Certified Professional for Requirements Engineering

RE@Agile

Lehrplan

Practitioner | Specialist

Stefan Gärtner, Peter Hruschka,
Markus Meuten, Gareth Rogers,
Hans-Jörg Steffe

2.2.0 | 24. Mai 2024

Nutzungsbedingungen

1. Einzelpersonen und Seminaranbieter dürfen den Lehrplan als Grundlage für Seminare verwenden, sofern die Inhaber der Urheberrechte als Quelle und Besitzer des Urheberrechts anerkannt und benannt werden. Des Weiteren darf der Lehrplan zu Werbezwecken nur mit Einwilligung des IREB e.V. verwendet werden.
2. Jede Einzelperson oder Gruppe von Einzelpersonen darf den Lehrplan als Grundlage für Artikel, Bücher oder andere abgeleitete Veröffentlichungen verwenden, sofern die Autoren und IREB e.V. als Quelle und Besitzer des Urheberrechts genannt werden.

Danksagung

Dieser Lehrplan wurde verfasst von: Lars Baumann, Stefan Gärtner, Peter Hruschka, Kim Lauenroth, Markus Meuten, Sacha Reis, Gareth Rogers und Hans-Jörg Steffe.

Review durch Rainer Grau. Mit Kommentaren von Jan Jaap Cannegieter, Andrea Hermann, Uwe Valentini und Sven van der Zee. Deutsch-Review durch Raik Arndt, Sibylle Becker und Ruth Rossi.

Die Freigabe der englischen Version wurde am 18. Februar 2022 durch das IREB Council auf Empfehlung von Xavier Franch genehmigt.

Allen sei für ihr Engagement gedankt.

Zweck des Dokuments

Dieser Lehrplan definiert die Lernziele und eine Zusammenfassung der Lerninhalte für die RE@Agile Practitioner und Specialist Zertifizierungen des International Requirements Engineering Board (IREB). Der Lehrplan dient den Schulungsanbietern als Grundlage für die Erstellung ihrer Kursunterlagen. Die Lernenden können sich anhand des Lehrplans auf die Prüfung vorbereiten.

Inhalt des Lehrplans

Das Modul RE@Agile der Fortgeschrittenenstufe spricht Fachleute der Berufsbilder *Requirements Engineering, Geschäftsanalyse/Business Analysis, Business Engineering, Organisationsgestaltung* an, welche ihre Kenntnisse und Fähigkeiten im Bereich RE@Agile vertiefen möchten.

Inhaltsabgrenzung

Beim Practitioner und Specialist werden – wie im Foundation Level – Grundlagen des Requirements Engineering vermittelt, die für alle Bereiche, z. B. eingebettete Systeme, sicherheitskritische Systeme, klassische Informationssysteme, gleichermaßen gültig sind. Dies heißt nicht, dass die Eignung von Ansätzen für die einzelnen Bereiche, unter Beachtung deren Besonderheiten, in einer Schulung nicht behandelt werden können. Es ist jedoch nicht das Ziel, spezifisches Requirements Engineering einer bestimmten Domäne darzustellen.

Es wird kein bestimmtes Vorgehens- und damit verbundenes Prozessmodell zugrunde gelegt, das eine Aussage über die Planung, Steuerung und Reihenfolge der Anwendung der erlernten Konzepte in der Praxis macht. Es geht nicht darum, einen bestimmten Prozess für Requirements Engineering oder gar das gesamte Software-Engineering besonders hervorzuheben.

Es wird definiert, was das Wissen von Requirements Engineers ausmacht, nicht jedoch die exakten Schnittstellen zu anderen Disziplinen und Prozessen des Software-Engineering.

Detaillierungsgrad

Der Detaillierungsgrad dieses Lehrplans erlaubt international konsistentes Lehren und Prüfen. Um dieses Ziel zu erreichen, beinhaltet dieser Lehrplan Folgendes:

- Allgemeine Lernziele
- Inhalte mit einer Beschreibung der Lernziele und
- Referenzen zu weiterführender Literatur (falls notwendig).

Lernziele/Kognitive Stufen des Wissens

Allen Modulen und Lernzielen in diesem Lehrplan ist eine kognitive Stufe zugeordnet. Die Stufen sind wie folgt klassifiziert:

- **K1: Kennen** (beschreiben, aufzählen, charakterisieren, erkennen, benennen, erinnern, ...) — Der Kandidat kann sich an zuvor gelernten Stoff erinnern oder ihn abrufen.
- **K2: Verstehen** (erklären, interpretieren, vervollständigen, zusammenfassen, begründen, klassifizieren, vergleichen, ...) – Der Kandidat kann die Bedeutung anhand von gegebenem Inhalt oder Situationen begreifen/erfassen.
- **K3: Anwenden** (spezifizieren, schreiben, entwerfen, entwickeln, implementieren, ...) — Der Kandidat kann Wissen und Fähigkeiten in gegebenen Situationen anwenden.
- **K4: Analysieren** (untersuchen, schlussfolgern, Argumente liefern, ...) – Der Kandidat kann ein gegebenes Problem analysieren, argumentieren, was getan werden sollte/kann, das Problem in Teile zerlegen, kritisches Denken anwenden, bezüglich Ursachen und Wirkungen argumentieren.
- **K5: Beurteilen** (kritisieren, beurteilen) — Der Kandidat kann eine gut begründete Kritik an einem gegebenen Artefakt äußern; ein fundiertes Urteil in einem gegebenen Fall abgeben.

Beachten Sie, dass ein Lernziel auf der kognitiven Wissensstufe Kn auch Elemente aller darunterliegenden kognitiven Wissensstufen (K1 bis Kn-1) enthält.

Beispiel: Ein Lernziel der Art „Die RE-Technik xyz anwenden" ist auf der kognitiven Wissensstufe (K3). Die Fähigkeit zur Anwendung setzt aber voraus, dass die Lernenden die RE Technik xyz kennen (K1) und dass sie verstehen, wozu diese Technik dient (K2).

Alle in diesem Lehrplan verwendeten Begriffe und Begriffe, die im Glossar genannt werden, sind zu kennen (K1), auch wenn sie in den Lernzielen nicht explizit genannt sind.

Das Glossar steht auf der IREB Homepage zum Download zur Verfügung https://www.ireb.org/de/downloads/#cpre-glossary-2-0.

Im Lehrplan sowie im dazugehörigen Handbuch wird die Abkürzung RE für Requirements Engineering verwendet.

Lehrplanaufbau

Der Lehrplan besteht aus **sechs Kapiteln**. Ein Kapitel umfasst eine Lerneinheit (LE). Für jede LE werden Unterrichts- und Übungszeiten angegeben. Das ist das Minimum, das in einem Kurs für die jeweilige LE aufgewendet werden sollte. Schulungsunternehmen steht es frei, mehr Zeit für die LEs und Übungen zu investieren. Sie sollten jedoch sicherstellen, dass der Zeitaufwand im Verhältnis zu den übrigen LEs beibehalten wird. Die für ein Kapitel wichtigen Begriffe werden zu Beginn jedes Kapitels aufgelistet. Teilweise sind sie im Glossar definiert oder sie werden im Kapitel erläutert.

Beispiel:
Kapitel 2: Projekte erfolgreich starten (K2)
Dauer: 120 Minuten + 60 Minuten Übungszeit
Begriffe: Produktvision, Produktziel, Stakeholder, Persona, Produktumfang, Systemgrenze

Das Beispiel zeigt, dass in Kapitel 2 die Lernziele der Stufe K2 enthalten sind und 180 Minuten für das Lehren des Materials in diesem Kapitel vorgesehen sind.

Jedes Kapitel kann Unterkapitel enthalten. In deren Titel findet sich ebenfalls die kognitive Stufe der betroffenen Teilinhalte.

Vor dem eigentlichen Text sind die Lernziele (LZ) gelistet. Die Nummerierung zeigt die Zugehörigkeit zu Unterkapiteln an.

Beispiel: LZ 3.3.1 Die Konzepte des INVEST-User-Storytelling können und anwenden

Dieses Beispiel verdeutlicht, dass das Lernziel LZ 3.3.1 im Unterkapitel 3.3 beschrieben wird.

Die Prüfung

Dieser Lehrplan umfasst Lerneinheiten und Lernziele für die Zertifizierungsprüfungen zum

- RE@Agile Practitioner
- RE@Agile Specialist

Die Prüfung zum Erlangen des RE@Agile Practitioner Zertifikats besteht aus einer **Multiple-Choice-Prüfung**.

Die Prüfung zum Erlangen des RE@Agile Specialist Zertifikats besteht aus einer **schriftlichen Ausarbeitung**.

Beide Prüfungen umfassen Prüfungsfragen zu allen Lerneinheiten und allen Lernzielen des Lehrplans.

Jede Prüfungsfrage kann Stoff aus mehreren Kapiteln des Lehrplans sowie mehreren Lernzielen oder auch von Teilen eines Lernziels beinhalten.

Die **Multiple-Choice-Prüfung** für das **Practitioner** Zertifikat

- prüft alle Lernziele des Lehrplans. Bei den Lernzielen der kognitiven Wissensstufen K4 und K5 beschränken sich die Prüfungsfragen jedoch auf Elemente auf den kognitiven Stufen K1 bis K3.
- kann unmittelbar im Anschluss an einen Kurs aber auch unabhängig davon (z. B. remote oder in einem Prüfzentrum) abgelegt werden.

Die **schriftliche Ausarbeitung** für das **Specialist** Zertifikat

- prüft alle Lernziele des Lehrplans auf den für die jeweiligen Lernziele angegebenen kognitiven Wissensstufen.
- folgt der Aufgabenbeschreibung zum RE@Agile Specialist, zu finden unter https://www.ireb.org/de/downloads/tag:advanced-level-written-assignment#top.
- erfolgt in Eigenregie und wird bei einer lizenzierten Zertifizierungsstelle eingereicht.

Für die **schriftliche Ausarbeitung** für das **Specialist** Zertifikat gelten zudem die folgenden generischen Lernziele:

LZ G1: Analysieren und illustrieren von RE@Agile -Problemen in einem Kontext, mit dem die Lernenden vertraut sind, oder der einem solchen Kontext ähnlich ist (K4).

LZ G2: Evaluieren und reflektieren der Anwendung von RE@Agile -Praktiken, Methoden, Prozessen und Werkzeugen in Projekten, an denen die Lernenden beteiligt waren (K5).

Eine Liste der von IREB lizenzierten Zertifizierungsstellen finden Sie auf der Website https://www.ireb.org.

Versions-Historie

Version	Datum	Kommentar	Autor
1.0.0	10. September 2018	Erste Version	Bernd Aschauer, Lars Baumann, Peter Hruschka, Kim Lauenroth, Markus Meuten, Sacha Reis and Gareth Rogers
1.0.1	10. September 2018	Rechtschreibung korrigiert Einige LE's neu formuliert, um dem Standard zu entsprechen. Inhaltlich keine Änderung. Danksagung hinzugefügt	Stefan Sturm
1.0.2	17. Dezember 2019	Umbenennung des Begriffs „Verfeinerungssitzung" durch den englischen Begriff "Refinement Meeting".	Hans-Jörg Steffe
2.0.0	1. Juli 2022	Lernziele aktualisiert Neufassung von Kapitel 6 Korrekturen in allen Kapiteln Überarbeitung der vorgeschlagenen Unterrichts- und Übungszeiten für alle Kapitel. Information über Advanced Level Prüfungssplit hinzugefügt.	Peter Hruschka, Markus Meuten, Gareth Rogers, Stefan Gärtner, Hans-Jörg Steffe
2.1.0	1. Mai 2024	Neues Corporate Design implementiert, kognitive Stufen des Wissens synchronisiert, Begriff Advanced Level eliminiert, "Voraussetzungen" für das Modul RE@Agile entfernt.	Stan Bühne, Ruth Rossi
2.2.0	24. Mai 2024	Kognitive Stufen des Wissens erneut gefixt.	Stefan Sturm

Inhalt

Informationen zum Modul RE@Agile

Das Modul RE@Agile richtet sich an Requirements Engineers und Experten für agile Entwicklungsprozesse. Der Schwerpunkt liegt auf dem Verständnis und der Anwendung von Verfahren und Techniken aus der Disziplin des Requirements Engineering in agilen Entwicklungsprozessen sowie auf dem Verständnis und der Anwendung von Konzepten, Techniken und essenziellen Prozesselementen agiler Ansätze in Requirements-Engineering-Prozessen. Die Zertifizierung versetzt Personen mit Requirements-Engineering-Kenntnissen in die Lage, in agilen Umgebungen zu arbeiten; Experten für agile Entwicklungsprozesse erlaubt sie, bewährte Requirements-Engineering-Verfahren und -Techniken in agilen Projekten anzuwenden.

