ISO 26262 55
ITIL 43, 55, 446
IVENA XT 465

K

Kano-Modell 94
Kapitelstruktur 379
Klassendiagramm 201
Kommunikationsmodelle 91
 Schulz von Thun 357
 Sender-Empfänger-Modell 91
 Vier-Seiten-Modell 92
Konfiguration 455
Konflikt 348
Konfliktanalyse 350
 Konfliktentwicklung 353
 Konfliktursachen 353
Konfliktarten 354
Konfliktauflösung 357
Konfliktentwicklung
 Fieberkurve der Konfliktentwicklung 354
Konfliktidentifikation 349
Konflikttrisiken 357

Konfliktsteckbrief 351
Konsolidierung 357
Konsolidierungsmatrix 363
Konsolidierungstechniken
 Abstimmung 360
 Abstimmungs- und Weisungsmethoden 360
 Analytische Techniken 361
 Annäherungsmethode 359
 Auswahlempfehlung 363
 Einigung 359
 Entscheidungsmatrix 363
 Hilfstechniken 361
 Kompromiss 359
 Konfigurationsvariante 360
 Ober-sticht-Unter 360
 Unterstützende Techniken 361
 Variantenbildung 360
Kontext
 siehe System
Kontextvisualisierung 181
 Datenflussdiagramm 182
 Kontextdiagramm 182
 Use-Case-Diagramm 182
Kreativitätstechniken 99

L

Lean Software Development 39
Leistungsfaktoren 96
Leitfaden
 Erstellung 497
 Inhalt 496
 Motivation 496
 Qualitätssicherungsleitfaden 306
Lernstufen 481
Lesetechniken 328

M

Man-Machine-Interface 286
MASTeR 218
 BedingungsMASTeR 241
 ConditionMASTeR 242
 Detailed FunctionalMASTeR 234
 Detaillierter FunktionsMASTeR 232
 EigenschaftsMASTeR 235

- EnvironmentMASTER 239
- FunctionalMASTER 230
- FunktionsMASTER 220
- ProcessMASTER 239
- PropertyMASTER 239
- ProzessMASTER 238
- UmgebungsMASTER 237
- Messvorlage
 - Definition 341
- Methode 6-3-5 100
- Metrik
 - Definition 335
- Mind Mapping 114
- MMI
 - siehe* Man-Machine-Interface
- N**
- Nachdokumentation 64
- Nachvollziehbarkeit
 - siehe* Traceability
- Neurolinguistisches Programmieren 124
- Nicht-funktionale Anforderung 268
 - an die Benutzeroberflächen 286
 - an durchzuführende Tätigkeiten 294
 - an Qualität 280
 - an rechtlich-vertragliche Aspekte 295
 - an sonstige Lieferbestandteile 292
 - Definition 268
 - Erhebungsprozess 270
 - Technologische 277
- NLP
 - siehe* Neurolinguistische Programmieren
- Nominalisierung 138
- Nutzungsablauf 532
- Nutzungskontext 528
- O**
- Oberflächenstruktur 127
- Objekt-ID 383
- P**
- PDCA-Zyklus 306
- Persona 525
- Persona-Konzept 525
- Persona-Steckbrief 526
- Pilotprojekt 498
- Plan-Do-Check-Act
 - siehe* PDCA-Zyklus
- Planguage 284
- Portfolio-Management 44
- Priorisierung 415
 - Priorisierungskriterium 415
 - Priorisierungstechnik 415
 - Ticket 451
- Problem-Management 43
- ProcessMASTER 239
- Product-Backlog 58
- Product-Owner 58
- Produktfaktoren
 - Basisfaktoren 95
 - Begeisterungsfaktoren 97
 - Leistungsfaktoren 96
- Produktlinie 469
 - Merkmalbasiert 473
 - Plattform 472
 - Repository-Vorgehen 471
- PropertyMASTER 239
- Prosaanforderung 187
- Prototyp 322
- ProzessMASTER 238
- Prozesswort 136, 227
 - Bedeutung 137, 228
 - Konkretisierung 233
- Prozesswortliste 228
- Prüfen von Anforderungen
 - Auswahl Prüfer 315
 - Fehlerkorrektur 305
 - Vorgehen 305
- Prüftechniken 317
 - Auswahlempfehlung 331
 - Hilfsmittel 328
 - Reviews 318
- Q**
- Qualität
 - Definition 302
- Qualitätsanforderungen 280
 - an die Änderbarkeit 281
 - an die Benutzbarkeit 281