1 Was ist RE@Agile (K2)

Dauer: 45 Minuten

Begriffe: Kooperation von Stakeholdern, iteratives Requirements Engineering, inkrementelles Requirements Engineering, Product Owner

Lernziele

LZ 1.1 Die Definition von RE@Agile kennen (K1)
LZ 1.2 Die Ziele von RE@Agile verstehen (K2)
LZ 1.3 Erkennen, dass die Verantwortung für gute Anforderungen in SCRUM beim Product Owner liegt (K1)

IREB definiert RE@Agile [IREB2017] als kooperativen, iterativen und inkrementellen Ansatz mit den folgenden vier Zielen:

1. Die relevanten Anforderungen in einem angemessenen Detaillierungsgrad zu kennen (zu jedem Zeitpunkt während der Systementwicklung)
2. Eine ausreichende Einigung der relevanten Stakeholder über die Anforderungen erzielen
3. Die Anforderungen gemäß den Rahmenbedingungen der Organisation zu erfassen (und zu dokumentieren)
4. Alle, auf Anforderungen bezogenen Aktivitäten gemäß den Prinzipien des Agilen Manifests durchführen

Dieser Definition liegen die folgenden zentralen Ideen zugrunde:

- RE@Agile ist ein kooperativer Ansatz:
  „Kooperativ" betont die Idee der Agilität einer intensiven Interaktion von Stakeholdern, indem das Produkt häufig geprüft, Feedback dazu gegeben und die Anforderungen im Projektverlauf geklärt und angepasst werden, wenn alle dazugelernt haben [AgileManifesto2001].
- RE@Agile ist ein iterativer Prozess:
  Daraus lässt sich die Idee der „Just in Time"-Anforderungen ableiten. Es müssen noch nicht alle Anforderungen vor Beginn der Arbeit am technischen Design und an der Implementierung vollständig sein. Stakeholder können diese iterativ im erforderlichen Detaillierungsgrad definieren und verfeinern, wenn die zu implementierenden Anforderungen zeitnah umgesetzt werden sollen [LaBai2003].
- RE@Agile ist ein inkrementeller Prozess:
  Im ersten Inkrement sollten die Anforderungen implementiert werden, die den größten Geschäftswert liefern oder die signifikantesten Risiken mindern. In den ersten Inkrementen wird die Erstellung eines Minimum Viable Product (MVP) oder eines Minimum Marketable Product (MMP) angestrebt. Anschließend wird das Produkt durch weitere Inkremente ergänzt, wobei diejenigen Anforderungen priorisiert werden, die den nächstgrößten Geschäftswert versprechen und dadurch den Gesamtwert des Ergebnisses am besten steigern [Reinertsen2009].

Das erste Ziel („relevante Anforderungen, die in angemessenem Detaillierungsgrad bekannt sind") bezieht sich auch auf den iterativen Ansatz: Als „relevant" gelten die Anforderungen, die zeitnah implementiert werden sollen. Diese Anforderungen müssen sehr präzise verstanden worden sein (einschließlich der zugehörigen Akzeptanzkriterien) – insbesondere auf Seiten der Entwickler. Sie müssen der „Definition of Ready" (DoR) genügen. Andere Anforderungen, die noch nicht von oberster Priorität sind, können zunächst auf einer allgemeineren Abstraktionsebene gehalten und erst dann detaillierter betrachtet werden, wenn sie wichtiger werden.

Voraussetzung für das zweite Ziel („ausreichende Einigung der relevanten Stakeholder") ist es, alle Stakeholder und deren Relevanz für das zu entwickelnde System zu kennen. Als für die Anforderungen verantwortliche Person (gewöhnlich der Product Owner), müssen Sie die Anforderungen mit den relevanten Stakeholdern verhandeln und daraus die Implementierungsreihenfolge ableiten.

Bei agilen Herangehensweisen wird der intensiven und laufenden Kommunikation über Anforderungen ein größerer Wert beigemessen als einer umfangreichen Dokumentation. Dennoch wird im dritten Ziel betont, wie wichtig Dokumentation in einem geeigneten Detaillierungsgrad ist (welcher von der jeweiligen Situation abhängt). Wenn eine Organisation die Dokumentation von Anforderungen aufbewahren muss (aus rechtlichen Gründen, zur Verfolgbarkeit, für Wartungszwecke usw.), muss auch bei agilen Ansätzen sichergestellt werden, dass diese Art der Dokumentation erstellt wird. Diese sollte jedoch nicht vorab erstellt werden. Es kann Zeit und Aufwand sparen, die notwendige Dokumentation parallel zur Implementierung oder sogar nach der Implementierung zu erstellen.

Das Anforderungsmanagement fasst sämtliche auszuführenden Aktivitäten zusammen, wenn die Anforderungen sowie anforderungsbezogene Artefakte einmal vorliegen. Dazu zählen Versions- und Konfigurationsmanagement ebenso wie die Nachverfolgbarkeit der Anforderungen sowie anderer Entwicklungsartefakte. Für ein ausgewogenes Kosten/Nutzen-Verhältnis empfiehlt RE@Agile ein sorgfältiges Management dieser Aktivitäten, um den damit verbundenen Aufwand im Rahmen zu halten. Beispiele sind:

- Umfangreiches Versionsmanagement lässt sich gegebenenfalls durch schnelle Iterationen zur Erstellung von Produktinkrementen ersetzen (d.h. der Änderungsverlauf von Anforderungen seit ihrem ersten Auftreten ist von geringerem Interesse als ihr aktuell gültiger Status).
- Die iterative Ableitung von Sprint Backlogs umfasst das Konfigurationsmanagement (Baselining), d.h. das Gruppieren der Anforderungen, die aktuell den höchsten Geschäftswert versprechen.

Aus diesem Grund ersetzt man einige der zeitintensiven (und auch papierintensiven) Aktivitäten des Anforderungsmanagements nicht-agiler Ansätze durch Aktivitäten, die auf agilen Prinzipien beruhen.

Mit Scrum wurde die Rolle des Product Owner eingeführt, wie im RE@Agile Primer [IREB2017] bereits erläutert. Dieser Product Owner trägt die Verantwortung für ein gutes Requirements Engineering in einer agilen Umgebung, wenngleich andere Stakeholder (wie Business-Analysten, Requirements Engineers und Entwickler) ihn bei diesem Prozess

unterstützen und unter Umständen auch den größten Teil der Arbeit in Bezug auf das Requirements Engineering erledigen.

Der Kürze halber nennen wir in diesem Lehrplan lediglich den Product Owner (als verantwortliche Person), wenn es um die Durchführung von anforderungsbezogenen Aufgaben geht. Dadurch möchten wir aber in keiner Weise, die von den zuvor genannten Stakeholdern ausgeführte Arbeit ausschließen.

In den folgenden Kapiteln dieses Lehrplans gehen wir näher auf die verschiedenen Aspekte von RE@Agile ein.

In Kapitel 2 behandeln wir die Voraussetzungen für eine erfolgreiche Systementwicklung: das Abwägen von Visionen/Zielen, Stakeholdern und des Systemumfangs.

In den Kapiteln 3 und 4 gehen wir auf den Umgang mit funktionalen Anforderungen, Qualitätsanforderungen und Rahmenbedingungen auf verschiedenen Granularitätsstufen ein.

In Kapitel 5 besprechen wir den Prozess des Schätzens, Ordnens und Priorisierens von Anforderungen zur Festlegung der Reihenfolge der Inkremente.

In den Kapiteln 2 bis 5 liegt der Schwerpunkt auf dem Umgang mit den Anforderungen eines einzigen Teams an Entwicklern (von 3 bis 9 Personen).

In Kapitel 6 wird die Skalierung des Requirements Engineering für größere, potenziell verteilte Teams dargestellt, einschließlich der übergeordneten Produktplanung und Roadmap-Erstellung.

2 Projekte erfolgreich starten (K3)

Dauer: 120 Minuten + 60 Minuten Übungszeit

Begriffe: Produktvision, Produktziel, Stakeholder, Persona, Produktumfang, Systemgrenze

Selbst bei agilen Herangehensweisen müssen einige wichtige Voraussetzungen geschaffen werden, bevor mit einer erfolgreichen iterativen und inkrementellen Systementwicklung begonnen werden kann.

2.1 Spezifikation von Vision und Ziel (K3)

Dauer: 30 Minuten + 15 Minuten Übungszeit

Lernziele

LZ 2.1.1 Vision und Ziel spezifizieren können und diese Spezifikation anwenden (K3)

Anhand der System- oder Produktvision wird das Gesamtziel beschrieben, das mit dem System/Produkt erreicht werden soll. In der agilen Literatur ist häufig von Produktvision anstelle von Systemvision die Rede. Die beiden Begriffe (Produkt- oder Systemvision) sind austauschbar und werden je nach der speziellen Domäne oder dem Kontext der Entwicklungsaktivität verwendet. Die Vision ist für jede Entwicklungsaktivität von höchster Bedeutung. Sie legt den Grundstein und dient als allgemeine Richtschnur für sämtliche Entwicklungsaktivitäten. Jede Anforderung sollte zum Erreichen der Systemvision beitragen [Scrumguide].

Der Unterschied zwischen einem Ziel und einer Anforderung lässt sich folgendermaßen definieren [Glinz2014]:

- Ein Ziel ist eine Aussage über einen gewünschten Sachverhalt (den ein Stakeholder erreichen möchte).
- Eine Anforderung ist eine Aussage über eine gewünschte Eigenschaft des Systems.

In bestimmten Domänen wird der Begriff Problem (das durch ein bestimmtes System gelöst werden soll) anstelle von Ziel verwendet. Aus RE-Perspektive stehen diese Begriffe in unmittelbarem Zusammenhang: Ein Problem ist eine Aussage über eine vorhandene und unerwünschte Eigenschaft, während durch ein Ziel der gewünschte zukünftige Zustand ausgedrückt wird.

Es gibt folgende alternative Ansätze zur Formulierung von Zielen oder Visionen:

- Die Formulierung von SMARTen Zielen [Doran1981]. SMART ist ein Akronym, das für „Specific" (spezifisch), „Measurable" (messbar), „Achievable" (erreichbar), „Relevant" (relevant) und „Time-bound" (terminiert) steht.
- Die Formulierung von Zielen nach PAM [Robertson2003]:
  - Was ist der Zweck (Purpose – P)?
  - Was ist der betriebliche Vorteil/Mehrwert (Business Advantage – A)?
  - Wie würde man diesen Vorteil messen ("M – Measure")?

- Product Vision Box/Design the Box [Highsmith2001]. Nach diesem Ansatz wird ein Paket für das Produkt erstellt, das die zentralen Vorteile/Ideen des Produkts für potenzielle Kunden präsentiert.
- Nachrichten aus der Zukunft [HeHe2011]: Verfassen Sie einen kurzen Zeitungsartikel, der am Tag nach einer erfolgreichen Produkteinführung veröffentlicht wird. Darin beschreiben Sie Gründe dafür, weshalb das Produkt so großartig ist.
- Vision Boards: eine grafische Collage mit Zielen, die ggf. nach kurz-, mittel- und langfristigen Visionen sortiert sind. Einen formaleren Ansatz für Vision Boards finden Sie unter [Pichler2011].
- Mit Canvas-Techniken [OsPi2011] wie der Business Model Generation Canvas oder der Canvas für Chancen (und viele weitere ähnliche Canvas-Arten) bekommen Sie Techniken an die Hand, mit denen Sie fokussierte und leicht verständliche visuelle Darstellungen erstellen können.

Für welche Art und Weise man sich auch entscheidet, jeder Stakeholder hat das Recht zu wissen, was das Team anstrebt. Die Definition der Vision und der anfänglichen Ziele muss daher zu Beginn eines Entwicklungsprojekts erfolgen.

Änderungen der Vision oder von Zielen sind selten, aber nicht unmöglich. Diese können beispielsweise auftreten, wenn neue Stakeholder hinzukommen oder die Arbeit am System ein fundamental neues Verständnis des Systems oder dessen Kontext zutage bringt. Änderungen der Vision oder von Zielen haben wahrscheinlich eine erhebliche Auswirkung auf die Entwicklung und sollten daher mit Vorsicht behandelt werden.