- an die Effizienz 282
- an die Funktionalität 282
- an die Übertragbarkeit 283
- an die Zuverlässigkeit 283
- Qualitätsaspekte
 - Abgestimmtheit 303
 - Dokumentation 303
 - Inhalt 303
- Qualitätskriterien
 - Anforderung 26
 - Anforderungsspezifikation 28
- Qualitätsmetrik
 - Definition 335
- Qualitätssicherung
 - Analytisch 304
 - Konstruktiv 304
- Qualitätsstandard
 - Automotive SPICE 55
 - Bootstrap 55
 - CMMI 55
 - FDA 55
 - ISO 9000 55
 - ISO 9001 55
 - ISO 26262 55
 - ITIL 55
 - SPICE 55
- R**
- Randbedingungen 270
- Rational Unified Process 53, 54
- Rechte und Rollen 399
- Rechtliche Verbindlichkeit 226
- Rechtlich-vertragliche Anforderung 295
- Referenzen
 - siehe* Traceability
- Release-Management 453
- Release- und Deployment-Management
 - Definition 446
- Repräsentationssystem der Sprache 93
 - Augenbewegungsmuster 93
 - Ich-Zustands-Modell 93
 - VAKOG 93
- Requirements-Engineer 11
- Requirements-Engineering
 - Aufwand 34
 - Definition 13
 - Haupttätigkeiten 14
 - Probleme 24
 - Requirements-Engineering-Leitfaden
 - Definition 370
 - Requirements-Management 367
 - Abhängigkeiten 375
 - Ablaufsteuerung 374
 - Arbeitsabläufe 398
 - Aufgaben 372
 - Auswertungen 375
 - Definition 368
 - Grundannahme 369, 370
 - Gründe 369
 - Informationsaustausch 373
 - Metamodell 377
 - Objekt-ID 383
 - Prozess 377
 - Rahmenbedingungen 372
 - Tool 371
 - Review 318
 - Rollenkonflikt 356
 - Rollen- und Rechte-Matrix 402
- S**
- Sachkonflikt 354
- Satir-Modell 479
- Schnittstelle 222, 277
- Scrum 39, 57
 - Aktivitäten 59
 - Artefakte 58
 - Backlog-Item 59
 - Daily-Scrum 60
 - Inkrement 59
 - Product-Backlog 58
 - Rollen 58
 - Sprint-Backlog 59
 - Sprint-Planning 59
 - Sprint-Retrospective 60
 - Sprint-Review 60
 - User-Story 59, 66
- Scrum-Master 58
- Sequenzdiagramm 197