Es besteht die Option, mit unterschiedlichen Visionsdokumenten zu arbeiten, die jeweils einen unterschiedlichen Zeithorizont in der Zukunft abdecken. Zu jedem Zeitpunkt sollte eine gültige, präzise Sechsmonats-, Ein- oder Zweijahresvision, die mit allen relevanten Stakeholdern abgestimmt wurde, den Entwicklern zur Verfügung gestellt werden. Diese Visionsdokumente werden während des Zyklus regelmäßig überprüft und aktualisiert.

Es kommt auch nicht selten vor, dass man sich auf die Realisierung einer bestimmten Teilmenge von Zielen für einen bestimmten Zeitraum (z.B. sechs Monate) fokussiert und anschließend die Ziele ändert, um eine andere Richtung zu verfolgen. Werden Visionen oder Ziele jedoch zu häufig geändert, ist das ein Zeichen dafür, dass es keine klare Richtung zur Steuerung der Produktentwicklung gibt [Reinertsen2009].

2.2 Festlegen der Systemgrenze (K3)

Dauer: 30 Minuten + 15 Minuten Übungszeit

Lernziele

LZ 2.2.1 Systemgrenzen festlegen können, um den Systemumfang vom Kontext abzugrenzen (K3)

Ein gemeinsames Verständnis des Systemumfangs (gemäß IREB: „die Bandbreite von Aspekten, die bei der Entwicklung eines Systems geformt und konzipiert werden kann") und des Kontexts (gemäß IREB „der Teil der Systemumgebung, der für das Verständnis des

Systems und dessen Anforderungen relevant ist") eines Systems sind Voraussetzung für eine effektive und effiziente Entwicklung [Glinz2014].

Missverständnisse in Bezug auf die Systemgrenze können zu Folgendem führen:

- Entwicklung von Funktionalitäten oder Komponenten, die nicht in der Verantwortung der Entwicklung lagen (der angenommene Systemumfang war zu groß)
- Falsche Annahme, dass Funktionalitäten oder Komponenten, die tatsächlich Bestandteil des Systems sind, außerhalb des Systems entwickelt werden sollten (der angenommene Systemumfang war zu klein)

Die Definition des Systemumfangs und des Kontexts gehen mit der Definition der Systemgrenze und der Kontextgrenze einher. Die System- und die Kontextgrenze lassen sich durch die Diskussion folgender Aspekte definieren:

- Welche Features oder Funktionalitäten muss das System und welche der Kontext bereitstellen? (Definition der Systemgrenze)
- Welche technischen Schnittstellen oder Benutzeroberflächen muss das System dem Kontext bereitstellen? (Definition der Systemgrenze)
- Was ist für das System irrelevant, d.h. was beeinflusst den Systemumfang nicht in irgendeiner Weise?
- Welche Aspekte des Systemkontexts können während der Systementwicklung geändert werden? (Definition des Systemumfangs)

Beachten Sie, dass die Kontextgrenze immer unvollständig ist, da die Kontextgrenze nur durch die Faktoren definiert werden kann, die man explizit vom Systemkontext ausschließt. Weitere Aspekte des Systemkontexts schließen insbesondere Stakeholder mit ein. Die Identifizierung und das Management von Stakeholdern ist in 2.3 beschrieben.

Die Systemgrenze lässt sich mit Hilfe verschiedener Techniken dokumentieren und klären, wie etwa:

- Ein Kontextdiagramm: dokumentiert das System, Bestandteile des Kontexts und deren Beziehung
- Natürliche Sprache, d.h. eine Liste von Features, Funktionalitäten, Aspekte des Kontexts, Aspekte des Umfangs und weitere Aspekte außerhalb des Kontexts
- Ein Use Case Diagramm: ein UML-Diagrammtyp, der die Akteure und Use Cases eines Systems modelliert. Es beschreibt den Kontext/Systemumfang auf einer detaillierten Ebene und ist besonders zur Klärung von Systemumfang und Kontext nützlich.
- Eine Story-Map: eine zweidimensionale Anordnung von User Storys in einem zweckdienlichen Modell, anhand dessen die Funktionalität des Systems leichter verständlich wird. Es beschreibt den Kontext/Systemumfang auf einer detaillierten Ebene und ist besonders hilfreich zur Klärung des Systemumfangs.

Die Definition eines initialen Systemumfangs muss zu Beginn eines Entwicklungsprojekts erfolgen. Systemumfang und Kontext verändern sich im Verlauf einer Entwicklung zwangsläufig aufgrund eines neuen oder veränderten Verständnisses des Systems oder des Kontexts. Daher sollte die Dokumentation des Systemumfangs und des Kontexts regelmäßig aktualisiert werden.

2.3 Identifizierung und Management von Stakeholdern (K3)

Dauer: 30 Minuten + 15 Minuten Übungszeit

Lernziele

LZ 2.3.1 Identifizierung und Management von Stakeholdern anwenden (K3)

Es kann große Auswirkungen haben, wenn man versäumt, einen wichtigen Stakeholder zu identifizieren und in die Entwicklung einzubeziehen: Werden die Anforderungen eines solchen Stakeholders zu spät (oder gar nicht) erkannt, hat dies unter Umständen kostspielige Änderungen oder gar ein unbrauchbares System zur Folge [PoRu2015].

Anhand des Zwiebelschalenmodells („Onion Model") von Ian Alexander [Alexander2005] lassen sich Stakeholder einfach identifizieren und klassifizieren. Das Modell besteht aus drei Arten von Stakeholdern: Stakeholder des Systems, Stakeholder des umgebenden Kontexts und Stakeholder aus dem weiteren Umfeld. Stakeholder des Systems bezeichnet man auch als direkte Stakeholder („direct stakeholders"). Stakeholder des umgebenden Kontexts und aus dem weiteren Umfeld werden auch als indirekte Stakeholder („indirect stakeholders") bezeichnet.

Wenn ein System menschliche Benutzer hat, ist der Benutzer einer der wichtigsten direkten Stakeholder.

Die Benutzer eines Systems decken in der Regel ein weites Spektrum von Personen mit unterschiedlichen Erwartungen, Haltungen und Voraussetzungen ab. Ein wichtiger erster Schritt ist es, ein Verständnis des Spektrums von Benutzern für ein bestimmtes System zu entwickeln. Ist die Anzahl von Benutzern gering, dann ist es ratsam, diese (oder deren Vertreter) durch persönliche Interviews kennenzulernen. Ist die Anzahl von Benutzern groß oder gar unbekannt, dann sollte das Benutzerspektrum mit anderen Mitteln erfasst werden, z.B. mit Personas [Cooper2004], die exemplarische Benutzer mit extremen Eigenschaften darstellen. Im Zeitalter von Data Analytics, Google Analytics und Big Data kann das Online-Benutzerverhalten für bereitgestellte Produktinkremente oftmals mit automatisierten Werkzeugen direkt analysiert werden.

Indirekte Stakeholder sind im umgebenden Kontext des Systems zu finden und können für eine Entwicklung ebenso wichtig sein, wie die Benutzer selbst. Das sind etwa Gesetzgeber, Regierungs- oder Normungsinstitutionen, Auditoren für die Verifizierung der Einhaltung von Bestimmungen in regulierten Märkten (Medizin, Transport, Luftfahrt), oder wenn man das Spektrum noch weiter fasst, nichtstaatliche Organisationen, Gewerkschaften oder Wettbewerber.

Die systematische Identifizierung von zentralen Stakeholdern sollte zu Beginn einer Entwicklung stattfinden und die Ergebnisse sollten im Rahmen einer kontinuierlichen Aktivität verwaltet werden. Eine einfache Auflistung mit Kontaktdaten und relevanten Attributen genügt in den meisten Kontexten. Diese Liste kann sich jederzeit ändern, entweder weil ein Stakeholder zunächst übersehen wurde oder weil Änderungen im Kontext aufgetreten sind, wie etwa die Etablierung einer neuen nichtstaatlichen Organisation.

Sämtliche Beteiligte an einer Entwicklung (z.B. Entwickler und Product Owner) sollten sich der Bedeutung von Stakeholdern bewusst sein und nach Anzeichen für neue oder fehlende Stakeholder suchen.

Abhängig von dem jeweiligen System und der Domäne, können auch bereits vorhandene Dokumentation und Altsysteme eine wichtige Quelle für Anforderungen darstellen. Diese sollten ähnlich wie Stakeholder systematisch identifiziert und verwaltet werden, siehe auch [IREB2019].

2.4 Visionen und Ziele, Stakeholder und Systemumfang im Gleichgewicht halten (K3)

Dauer: 30 Minuten + 15 Minuten Übungszeit

Lernziele

LZ 2.4.1 Abwägung von Visionen und Zielen, Stakeholder und Systemumfang anwenden können (K3)

Die Definitionen von Visionen und Zielen sowie Stakeholdern und Systemgrenzen sind voneinander abhängig [PoRu2015]:

- Die relevanten Stakeholder formulieren die Vision und die Ziele. Aus diesem Grund kann sich die Identifizierung eines neuen relevanten Stakeholders auf die Vision oder die Ziele auswirken.
- Mit Visionen und Zielen lässt sich die Identifizierung von neuen Stakeholdern lenken, indem man die Frage stellt: Welcher Stakeholder könnte am Erreichen der Vision/der Ziele Interesse haben oder ist vom Erreichen der Vision/der Ziele betroffen?
- Visionen und Ziele können zum Definieren eines initialen Systemumfangs verwendet werden. Dazu muss man folgende Frage beantworten: Welche Elemente sind zum Erreichen der Vision/der Ziele nötig?
- Eine Änderung der Systemgrenze (und dadurch des Systemumfangs) kann sich auf die Vision/die Ziele auswirken. Wird ein Aspekt aus dem Systemumfang entfernt, muss verifiziert werden, dass das System immer noch zum Erreichen der Vision/der Ziele ausreicht.
- Stakeholder definieren die Systemgrenze. Deshalb kann sich die Identifizierung eines neuen relevanten Stakeholders auf den Systemumfang auswirken.
- Für Änderungen am Systemumfang (z.B. zur Erfüllung eines Ziels) ist die Zustimmung der relevanten Stakeholder erforderlich.

Diese wechselseitigen Abhängigkeiten sollten genutzt werden, um alle drei Elemente im Gleichgewicht zu halten und die Auswirkungen der Veränderung eines der drei Elemente auf die anderen zu untersuchen.

Aufgrund dieser engen Abhängigkeiten zwischen Vision und Zielen, Stakeholdern und Systemumfang empfehlen wir, diese Elemente zusammen und kohärent zu behandeln.

3 Umgang mit funktionalen Anforderungen (K4)

Dauer: 195 Minuten + 120 Minuten Übungszeit

Begriffe: Funktionale Anforderungen, Thema, Epic, Feature, User Story, Akzeptanzkriterien, Aufteilungs- und Gruppierungstechniken, Definition of Ready (DoR), INVEST

In dieser grundlegenden Lerneinheit werden **funktionale Anforderungen** in erster Linie **aus einer statischen Perspektive** betrachtet, d. h. die **Strukturierung eines größeren Anforderungsumfangs** in Abstraktionshierarchien.

3.1 Unterschiedliche Stufen der Anforderungsgranularität (K1)

Dauer: 15 Minuten

Begriffe: Anforderungsgranularität, Hierarchien von Anforderungen

Lernziele

LZ 3.1.1 Wissen, dass funktionale Anforderungen auf verschiedenen Granularitätsstufen existieren (K1)

Stakeholder kommunizieren Anforderungen gewöhnlich in unterschiedlichen Stufen der Granularität. Manchmal wünschen sie, dass große Funktionalitätsblöcke hinzugefügt oder geändert werden, manchmal nur geringfügige Details. Die Erhebung und Strukturierung sämtlicher Anforderungen ist die Aufgabe des Product Owners (mit Unterstützung der anderen Stakeholder) [Scrumguide].

Sowohl grobe als auch detaillierte Anforderungen sind nützlich. Grobe Anforderungen erlauben es dem Product Owner, einen Überblick über alle funktionalen Anforderungen zu behalten. Dies ist vor allem notwendig für langfristiges Planen, das Erstellen von Roadmaps, das Auswählen von Anforderungen, die einen frühzeitigen Geschäftswert versprechen, für allgemeine Einschätzungen und vieles mehr.

Detaillierte Anforderungen sind notwendig, um ein umfassendes Verständnis der Details zu erlangen, welches die Entwickler für die Implementierung dieser Anforderungen benötigen [Patton2014].

Im Kontext von Agilität werden die Begriffe Thema, Epic und Feature verwendet, um die verschiedenen Stufen der Granularität darzustellen, wenngleich noch kein klarer Konsens über deren exakte Reihenfolge besteht. Häufig wird die Hierarchie „Epic – Feature – User Story" bevorzugt, wobei Teile der Hierarchie nach Themen gruppiert werden. Vergleichen Sie dazu auch Abbildung 1 in [IREB2017] im Kapitel 3.1.5 und die Ausführungen im folgenden Kapitel.

3.2 Identifizierung, Dokumentation und Kommunikation von funktionalen Anforderungen (K4)

Dauer: 45 Minuten + 30 Minuten Übungszeit

Begriffe: Wertbasierte und designorientierte Anforderungen, Gruppierungen von Anforderungen, Epic, Thema, Feature, Storys, T-Approach

Lernziele

LZ 3.2.1 Die Dekomposition von Anforderungen auf oberster Ebene analysieren und anwenden können, um die Iterationsplanung und das Erstellen einer Roadmap zu unterstützen (K4)

LZ 3.2.2 Unterschiedliche Dekompositionsstrategien für eine sehr große Anzahl von Anforderungen analysieren und anwenden (K4)

LZ 3.2.3 Identifizierung, Dokumentation und Kommunikation von funktionalen Anforderungen auf unterschiedlichen Granularitätsstufen analysieren und anwenden (K4)

RE@Agile strebt „Just-in-Time-Anforderungen" an [IREB2017]. Um zu ermitteln, welche Anforderungen ausreichend detailliert definiert werden müssen, ist ein Überblick erforderlich.

Die Vision und die Ziele reichen unter Umständen für solche Entscheidungen nicht aus. Daher besteht ein Zwischenziel für einen Product Owner darin, einen Überblick über sämtliche Anforderungen im Systemumfang eines Systems zu erstellen, d.h. die volle Breite abzudecken, ohne zu sehr in die Tiefe zu gehen. Dies wird häufig als T-Approach bezeichnet, da der breite Überblick an die horizontale Linie des Buchstabens T erinnert, während die Detaillierung der Anforderungen, die den größten Geschäftswert versprechen, an die vertikale Linie erinnern.

Zur Zerlegung der Vision oder der Ziele lassen sich unterschiedliche Kriterien heranziehen, um eine Reihe von allgemeinen Anforderungen zu erhalten, die, zusammen betrachtet, den geplanten Systemumfang umfassen. Zahlreiche Methoden legen die Verwendung einer prozessorientierten Zerlegung nahe [GoWo2006] [Lamsweerde2009], d.h. die Aufteilung der Funktionalität in Geschäftsprozesse, Use Cases oder große Storys. Diese Art der Zerlegung erfüllt die ersten drei Kriterien von INVEST, wie im nächsten Kapitel erläutert wird, d.h. die resultierenden Prozesse sind unabhängig, verhandelbar und – was am wichtigsten ist – werthaltig.

Es ist denkbar, alternative Dekompositionsstrategien auf Geschäftsobjekten, der Zerlegung von Teilsystemen eines bereits vorhandenen Systems (d.h. ein versionsbasierter oder designorientierter Ansatz), der Hardware- oder auf der geografischen Verteilung von Teilsystemen aufzubauen.

Die Ergebnisse einer prozessorientierten Zerlegung werden als Use Cases oder User-Storys bezeichnet, die resultierenden Teile der alternativen Zerlegung bezeichnet man häufig als Epics, Themen oder Features.

Ganz gleich, wie man die großen Funktionsblöcke nun bezeichnet, sie stellen eine gute Basis für eine (grobe) Schätzung dar, können bereits geordnet werden und bringen so eine Roadmap oder einen vorläufigen Zeitplan für die Iterationen (oder Sprints – laut Scrum-Terminologie) hervor.

Diese allgemeinen Anforderungen müssen allesamt detaillierter ausgearbeitet werden, bevor die Entwickler sie implementieren können (siehe nächste Kapitel).

Für Kommunikations- und Dokumentationszwecke werden häufig Storys, Epics, Features oder Themen auf physischen Karten erfasst, die im Product Backlog gesammelt werden. Alternativ gibt es auch zahlreiche Werkzeuge zum elektronischen Erfassen von Anforderungen. Sobald allgemeine Anforderungen in einem Set von detaillierteren Anforderungen verfeinert werden, entstehen Anforderungshierarchien. Im Idealfall kann der Product Owner die unterschiedlichen Ebenen verwalten, ohne dass nach der Verfeinerung die Gruppierungen auf höherer Ebene verloren gehen.

Story-Maps [Patton2014] bilden eine Möglichkeit, Karten in zwei Dimensionen anzuordnen und zu visualisieren, sodass Epics und Features auch dann noch sichtbar sind, wenn sie bereits zu User Storys verfeinert wurden.

3.3 Arbeiten mit User-Storys (K3)

Dauer: 30 Minuten + 30 Minuten Übungszeit

Begriffe: User Story, INVEST-Kriterien, 3C-Prinzip, Akzeptanzkriterien

Lernziele

LZ 3.3.1 Die Konzepte des INVEST User Story Telling anwenden (K3)
LZ 3.3.2 Die Erweiterung oder Abstimmung von Themen, Epics, Features und User-Storys mit zusätzlichen Anforderungsartefakten zur Verbesserung der Kommunikation mit Stakeholdern anwenden (K3)

User-Storys sind ein gängiger Ansatz zur Strukturierung von Anforderungen. [Cohn2010] empfiehlt eine Formel zum Erfassen von User-Storys. Die drei Bestandteile dieser Formel stellen sicher, dass

1. wir jemanden haben, der diese Funktionalität haben möchte („Als Anwender …"),
2. wir wissen, was der Anwender möchte („… möchte ich …") und
3. wir den Grund oder die Motivation dahinter nachvollziehen können („… damit …").

Die Formel hilft dabei, darüber nachzudenken, wer was und weshalb haben möchte. Es ist jedoch weniger der Formalismus, der User-Storys erfolgreich macht, sondern das Stellen und Beantworten dieser drei Fragen.

Bill Wake entwickelte „INVEST" [Wake2003] als schönes Akronym für die Beschreibung der Merkmale einer User Story. Die sechs Buchstaben haben folgende Bedeutung:

- **I**: User-Storys sollten Independent – also unabhängig – sein. Das heißt, es sollte möglich sein, User-Storys in einer beliebigen Reihenfolge mit minimalen wechselseitigen Abhängigkeiten zu implementieren.
- **N**: User-Storys sollten nicht als schriftlicher Vertrag, sondern mehr als eine Zusage künftiger Negotiations – Verhandlungen – oder eine Erinnerung an vergangene Diskussionen betrachtet werden. Experten sind in der Rolle eines Requirements Engineer geschulte Vermittler dieser Verhandlung zwischen Geschäftsexperten und Entwicklern, um die Bedeutung der User Story zu erschließen.

- **V**: Jede User Story sollte einen Value – Wert – schaffen. Weitere Details über den Geschäftswert folgen in 5.
- **E**: Eine User Story sollte Estimable – schätzbar – sein. Wenn der für die Implementierung erforderliche Aufwand nicht geschätzt werden kann, ist die Story noch nicht klar genug. Weitere Details über die Schätzung folgen in 5.
- **S**: User-Storys sollten (letztendlich) Small – klein – genug sein, um im Rahmen eines Sprints entwickelt werden zu können, und dennoch bereits einen Wert für das Unternehmen oder andere Stakeholder liefern. Product Owner müssen Storys in Blöcke aufteilen, die klein genug sind, damit die Entwickler die Story in einer Iteration fertigstellen können. Die Techniken für das Aufteilen von Storys werden in 0 behandelt.
- **T**: User-Storys sollten Akzeptanzkriterien beinhalten, damit sie Testable – testbar – sind. Diese Kriterien werden in der Regel vor der Umsetzung der Story formuliert. Techniken für die Spezifikation von Akzeptanzkriterien werden in 3.5 behandelt. Product Owner sollten die Stolpersteine kennen, die durch nicht testbare User-Storys auftreten, und sie sollten diese in testbare User-Storys umformulieren können.

Storys werden in der Regel auf Karten erfasst. Beim 3C-Prinzip (Card, Conversation, Confirmation – Karte, Diskussion, Bestätigung) [Jeffries2001] wird betont, dass die Karte nur ein Mittel zum Zweck ist – ein physischer Gegenstand in greifbarer, beständiger Form von etwas, das andernfalls lediglich eine Abstraktion wäre. Die Karte sollte nicht als Äquivalent für eine gut formulierte Anforderung betrachtet werden. Die physische Card/Karte (bzw. deren Äquivalent in einem Werkzeug) ist dazu da, die Conversation/die Diskussion über das Thema anzustoßen, um Product Owner, andere Stakeholder und die Entwickler zur Besprechung der Karte zusammenzubringen und ein tieferes Verständnis des Inhalts zu erlangen sowie offene Fragen zu klären. Unabhängig vom Umfang der Diskussionen folgt die Confirmation/die Bestätigung als Feuerprobe für die Story: In der Regel werden Akzeptanztests der Story hinzugefügt, d.h. die Aspekte, die der Product Owner prüft, wenn das Team die vollständige Implementierung der Story meldet.

Bei agilen Entwicklungsprozessen steht zwar die Anwendung von User-Storys im Mittelpunkt, dies schließt jedoch den Einsatz von herkömmlichen Analyseartefakten wie Kontextdiagrammen, Business Use Cases, System Use Cases, Objektmodellen, Zustandsdiagrammen, Aktivitätsdiagrammen usw. nicht aus, wenn diese im Kontext eines bestimmten Projekts zweckmäßig ist (vgl. z.B. [Scrumguide]). Ein geschulter Product Owner sollte verstehen, wie User-Storys zusammen mit anderen Artefakten verwendet werden können und er sollte den Stakeholdern bei Bedarf deren Bedeutung erläutern können.

3.4 Aufteilungs- und Gruppierungstechniken (K4)

Dauer: 45 Minuten + 30 Minuten Übungszeit

Begriffe: zusammengesetzte Story, vertikales Schneiden, horizontales Schneiden, Aufteilungsmuster, Gruppierungs- und Abstraktionsmuster, Story-Map, Spike

Lernziele

LZ 3.4.1 Aufteilungstechniken für grobe funktionale Anforderungen analysieren und anwenden, um detailliertere, präzisere Anforderungen zu erhalten (K4)

LZ 3.4.2 Das Gruppieren und Abstrahieren von detailgenauen funktionalen Anforderungen in gröbere Anforderungen analysieren und anwenden, um mit Komplexität umzugehen, einen besseren Überblick zu erhalten sowie übergeordnete Planungen zu erstellen (K4)

Um User-Storys zu erstellen, die klein genug sind, um sie in einem einzigen Sprint umsetzen zu können, sollten größere Storys in detailliertere Storys aufgeteilt werden. Zu diesem Zweck lassen sich verschiedene Muster anwenden, die von der Verringerung der Featureliste bis zur Eingrenzung der geschäftlichen Variationen oder Informationskanäle reichen [Leffingwell2010]. Wenn eine User Story ein inakzeptables Maß an Unsicherheiten aufweist, kann der Fokus der Entwickler in einem bestimmten Sprint mithilfe von Spikes auf das Problem gelenkt werden.

Selbst detaillierte User-Storys sollten so definiert werden, dass sie für mindestens einen Stakeholder einen gewissen Wert liefern.

Die Zerlegung von User-Storys resultiert in Anforderungshierarchien, wie in 3.1 erläutert. Diese hierarchische und dreidimensionale Struktur lässt sich auch als zweidimensionale Story-Map darstellen [Patton2014]. In der horizontalen Dimension werden größere Gruppierungen (wie große Storys, Epics, Features) dargestellt, was einen Überblick über die Anforderungen bietet; in der vertikalen Dimension können sämtliche Details für die größeren Gruppen hinzugefügt und nach deren Zuordnung zu Sprints und Releases geordnet werden.