- Sicht 418
 - selektive 418
 - verdichtende 419
 - SOPHIST-REgelwerk 123
 - Anwendung 161
 - Ermittlungstechnik 113
 - Priorisierung 163
 - Prüftechnik 329
 - Regeln 136
 - Vorgehen 133
 - Spezifikation 418
 - Spezifikationslevel 39
 - Spezifikationsstruktur 379
 - SPICE 55
 - Sprachliche Effekte 131
 - Sprint 60
 - Sprint-Backlog 59
 - Sprint-Goal 60
 - Sprint-Planning 62
 - Sprint-Retrospective 60
 - Sprint-Review 60
 - STABLE 433
 - Stakeholder 79
 - Analyse 82
 - Anforderungsquelle 77
 - Stakeholdertabelle 81
 - Stakeholderliste
 - siehe* Stakeholdertabelle
 - Standardgliederung 380
 - Stellungnahme 319
 - Story-Mapping 253
 - Strukturierung 432
 - Aktivitätsdiagramm 438
 - funktionale Anforderung 433
 - nicht-funktionale Anforderung 432
 - STABLE 433
 - Use-Case 436
 - User-Story 434
 - Zustandsdiagramm 442
 - Strukturkonflikt 356
 - Substantiv 151, 228
 - Bedeutung 229
 - Synonym 227
 - System
 - Kontext 85
 - Kontextabgrenzung 85, 86
 - Kontextgrenze 87
 - Systemabgrenzung 86
 - Systemgrenze 87
 - Umfang 85
 - Systemaktivität
 - Benutzerinteraktion 222
 - Definition 221
 - Schnittstellenanforderung 222
 - Selbsttätige 222
 - Systemanalyse
 - Motivation 10
 - Überblick 42
 - Ziel 35
 - Systemanalytiker
 - siehe* Requirements-Engineer
 - Systemanforderung
 - Dokumentation 185
 - Systemanwendungsfall
 - siehe* Use Case
 - Systemarchäologie 110
 - Szenario 530
 - Dokumentationstechnik 187
 - Ermittlungstechnik 116
 - Usability-Engineering 524
- ## T
- Technical Story 254, 275
 - Technologische Anforderungen 277
 - Testfall 323
 - Zustandsdiagramm 396
 - Theme 59, 249
 - Ticket
 - Priorisierung 451
 - Typen 449
 - Tiefenstruktur 127
 - Tilgung 129
 - Trace 426
 - Attribut 427
 - Hyperlink 427
 - Matrix 428
 - Textuell 426
 - Traceability 421
 - Änderungen 453

- Definition 421
- Eltern-Kind-Verbindung 422
- Gleiche Detaillierungsebene 425
- Verfeinerung 423
- Verschiedene Informationsarten 425
- Vorgehen 429
- Ziele 422
- Transformationsgrammatik 125
- Transformationsprozess 125
- Tuning
 - Definition 450
- U**
- UI
 - siehe* User-Interface
- UmgebungsMASTeR 237
- Unbewusstes Wissen 97
- Und-Oder-Baum
 - Zieldokumentation 84
- Unterbewusstes Wissen 95
- Unterstützende Techniken 112
- Usability 524
- Use-Case
 - Analyse 47
 - Beschreibung 171, 192
 - Diagramm 170, 189
 - Modellierung 116
- User-Interface 286
- User-Story 59, 67, 249
 - Akzeptanzkriterium 250
 - Aufbau 249
 - Meta-Pattern 256
 - Minimal-Ansatz 258
 - Reduktions-Ansatz 259
 - Schneidung 256
- V**
- Variantenmanagement 515
- Verb 227
- Verfolgbarkeit
 - siehe* Traceability
- Verfolgbarkeitsmodell 429
- Vergessenskurve nach Ebbinghausen 19
- Verhaltensstrategien 358
- Versionierung 404
- Verzerrung 130
- V-Modell XT 53, 380
- Volere 380
- Vollverb 137
- Vorgehen
 - agil 52, 56
 - iterativ-inkrementell 56
 - konventionell 52
 - leichtgewichtig 34
 - schwergewichtig 34
 - wasserfallartig 53
- Vorgehensmodell
 - ASD 56
 - Crystal 57
 - Kanban 57
 - Rational Unified Process 54
 - Scrum 57
 - V-Modell XT 53
- W**
- Wahrnehmungstransformationen 127
- Walkthrough 319
- Wechsel der Perspektive 101
- Wertekonflikt 355
- Wiederverwendung 111
 - Kandidaten 461
 - Regelgeleitet 462
- Workshop 113
- Z**
- Zeigarnik-Effekt 20
- Ziel 75
 - Dokumentation 84, 180
 - Merkmal 83
 - Zielbaum
 - siehe* Und-Oder-Baum
- Zieldokumentation 84
 - Planguage 84
 - Und-Oder-Baum 84
 - Zielschablone 84
- Zustandsdiagramm 199
- Zustandsmodell, Informationsarten 394