Es ist meist am besten, diese Zerlegung und Gruppierung von Anforderungen gemeinsam mit anderen Teammitgliedern durchzuführen, die tiefere Einblicke in das Geschäft und die technischen Abhängigkeiten haben, die zwischen den User-Storys bestehen können.

3.5 Wann sollten Sie aufhören zu zerlegen (K4)?

Dauer: 45 Minuten + 30 Minuten Übungszeit

Begriffe: Definition of Ready (DoR), Definition of Done (DoD), Schätzen, Akzeptanztest

Lernziele

LZ 3.5.1 Analysieren und anwenden der Verfeinerung von Anforderungen bis zu einem Maß, das für die Implementierung dieser Anforderungen adäquat ist (K4)

LZ 3.5.2 Qualitätssicherung von User-Storys in agilen Softwareentwicklungen analysieren und anwenden (K4)

Der Product Owner ist für die Fortführung der Diskussionen mit den Entwicklern verantwortlich, bis beide Seiten ein gemeinsames Verständnis von den Anforderungen erreicht haben [Meyer2014]. Um zu beurteilen, wann dieser Punkt erreicht ist, kann das

Pareto-Prinzip angewendet werden: Anforderungen müssen nicht zu 100 % perfekt definiert sein, aber gut genug, um die zentralen Fragen des Teams zu adressieren, und klar genug, damit der Implementierungsaufwand geschätzt werden kann. Beginnt man mit der Implementierung, wenn noch zu viele Fragen offen sind, kann das die Geschwindigkeit der Entwicklung erheblich verringern und gemessen an den Prognosen zu Verzögerungen führen.

Für diese Stufe des gemeinsamen Verständnisses wurde für Agile die Definition of Ready (DoR) [AgileAlliance] definiert. Eine Story ist „ready", also bereit, wenn sie die INVEST-Kriterien [Wake2003] erfüllt; dies gilt vor allem für die letzten drei Buchstaben des Akronyms:

- Die Entwickler konnten die Story schätzen und der geschätzte Wert ist klein genug, damit die Story in einer Iteration umgesetzt werden kann. Lawrence zufolge sollte die Story nicht nur in eine Iteration passen, sondern außerdem so klein sein, dass der nächsten Iteration 6 bis 10 Storys zugewiesen werden können.
- Der Product Owner hat Akzeptanzkriterien für die Story geliefert. Auf Basis des CCC-Prinzips stimmen alle darüber überein, dass genügend Diskussionen stattgefunden haben (Conversation) und die Kriterien zur Bestätigung (Confirmation) einer erfolgreichen Umsetzung in Form von Akzeptanztests definiert wurden.

Die Formulierung von Akzeptanzkriterien kann auf unterschiedliche Art und Weise erfolgen [Beck2002]. Die Akzeptanzkriterien können in informeller, natürlicher Sprache formuliert sein und nach der Implementierung geprüft werden. Sie können mithilfe der Gherkin-Syntax (Angenommen/Wenn/Dann) ein wenig formaler gestaltet werden, um die Sätze zu strukturieren.

Einige Methoden befürworten sogar die Verwendung von TDD (Test Driven Development) mit einer formalen Codierung der Testfälle, damit sie nach der Implementierung automatisch ausgeführt werden können [Meyer2014]. Dieser formale Ansatz ist zwar sehr präzise, kann jedoch für Product Owner und betriebswirtschaftlich orientierte Stakeholder schwer durchzuführen sein.

Die Definition of Ready (DoR) ist für den Product Owner das Äquivalent zur Definition of Done (DoD) der Entwickler. Nach den in der DoD festgelegten Kriterien wird bestimmt, ob eine Story erfolgreich implementiert wurde. Die DoR hingegen definiert, dass die Entwickler genügend Informationen über eine User Story haben, um sie innerhalb eines Sprints zu erledigen.

Die Besprechung von Anforderungen mit Entwicklern benötigt Zeit und findet am besten vor der Planung eines Sprints ("sprint planning") statt. Bei der Planung kann dann der Fokus auf die Auswahl der richtigen User-Storys gelegt werden und deren Zuordnung zu den verantwortlichen Entwicklern. Im Idealfall haben die Entwickler die Entstehung der Anforderungen verfolgt und den Product Owner mit ihren Fragen und Schätzungen unterstützt.

Es sind auch unterschiedliche Formen möglich ein Produkt Backlog Refinement durchzuführen. Zum Beispiel sind Refinement Meetings eine effizientere Möglichkeit, um Verfeinerungen durchzuführen, als einzelne Entwickler wiederholt zu stören.

Die Zeit für Refinement Meetings und sämtliche dazugehörigen Aktivitäten geht von der für die gesamte Iteration verfügbaren Zeit ab. Im Scrum Guide wird empfohlen, maximal 10% der Kapazität der Entwickler für die Verfeinerung zu verwenden: Wenn mehr Zeit als das benötigt wird, sollte die Iteration neu geplant werden, um mehr Genauigkeit in das Product Backlog zu bekommen, da dies ein Warnzeichen für eine schlechte Qualität der Anforderungen ist. Product Owner sollten das Verhältnis zwischen Länge der Iteration, Risiko und Verwaltungsaufwand für die Iteration kennen. Weiter sollten sie wissen, dass es kürzere Feedbackschleifen als den Sprint an sich gibt.

3.6 Projekt- und Produktdokumentation von Anforderungen (K2)

Dauer: 15 Minuten

Begriffe: Projekt- und Produktdokumentation, Gründe für die Aufbewahrung von Dokumentation

Lernziele

LZ 3.6.1 Verstehen, wie zwischen Projekt- und Produktinformationen/Dokumentation unterschieden wird (K2)

LZ 3.6.2 Methoden und Techniken zur Aufbewahrung von Informationen für die künftige Verwendung kennen (K1)

Für das Projektteam sind Story-Cards zur Erinnerung an die laufenden Diskussionen ausreichend, um als Grundlage für die Entwicklung zu dienen [Cohn2010]. Für Entwickler, die kontinuierlich mit einem Projekt beschäftigt sind, ist oftmals der eigentliche Code ausreichend, um zu verstehen, was zuvor getan wurde.

Es gibt jedoch über die direkt an der Projektentwicklung Beteiligten hinaus noch zahlreiche andere Stakeholder – den Kunden, das weitere Unternehmensumfeld, Support-Teams, andere betroffene Projektteams usw. –für die der Code und die User-Storys nicht genügend Kontext und Struktur bieten, um zu verstehen, wie sich die Arbeit auf sie auswirken wird. Deshalb stellen diese Stakeholder unterschiedliche Zielgruppen für die Dokumentation dar.

Des Weiteren hat das aus einem Entwicklungsprojekt resultierende Produkt (hoffentlich) eine Lebensdauer, die über den Projektzeitraum hinausreicht, und es kann durchaus in mehreren Entwicklungsprojekten weiterentwickelt werden. Requirements-Engineering-Artefakte können in der Regel als ausschließlich projektrelevant eingeordnet werden, oder sie bilden einen Teil der Produktdokumentation, die für die zukünftige Verwendung aufbewahrt wird. Die Trennung von Produkt- und Projektangelegenheiten kann sich als gute Investition in eine nachhaltige Dokumentation erweisen.

Nach den Prinzipien agiler Entwicklungsprozesse sollte nur dann Aufwand in die Dokumentation fließen, wenn es dafür einen Konsumenten gibt. Außerdem ist bei der Entwicklung dieses Artefakts regelmäßiges Feedback von diesem Konsumenten wichtig. Zu den Techniken zur Minimierung des Dokumentationsaufwands zählen u.a. dedizierte, Wiki-orientierte Dokumentationssysteme, vor allem für produktorientierte Artefakte, die sich im Laufe der Zeit weiterentwickeln [Weerd et al.2006].

4 Umgang mit Qualitätsanforderungen und Randbedingungen (K3)

Dauer: 90 Minuten + 30 Minuten Übungszeit

Begriffe: Qualitätsanforderungen, Randbedingungen, Qualitätsbaum, DoD

In dieser Lerneinheit werden die Qualitätsanforderungen und Randbedingungen in agilen Entwicklungsprojekten beleuchtet. Auch wenn der Begriff „nicht-funktionale Anforderungen" in der Praxis immer noch häufig als Überbegriff zu finden ist, verwenden wir die konkreteren und präziseren Kategorien „Qualitätsanforderungen" und „Randbedingungen" nach [Glinz2014].

Zu Beginn sind Qualitätsanforderungen oftmals bewusst vage formuliert. Sie müssen als Ausgangsbasis in dieser vagen Form erfasst werden. Vage Qualitätsanforderungen und Randbedingungen können in präzisere Anforderungen transformiert werden. Dabei werden manchmal aus vagen Qualitätsanforderungen konkrete funktionale Anforderungen abgeleitet.

4.1 Die Bedeutung von Qualitätsanforderungen und Randbedingungen verstehen (K2)

Dauer: 15 Minuten

Lernziele

LZ 4.1.1 Mit der Bedeutung von Qualitätsanforderungen in agilen Projekten verstehen (K2)
LZ 4.1.2 Kategorisierungsschemata für Qualitätsanforderungen und Randbedingungen verstehen (K2)

Viele agile Methoden sind ausschließlich auf funktionale Anforderungen ausgerichtet und legen nicht genügend Gewicht auf Qualitätsanforderungen und Randbedingungen [Meyer2014].

Die für ein System vorgesehenen zentralen Randbedingungen und Qualitäten sollten im Lebenszyklus eines Produkts frühzeitig explizit benannt werden, da sie für wichtige Entscheidungen hinsichtlich der Architektur ausschlaggebend sind (Infrastruktur, Softwarearchitektur und Softwaredesign). Werden sie in einem Projekt ignoriert oder zu spät erkannt, gefährdet dies unter Umständen die gesamte Entwicklungstätigkeit. Andere Qualitäten können iterativ erfasst werden, just in time, so wie es auch bei funktionalen Anforderungen der Fall ist [Meyer2014].

Kategorisierungsschemata für Qualitätsanforderungen und Randbedingungen (z.B. [RoRo2012], [ISO25010]) können als Checklisten verwendet werden, damit keine wichtigen Kategorien vergessen werden.

4.2 Qualitätsanforderungen präzisieren (K2)

Dauer: 30 Minuten

Lernziele

LZ 4.2.1 Detaillierung und Zerlegung von Qualitätsanforderungen und Randbedingungen verstehen (K2)
LZ 4.2.2 Ableitung Funktionaler Anforderungen aus Qualitätsanforderungen verstehen (K2)
LZ 4.2.3 Spezifikation von Akzeptanzkriterien für Qualitätsanforderungen verstehen (K2)
LZ 4.2.4 Verstehen des Mehrwerts von Qualitätsbäumen (K2)

Qualitätsanforderungen müssen auf eine Weise an die Entwickler kommuniziert werden, die eindeutig ist und die gleichzeitig die Anforderungsverfeinerung und -zerlegung ebenso wie für funktionale Anforderungen unterstützt.

Die Zerlegung (oder Detaillierung) von Qualitätsanforderungen bedeutet, dass diese auf einem niedrigeren Detaillierungsgrad spezifiziert werden, beispielsweise durch Generalisierung in Kategorisierungsschemata wie „Benutzbarkeit" und anschließend eine Präzisierung der Anforderungen erfolgt, beispielsweise für „einfache Benutzbarkeit" und „leichte Erlernbarkeit".

Ableiten bedeutet, dass man Qualitätsanforderungen durch das Definieren funktionaler Anforderungen erreichen kann, d.h. durch das Vorschlagen von Funktionen, die eine gewünschte Qualität oder Randbedingung bewirken. Ein Beispiel für die Verfeinerung einer Sicherheitsanforderung ist die Einführung eines Rollenkonzepts und Passwörtern. Qualitätsbäume ([BOSSANOVA], [Clements et al. 2001]) sind ebenfalls ein bewährtes Verfahren, um Qualitätsanforderungen zu strukturieren.

Für Qualitätsanforderungen müssen ebenfalls Akzeptanzkriterien definiert werden.

Qualitätsanforderungen sollten, genau wie andere Arten von Anforderungen, testbar sein [PoRu2015].

Die Art der verwendeten Akzeptanzkriterien hängt von der Qualitätskategorie ab: Ein quantifizierbares Akzeptanzkriterium, beispielsweise für eine Anforderung zur Benutzbarkeit, könnte lauten „90 von 100 Benutzern müssen in der Lage sein, eine Rechnung in weniger als drei Minuten zu erfassen".

4.3 Qualitätsanforderungen und Backlog (K3)

Dauer: 30 Minuten + 30 Minuten Übungszeit

Lernziele

LZ 4.3.1 Gegebenenfalls Zuordnung von Qualitätsanforderungen zu funktionalen Anforderungen anwenden können (K3)
LZ 4.3.2 Das separate Anlegen von Backlog Items für Qualitätsanforderungen anwenden können (K3)
LZ 4.3.3 Andere Qualitätsanforderungen als Teil der DoD verstehen (K2)

Generalisierte Qualitätsanforderungen müssen mit spezifischeren funktionalen Anforderungen verknüpft werden [PoRu2015], z.B. ein quantifizierbarer Durchsatz, der einer User Story hinzugefügt wird oder spezifische Sicherheitsfunktionen, die ein Epic ergänzen.

Andere Qualitäten wie Skalierbarkeit, Wartbarkeit oder Robustheit sollten der Entwicklung mitgeteilt und in jeder einzelnen Iteration geprüft werden. Zu diesem Zweck werden diese Qualitäten üblicherweise in die Definition of Done aufgenommen. Unterstützt wird dies häufig durch automatisierte Tests [Leffingwell2010].

Ein weiterer Ansatz ist die separate Aufzeichnung (außerhalb des Product Backlog) dieser Qualitäten, etwa in Form einer gemeinsamen Liste oder einer Checkliste, damit sie für die Teams immer präsent sind; diese Anforderungen sind alle gleichrangig (da sie alle erfüllt werden müssen) [Leffingwell2010].

Es hat sich außerdem bewährt, die Beziehungen zwischen funktionalen und betroffenen Qualitätsanforderungen, etwa durch eine Matrix an einer Wand, sichtbar zu machen und die Beziehung „betroffen von" in den entsprechenden Zellen zu kennzeichnen.

4.4 Randbedingungen verdeutlichen (K2)

Dauer: 15 Minuten

Lernziele

LZ 4.4.1 Verschiedene Arten von Randbedingungen in agilen Projekten verstehen (K2)
LZ 4.4.2 Randbedingungen charakterisieren können (K1)

Randbedingungen sind eine wichtige Art von Anforderungen, welche die Designentscheidungen der Entwickler eingrenzt [Glinz2014]. Randbedingungen lassen sich entweder als Produkt- oder als Prozessrandbedingungen kategorisieren. Produktrandbedingungen beinhalten die geforderte Verwendung von Technologien, die Wiederverwendung vorhandener Komponenten, Entscheidungen über "entwickeln oder kaufen", bzw. Ressourcen in Form von Material, Wissen und Kompetenzen. Die Prozessrandbedingungen geben hingegen organisatorische oder Entwicklungsprozesse vor. Dazu zählen organisatorische Richtlinien und Vorschriften, finanzielle Limits, Normen, Compliance-Vorschriften und Audits sowie rechtliche und behördliche Randbedingungen.

Es ist wichtig, diese Randbedingungen explizit zu machen, damit sich jeder im Team dieser bewusst ist. Randbedingungen, die für die größten Einschränkungen sorgen, sollten im Projektverlauf frühzeitig bekannt sein. Sonstige Randbedingungen sollten erfasst werden, sobald sie erkannt werden.

Derartige Randbedingungen gelten normalerweise für eine Vielzahl von Projekten. Wenn sie also einmal erfasst wurden, können sie leicht wiederverwendet werden.

5 Priorisieren und Schätzen von Anforderungen (K3)

Dauer: 120 Minuten + 90 Minuten Übungszeit

Begriffe: Backlog-Sortierung und -Priorisierung, Geschäftswert, MVP, MMP, ROI, Kosten für Verzögerungen ("cost of delay"), WSJF, Markteinführungszeit ("time to market"), Planning Poker, Estimation Wall, Cone of Uncertainty, Velocity, Sizing, Referenz-Storys, T-Shirt-Größen, Fibonacci-Folge

5.1 Ermittlung des Geschäftswerts (K3)

Dauer: 45 Minuten + 15 Minuten Übungszeit

Lernziele

LZ 5.1.1 Die Ermittlung des Geschäftswerts verstehen (K2)
LZ 5.1.2 Verstehen, wie man den Geschäftswert nutzt, um Backlog Items zu sortieren (K2)
LZ 5.1.3 Anwendung alternativer Berechnungsmethoden für den Geschäftswert (K3)
LZ 5.1.4 Verstehen, wie man Maßnahmen zur Bewertung des Geschäftswerts auf strategische Ziele der Organisation ausrichtet (K2)
LZ 5.1.5 Das Konzept des MVP (Minimum Viable Product) verstehen (K2)
LZ 5.1.6 Das Konzept des MMP (Minimum Marketable Product) verstehen (K2)

Agile Herangehensweisen zielen darauf ab, den gesamten Geschäftswert zu maximieren und den Erstellungsprozess des Geschäftswerts insgesamt dauerhaft zu optimieren [Leffingwell2010]. Sämtliche Anforderungen (ob grob- oder feingranular) sollten in erster Linie nach dem Wert sortiert werden, den sie der Organisation bieten. Eine Voraussetzung dafür ist eine abgestimmte Definition dessen, was den Geschäftswert für das Produkt/Unternehmen ausmacht.

Der Geschäftswert wird nicht nur über den Profit definiert: Zu alternativen Berechnungsmethoden zählen u. a. die Gesamtkapitalrentabilität (Return on Investment, ROI), die Amortisationszeit, der Kapitalwert (Net Present Value), Weighted Shortest Job First (WSJF, „Wert/Aufwand-Verhältnis"), die Kosten für Verzögerungen (Cost of Delay) und die Balanced Scorecard. Der Marktwert, die Markteinführungszeiten (time to market) und die Reduzierung potenzieller Risiken stellen allesamt potenzielle Arten von Geschäftswert dar, ebenso wie betriebliche und organisatorische Exzellenz [Reinertsen2009].

Die Definition des Geschäftswerts kann in der Tat für jede Organisation, jedes Projekt und aus der Perspektive unterschiedlicher Stakeholder anders aussehen. Experten sollten wissen, wie sie Maßnahmen zum Erzielen des Geschäftswerts auf die strategischen Ziele der Organisation ausrichten und sie sollten diese Ausrichtung bei einer Veränderung der Ziele anpassen können.

Das KANO-Modell ist eine Technik, mit der sich der Geschäftswert in den Anforderungen identifizieren lässt. Im Allgemeinen ist die relative Einschätzung des Geschäftswertes manchmal gegenüber der Berechnung des absoluten Geschäftswerts vorzuziehen.

Zentrale Begriffe in agilen Entwicklungsprozessen sind MVP (das Minimum Viable Product) und MMP (das Minimum Marketable Product) [IREB2017].

Das MVP kann den Aufwand erheblich reduzieren, wobei der Geschäftswert nahezu gleich bleibt. MVPs lassen sich nicht nur für das gesamte Produkt, sondern auch für ein bestimmtes Feature definieren.

Ein MVP eignet sich zwar sehr gut für validiertes Lernen, für den langfristigen Betrieb reicht es allerdings nicht aus; für Bereitstellungen in der Praxis ist das MMP für das Unternehmen sehr wichtig. Bei einem MVP geht es primär um die Reduzierung des Systemumfangs: Der Geschäftswert soll mit dem minimal möglichen Feature-Set geschaffen und bei erfolgreichem Einsatz der Features später gesteigert werden.

5.2 Geschäftswert, Risiken und Abhängigkeiten (K3)

Dauer: 45 Minuten + 45 Minuten Übungszeit

Lernziele

LZ 5.2.1 Priorisierung von Backlogs anwenden (K3)
LZ 5.2.2 Verschiedene grundlegende Priorisierungsstrategien anwenden (K3)
LZ 5.2.3 Ermittlung von Abhängigkeiten von Anforderungen anwenden (K3)
LZ 5.2.4 Reihenfolge von Backlog Items durch Berücksichtigung der Abhängigkeiten verstehen (K2)
LZ 5.2.5 Die Abhängigkeiten zwischen potenziellem Geschäftswert und zugehörigen Risiken verstehen (K2)

In den unterschiedlichen Arten von Backlogs (Unternehmen, Produkt, Sprint) können verschiedene grundlegende Priorisierungsstrategien angewendet werden [Leffingwell2010]. Besteht beispielsweise das primäre Ziel eines Unternehmens darin, das Produkt frühzeitig auszuliefern und Marktanteile zu gewinnen, könnte es eine Strategie wählen, mit der es rasch einen Geschäftswert erzielt. Ein Lieferant, der etwa um jeden Preis Produktrücksendungen aufgrund von mangelhafter Leistung oder Sicherheitsmängeln vermeiden möchte, wird dagegen eine Strategie der frühen Risikoreduktion bevorzugen.

Es bestehen sehr häufig wechselseitige Abhängigkeiten zwischen potenziellem Geschäftswert und Risiken. Die Konzentration auf einen bestimmten Geschäftswert kann gewisse Risiken verursachen, wird der Fokus auf den Geschäftswert geändert, können sich auch die Risiken ändern [Reinertsen2009].

In jedem Fall sollte die Sortierung von Anforderungen gemäß der gewählten Strategie angepasst und dabei die Abhängigkeiten zwischen den Anforderungen berücksichtigt werden.

5.3 Schätzen von User-Storys und anderen Backlog Items (K3)

Dauer: 30 Minuten + 30 Minuten Übungszeit

Lernziele

LZ 5.3.1 Prognosen und Schätzungen anwenden (K3)
LZ 5.3.2 Verstehen, wie man eine mittelfristige Prognose ableitet (K2)
LZ 5.3.3 Die Vorteile von relativen, kategorisierenden und Gruppenschätzungen verstehen (K2)
LZ 5.3.4 Verstehen von Schätzungstechniken (K2)

Selbst in einer perfekten agilen Welt ist man auf Prognosen angewiesen, um zu ermitteln, wie viel Arbeit in einer zuvor festgelegten Iteration (Timebox) erledigt werden kann. Aus zwei Gründen dürfen keine Elemente ohne Schätzung in einen Sprint in Scrum aufgenommen werden [Cohn2005]:

1. Es ist unklar, ob das Element innerhalb des Sprints abgeschlossen werden kann und dadurch die Auslieferung funktionierender Software am Ende des Sprints verhindert wird.
2. Ohne Diskussionen und Schätzungen hat ein Team keinen Referenzpunkt (Planung vs. tatsächliche Ausführung), um daraus für die folgenden Sprints zu lernen.

Zudem sind Entwicklungsorganisationen mit mehr als einem Team gewöhnlich auf Prognosen angewiesen, um die Arbeit richtig zu priorisieren und zu planen.

Die Prognose von Entwicklungen findet in unterschiedlichen zeitlichen Rahmen bzw. Zeithorizonten statt. Das Übereinkommen darüber, welche Anforderungen der nächsten Iteration zuzuordnen sind, sollte recht detailliert ausfallen; längerfristige Iterationspläne enthalten dagegen weniger Details, sie sollten jedoch grobe Schätzungen und Planungen ermöglichen.

Mit einer solchen Planung können auch Fortschrittsberichte erstellt werden, etwa darüber, welche Epics, Features und User-Storys tatsächlich zu den erwarteten Releaseterminen geliefert wurden. Dieses Vorgehen unterstützt außerdem die Identifizierung von Anforderungen, die aufgeteilt werden müssen, weil sie zu groß für eine Schätzung sind [Leffingwell2010].

Die initialen Projektschätzungen sind häufig ungenau. Sie werden jedoch zunehmend genauer, da die Aktivität iteriert wird (ein Prinzip, das als „Cone of Uncertainty" bekannt ist). Durch die Analyse, was in vorherigen Iterationen geliefert wurde, kann die Arbeitsgeschwindigkeit ("velocity") des Teams berechnet werden. Damit lässt sich die Kapazität für künftige Iterationen besser schätzen [McConnel2006].

Für bessere mittelfristige Schätzungen werden große Anforderungen wie Epics auf Features heruntergebrochen. So können Schätzungen auf Feature-Ebene durchgeführt und dann zum Epic summiert werden. Diese Schätzungen sind zwar immer noch nicht sehr präzise, aber für die Epic-Schätzung hilfreich. Außerdem dienen sie als eine Art Prüfung der Kenntnisse über das Epic (Annahmen usw.).

Agile Methoden empfehlen einige bewährte Verfahren, mit deren Hilfe bessere und präzisere Schätzungen möglich sind:

- Alle an der Schätzung Beteiligten müssen dasselbe Verständnis der zu erledigenden Arbeit haben.
- Schätzungen müssen von denjenigen vorgenommen werden, die die Arbeit ausführen, also dem cross-funktionalen Team (Entwickler in Scrum). Durch den Austausch von Wissen und Annahmen über die zu erledigende Arbeit werden alle beteiligten Personen auf denselben Wissensstand gebracht.
- Schätzungen sollten relativ zur bereits erledigten Arbeit vorgenommen werden (Schätzung durch Analogie), da die Wahrscheinlichkeit größer ist, dass sie präziser ausfallen als absolute Schätzungen.
- Schätzungen sollten in einer künstlichen Einheit erfolgen, die Aufwand, Komplexität und Risiko vereint. Die Verwendung einer künstlichen Einheit ist notwendig, um alle auf die neue Art der Schätzung einzustimmen, da der Einsatz von Stunden/Tagen in Schätzungen zu bestehendem, herkömmlichem Verhalten verleitet.
- Die Verwendung des Auszugs aus der Fibonacci-Folge oder von T-Shirt-Größen erleichtern diese Art der Schätzung (Verwendung von Begriffen wie „größer als/kleiner als" anstelle von „genau zweimal mehr als ...").

Für relative Schätzungen stehen mehrere Techniken zur Verfügung. Die bekannteste davon ist das sogenannte Planning Poker [Cohn2005]. Beim Planning Poker besprechen die Entwickler die Anforderung und jeder wählt eine Karte aus dem Planning-Poker-Set aus, das auf der Fibonacci-Folge beruht und die relative Größe der neuen Anforderung im Verhältnis zu einer gemeinsamen Basis darstellt. Nachdem jeder eine Pokerkarte ausgewählt hat, diskutieren die Teammitglieder mit der niedrigsten und der höchsten Schätzung die Gründe für ihre Schätzungen und versuchen, die anderen Teammitglieder von ihrer Argumentation zu überzeugen. Anschließend wird die nächste Schätzungsrunde gestartet.

Kann sich das Team nicht innerhalb von drei Runden auf einen gemeinsamen Wert einigen, wird die Anforderung an den Product Owner zurückgegeben. Einigt sich das Team, dann wird der gemeinsame Wert zugeordnet.

Der Vorteil der Planning-Poker-Technik ist, dass sie sich sehr gut für die Schätzung durch neue und unerfahrene Teams eignet, da ein zu starker Einfluss ("Anchoring") durch einzelne Teammitglieder verhindert wird. Nachteilig ist, dass das Planning Poker sehr zeitaufwendig ist. [Hinweis: Das Buch „Schnelles Denken, langsames Denken" von D. Kahneman [Kahneman2016] bietet eine sehr gute Einführung in die unbewusste Beeinflussung ("Anchoring") und andere psychologische Effekte in Bezug auf Denken und Urteilsvermögen.]

Um diesen Nachteil zu überwinden, werden in erfahreneren Teams verbesserte Techniken angewandt.

Eine Vereinfachung der Planning-Poker-Technik beruht auf denselben Prinzipien wie das Planning Poker, dabei kommt jedoch eine andere Ermittlungsweise der richtigen Schätzung zum Einsatz.

Anstatt dass jedes Teammitglied eine eigene Schätzung erstellt, wird auf einem Tisch ein Poker-Kartenset verteilt, und die Referenzanforderungen werden in dem entsprechenden „Container" platziert, den die Pokerkarte darstellt. Anschließend werden die Anforderungen von den Teammitgliedern nach einem Round-Robin-Ansatz ausgewählt, wobei jedes Teammitglied die Möglichkeit hat, eine neue Anforderung in dem entsprechenden „Container" zu platzieren oder eine bereits platzierte Anforderung einem anderen Container zuzuordnen. Wird eine Anforderung mehrmals neu zugewiesen, wird sie entfernt und an den Product Owner zurückgesendet. Dieser Ansatz ist wesentlich zeitsparender, aber die Voraussetzung dafür ist ein Team, das reif genug ist, um Zuordnungen von anderen Teammitgliedern abzulehnen, anstatt diesen einfach zuzustimmen (das sog. „Anchoring").

Der nächste Entwicklungsschritt wird gewöhnlich als „Affinity Estimation" bezeichnet und wird zur Schätzung größerer Anforderungsmengen verwendet, z.B. für die Grobschätzung als Vorbereitung für Release-Planungen. Der Unterschied zum vorherigen Ansatz besteht darin, dass die Anforderungen nicht per Round-Robin-Ansatz zugeordnet werden, sondern jedes Teammitglied einen Teil der Anforderungen erhält und diese stillschweigend den über das Poker-Kartenset dargestellten „Containern" zuordnet. Nach der stillschweigenden Zuordnung dürfen alle Beteiligten die zugeordneten Anforderungen prüfen und in Frage gestellte Zuordnungen kennzeichnen. Dies führt in der Regel zu einer Quote von 20 bis 30 % von Anforderungen, für die Diskussionsbedarf besteht, und 70 bis 80 % von Anforderungen, die von allen Teammitgliedern akzeptiert werden.

6 Skalierung von RE@Agile (K2)

Dauer: 105 Minuten

Begriffe: Skalierungs-Frameworks, Skalierung von Anforderungen und Teams, Strukturierung von Anforderungen und Teams im Großen, Roadmaps und umfangreiche Planung, Produktvalidierung

6.1 Skalierung von Anforderungen und Teams (K2)

Dauer: 30 Minuten

Lernziele

LZ 6.1.1 Beschreiben gängiger Beispiele für Skalierungs-Frameworks (K1)
LZ 6.1.2 Verstehen der Herausforderungen und Mechanismen zur Skalierung agiler Anforderungen (K2)

Für die Skalierung agiler Entwicklungsprozesse gibt es unterschiedliche Frameworks und Implementierungen. Bei den Frameworks handelt es sich um Generalisierungen konkreter Implementierungen, die durch etablierte Experten und Beratungsunternehmen zur Skalierung agiler Entwicklungsprozesse geprägt wurden. Alle Skalierungs-Frameworks beruhen auf agilen Werten und Prinzipien. Jedes spezifische Framework basiert auf einer Auswahl bewährter agiler Verfahren und bestimmter Methoden, Techniken und anderer Elemente, die in einem kohärenten Gesamtkonzept kombiniert wurden. Skalierungs-Frameworks variieren hinsichtlich ihres Reifegrades, der Anzahl bewährter Verfahren, Richtlinien und Regeln sowie dem Grad der Anpassbarkeit des Frameworks an bestimmte Bedürfnisse einer Organisation.

Seit etwa 2010 wurde eine Reihe verschiedener agiler Skalierungs-Frameworks entwickelt. Dazu gehören Nexus [NEXUS], SAFe [SAFe], LeSS [LeSS], Scrum@Scale [S@S], BOSSA Nova [BOSSANOVA], Scrum of Scrums [SofS]und Spotify [Spotify2012].

Skalierungs-Frameworks werden auch deshalb eingesetzt, weil es manchmal nicht ausreicht, mit nur einem Team zu arbeiten: Wenn entweder das Produkt sehr komplex ist oder eine kürzere Zeit bis zur Marktreife benötigt wird, muss die agile Entwicklung auf mehrere Teams skaliert werden. Darüber hinaus könnte die geografische Verteilung oder die Qualifikation der Beteiligten zu verteilten Teams führen. Sobald wir mehrere Teams parallel arbeiten lassen, besteht die Herausforderung darin, dass die Anforderungen koordiniert und die Kommunikation zwischen den Teams definiert werden muss.

Anforderungen müssen gruppiert werden, um sie verschiedenen Teams zuordnen zu können. Für diese Gruppierungen schlagen die Frameworks unterschiedliche Namen für verschiedene Abstraktionsebenen vor. Wir empfehlen dringend, dass sich jede Organisation auf die Namen für diese verschiedenen Anforderungsstufen einigen und Richtlinien für die verschiedenen Stufen definieren sollte, und sich dann an diese halten sollte. Diese Namen sind oft durch das Skalierungs-Framework oder die zur Erfassung der Anforderungen verwendeten Werkzeuge vordefiniert.

Jedes Team benötigt jemanden, der die Verantwortung für das Anforderungsmanagement übernimmt. Für ein einzelnes Team wird diese Person üblicherweise als Product Owner bezeichnet, wie wir in den vorherigen Kapiteln erläutert haben. Wenn Anforderungen für mehrere Teams verwaltet werden müssen, schlagen einige Frameworks unterschiedliche Job-Titel für die Verwaltung von Anforderungen in größeren Teams vor. Unabhängig von den Job-Titeln: Stellen Sie sicher, dass es auf jeder Organisationsebene eine klare Verantwortung für das Anforderungsmanagement gibt.

Einzelne Teams arbeiten in kurzen Iterationen, oft Sprints genannt (wie in den vorherigen Kapiteln besprochen). Zu Beginn eines Sprints wird eine Menge von Anforderungen aus dem Product Backlog ausgewählt, und am Ende des Sprints wird deren Umsetzung überprüft. Bei der Skalierung fassen die Frameworks diese kurzen Iterationen oft zu längeren Iterationen zusammen (die z. B. zwei oder drei Monate statt 2 – 4 Wochen dauern), die als Releases bezeichnet werden. Daraus ergibt sich die Notwendigkeit von Release-Backlogs, die den Kontext und die Ziele für die beteiligten Teams zur Organisation ihrer Sprints und die Integration der Sprint-Ergebnisse vorgeben.

6.2 Kriterien für die Strukturierung von Anforderungen und Teams im Großen (K2)

Dauer: 30 Minuten

Lernziele

LZ 6.2.1 Kennen der Prinzipien zur Organisation eines Backlogs und zur Kommunikation über Anforderungen in einem Netzwerk von Teams (K1)

LZ 6.2.2 Verstehen der Anforderungsaufteilung in verschiedenen (Projekt-)Umgebungen (K2)

Aus der Anforderungsperspektive müssen wir den "Kreislauf" schließen, von der anfänglichen (Business-)Anforderung der Stakeholder, über die Aufteilung komplexer Anforderungen in kleinere, von jeweils einem Team an Entwicklern handhabbare Teile, bis hin zur Sicherstellung, dass die zusammengesetzten Ergebnisse eine Lösung bilden, die für das Business freigegeben werden kann.

Ausgefeilte Strukturen und Praktiken werden benötigt, um die Zusammenarbeit in Teams, Anforderungsänderungen und eine schnelle Produktlieferung in der groß angelegten Produktentwicklung zu unterstützen. Andernfalls verschwenden Entwickler möglicherweise Aufwand bei der Koordination mit anderen Teams, die für ihre Arbeit nicht relevant sind.

Komplexe Anforderungen müssen von mehreren Product Ownern und Entwicklern verstanden und verwaltet werden. Dazu werden Anforderungen rekursiv in weitere Anforderungen aufgeteilt, bis sie einfach genug sind, um von einem Team entwickelt zu werden, wodurch eine Hierarchie von Anforderungen entsteht [GoWo2006]. Kommunikation ist sowohl nach oben als auch nach unten in dieser Hierarchie erforderlich, z. B. zwischen Product Ownern, die auf verschiedenen Abstraktionsebenen an verbundenen Anforderungen arbeiten (oder sich auf Prioritäten einigen), sowie zwischen den Entwicklern, z. B. bei der Zusammenstellung von Deliverables zu End-to-End-Produktinkrementen.

Um die Zusammenarbeit und Kommunikation richtig zu unterstützen, müssen die Anforderungen in einem logischen Backlog verwaltet werden. Der Kerngedanke ist, dass jede Anforderung nur an einer Stelle verwaltet wird, um Redundanzen oder Widersprüche zu vermeiden. Dies kann auch bei einer weiteren Unterteilung des Backlogs in Team-Backlogs erreicht werden. Während der Verfeinerung grober Anforderungen können Product Owner an Backlog Items arbeiten, die noch keinem Team zugeordnet sind, oder sie können komplexe Anforderungen aufteilen und die daraus resultierenden Backlog Items an die Teams zur weiteren Verfeinerung weitergeben (siehe [NEXUS], [SAFe], [LeSS] für weitere Informationen). Um die Verfolgbarkeit zwischen Anforderungen auf verschiedenen Abstraktionsebenen sicherzustellen, sollten Product Owner die jeweiligen Backlog Items miteinander verknüpfen.

Die Produktentwicklung wird es schwer haben zeitnah auf Änderungen zu reagieren, wenn jedes Team auf ein kompliziertes Geflecht von Interaktionen mit anderen Teams angewiesen ist, um jede Entscheidung abzustimmen. Es ist eine Teamstruktur erforderlich, die es den Teams ermöglicht, sich selbst um die Wertschöpfung herum zu organisieren: Um besser auf das Feedback der Stakeholder reagieren zu können, um unabhängig vernünftige Entscheidungen zu treffen und um End-to-End-Features zu liefern [Anderson2020], [Reinertsen2009].

Die Aufteilung von Anforderungen ist notwendig, um grobe Anforderungen herunterzubrechen, damit sie jeweils Entwicklern zugeordnet werden können. Um lieferbare Produktinkremente mit minimalen Abhängigkeiten zu anderen Teams zu liefern, sollten agile Teams an lose gekoppelten, End-to-End-Features arbeiten (vgl. Feature-Teams, wie sie in [Larman2017] eingeführt wurden). Um dies zu erreichen, wird der Produktumfang in kleinere, lose gekoppelte Einheiten mit intern konsistenter Funktionalität partitioniert (d.h. feature-basierte Aufteilung der Anforderungen). Jede Einheit hat klar definierte funktionale Grenzen. Die Grenzen sollten klar sein, um eine effektive Zusammenarbeit zu ermöglichen. Wenn die Anforderungen auch nach diesen funktionalen Grenzen aufgeteilt werden, können die einer bestimmten Einheit zugewiesenen Product Owner mit einem höheren Grad an Unabhängigkeit an Features arbeiten.

Leider ist es in vielen Fällen nicht so einfach, Anforderungen basierend auf lose gekoppelte Einheiten von einer End-to-End-Funktionalität zu zerlegen. Aufgrund des architektonischen Designs (z. B. Technologie, Infrastruktur, Systemkomponenten, gemeinsame Plattform, Architekturschichten wie Front- und Backend) sowie organisatorischer Überlegungen (fachliche Kompetenzen, Standort des Teams, Sub-Auftragnehmer) können sich Funktionalitäten überschneiden. Das bedeutet, dass verschiedene agile Teams zusammenarbeiten müssen, um bestimmte Funktionen zu implementieren und, dass ihre jeweiligen Product Owner bei den Anforderungen enger zusammenarbeiten müssen.

Um Features kollaborativ zu implementieren, benötigen agile Teams ein gemeinsames Verständnis der Anforderungen und deren geschäftlichen Kontextes. Sie müssen sich auch auf übergreifende Anforderungen, Constraints und gemeinsame technische Schnittstellen einigen, damit die Ergebnisse verschiedener Teams zu funktionierenden Inkrementen integriert werden können.

Die Integration und das Testen von Features werden komplexer und die Synchronisierung der Teams mithilfe von Backlogs und Roadmaps ist noch wichtiger. Ein anderer Ansatz besteht darin, ein separates agiles Team um bestimmte Features herum zu bilden, oder die notwendigen Engineers anzuheuern, um bestimmte Funktionen unabhängig zu entwickeln. Siehe auch [Anderson2020] für weitere Details zu agilem Organisationsdesign und Verfahren.

6.3 Roadmaps und umfangreiche Planung (K2)

Dauer: 30 Minuten

Lernziele

LZ 6.3.1 Charakterisieren des Unterschieds zwischen einer Roadmap und einem Backlog (K1)
LZ 6.3.2 Verstehen der Erstellung und des Managements einer Roadmap (K2)

In der groß angelegten Produktentwicklung verwalten Product Owner Anforderungen im produktfokussierten Backlog. Im Gegensatz zum Backlog wird eine Roadmap zur inkrementellen Planung der Produktentwicklung verwendet.

Eine Roadmap ist eine Vorhersage, wie das Produkt wachsen wird [Pichler2016]. Roadmaps verändern nicht den Inhalt der Backlog Items, sondern ordnen sie auf einer Zeitachse an. Sie beantworten die Frage, wann wir welche Features ungefähr erwarten können.

Eine Roadmap ist ein nützliches Mittel, um (strategische) Ziele und Entscheidungen an die Entwickler und andere Stakeholder zu kommunizieren. Sie bricht ein langfristiges Ziel in überschaubare Iterationen herunter, stellt Abhängigkeiten zwischen den Teams dar und gibt den Stakeholdern Orientierung und Transparenz.

Zu Beginn der agilen Produktentwicklung ist wenig über das Produkt oder die Arbeit der Teams bekannt. Der Umfang des Produkts sowie die Kostenschätzungen sind daher mit einem hohen Maß an Unsicherheit behaftet. Je mehr Iterationen durchgeführt werden und je mehr Feedback von den Stakeholdern eingeholt wird, desto mehr nimmt die Unsicherheit ab, was zu einer zuverlässigeren Planung und einer stabilen Roadmap führt. Dieses Prinzip ist als Cone of Uncertainty [Boehm1981] bekannt.

Obwohl dieser Grundsatz generell für alle agilen Entwicklungsprojekte gilt, wird er bei der Entwicklung umfangreicher Produkte noch wichtiger, da die Risiken aufgrund der Produktkomplexität und der potenziellen schlechten Abstimmung über mehrere Teams hinweg – und folglich der Bedarf an mehr Planung – noch größer sind.

Eine Roadmap zeigt strategische Ziele, Meilensteine und grobe Anforderungen. Wichtige Meilensteine können entweder intern oder durch externe Ereignisse, wie z.B. eine Messe oder die Einführung einer neuen Vorschrift, bestimmt sein. Die Darstellung einer Roadmap ist abhängig von ihrem Zweck, ihrer Zielgruppe und ihrem Planungshorizont.

Für Kunden, Management-Sponsoren und das Business ist eine langfristige **Produkt-Roadmap** mit strategischen Zielen und groben Produktanforderungen oft ausreichend [Pichler2016].

Umgekehrt müssen Entwickler die weiteren Details kennen, die durch detaillierte Backlog Items (zum Beispiel Storys und Tasks) repräsentiert werden, sowie die Abhängigkeiten zwischen ihnen. Diese Informationen werden durch mittelfristige **Delivery-Roadmaps (langfristige Versionsplanung)** [SAFe] bereitgestellt.

Um eine langfristige Produkt-Roadmap zu entwickeln, muss ein Product Owner zunächst eine Produktvision und eine Strategie definieren. Danach müssen Product Owner die groben Anforderungen erheben, indem sie sich mit den notwendigen Stakeholdern auseinandersetzen. Es ist nicht nötig, an dieser Stelle Zeit in detaillierte Anforderungen zu investieren. Obwohl die Anforderungen in diesem Stadium mit einem hohen Maß an Unsicherheit behaftet sind, ist die Produkt-Roadmap als grober Iterationsplan für den ersten Schnitt gut genug, um die Planung und Synchronisation zu unterstützen.

Um eine mittelfristige Delivery-Roadmap zu erstellen, müssen die Product Owner die Backlog Items aus der bestehenden Produkt-Roadmap verfeinern und priorisieren. Diese Items müssen von den Entwicklern grob geschätzt werden, auch wenn die Schätzungen in diesem Stadium noch ungenau sind (z. B. T-Shirt-Größen). Die Schätzung muss nur gut genug sein, um einen Überblick über kommende Iterationen zu ermöglichen.

Das Erstellen und Aktualisieren von Delivery-Roadmaps geschieht typischerweise bei Face-to-Face Planungsrunden, die als Big Room Planning (oder PI Planning in SAFe) bekannt sind und in regelmäßigen Abständen stattfinden.

6.4 Produkt-Validierung (K2)

Dauer: 15 Minuten

Lernziele

LZ 6.4.1 Verstehen konkreter Methoden zur Validierung von Produktanforderungen in der agilen Entwicklung (K2)

Der Grundgedanke der agilen Entwicklung ist es, einen kleinen Teil des Produkts zu entwickeln, durch die Einbindung von Stakeholdern Feedback zu generieren und die Produktentwicklung entsprechend der Erkenntnisse und Ergebnisse anzupassen. Zu diesem Zweck müssen die Product Owner eine neu freigegebene Produktversion verwenden, um deren Geschäftswert zu untersuchen und zu prüfen, ob die Produktanforderungen richtig verstanden wurden.

Ein Sprint-Review ist ein geeignetes Mittel, um den Stakeholdern ein im letzten Sprint entwickeltes Produktinkrement zu präsentieren. In einer umfangreichen Produktentwicklung kann die gleiche Idee ebenfalls verwendet werden. Aber anstatt einen einzelnen, von einem Team entwickelten Produkt-Teil zu überprüfen, werden alle Teamergebnisse zu einem funktionierenden Produktinkrement integriert, das es wert ist, validiert zu werden. Das Inkrement wird den Stakeholdern in einem **Produktreview (Demonstration)** vorgeführt, was zu einem besseren Eindruck des gesamten Produkts führt [SAFe], [Larman2017], [LeSS].

Ein weiterer Ansatz für die Produktvalidierung in der umfangreichen Produktentwicklung ist **Datenanalyse** [Maalej et al.2016]. Das integrierte Produktinkrement wird an Benutzer ausgeliefert und anhand deren Verhalten wird gemessen, ob die Produkt-Features einen positiven, neutralen oder negativen Einfluss haben. Product Owner können die Ergebnisse nutzen, um potenziell schlecht konzipierte Features zu identifizieren. Um die identifizierten Probleme besser zu verstehen, müssen sie möglicherweise erneut die üblichen Techniken zur Anforderungserhebung und -analyse anwenden.

BEGRIFFSDEFINITONEN, Glossar

Das Glossar definiert die für RE@Agile relevanten Begriffe. Das Glossar steht auf der IREB Homepage zum Download zur Verfügung: https://www.ireb.org/de/downloads/#re-agile-glossary

LITERATURVERZEICHNIS

[AgileAlliance] Glossar der Agile Alliance: Definition des Begriffs „Definition of Ready": https://www.agilealliance.org/glossary/definition-of-ready. Zuletzt besucht im April 2024.

[AgileManifesto2001] Agile Manifesto: http://Agilemanifesto.org 2001. Zuletzt besucht im April 2024.

[Alexander2005] Alexander, I. F.: A Taxonomy of Stakeholders – Human Roles in System Development. International Journal of Technology and Human Interaction, Vol 1, 1, 2005, pages 23–59.

[Anderson2020] Anderson, J.: Agile Organisationsgestaltung – Growing Self-Organizing Structure at Scale. Leanpub, 2020.

[Beck2002] Beck, K.: Test Driven Development: By Example. Addison Wesley, 2002.

[Boehm1981] Boehm Barry W.: Software Engineering Economics. Prentice Hall, 1981.

[BOSSANOVA] BOSSA nova , zuletzt besucht im April 2024.

[Clements et al. 2001] Clements, P., Kazman, R., Klein, M..: Evaluating Software Architectures. SEI Series in Software Engineering, 2001.

[Cohn2010] Cohn, M.: User Stories: für die agile Software-Entwicklung mit Scrum, XP u.a. Mitp-Verlag, 2010.

[Cohn2005] Cohn, M.: Agile Estimating and Planning. Prentice Hall, Nov 2005.

[Conway1968] Conway, M.E.: How Do Committees Invent? Datamation Magazine, 1968. http://www.melconway.com/Home/Committees_Paper.html. Zuletzt besucht im April 2024.

[Cooper2004] Cooper, A.: The Inmates are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity. Pearson Education, 2004.

[Doran1981] Doran, G.T.: There's a S.M.A.R.T. way to write management's goals and objectives. Management Review, 1981. AMA FORUM. 70 (11): 35–36.

[Glinz2014] Glinz, M.: A Glossary of Requirements Engineering Terminology. Standard Glossary for the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version 1.6, 2014. https://www.ireb.org/de/downloads/#cpre-glossary. Zuletzt besucht im April 2024.

[GoWo2006] Gorschek, T., Wohlin, C.: Requirements Abstraction Model. Requirements Engineering Journal, Vol. 11, No. 1, pp. 79–101, 2006.

[HeHe2011] Heath, C., Heath, D.: Switch: Veränderungen wagen und dadurch gewinnen. FISCHER Scherz, 2011.

[Highsmith2001] Highsmith, J.: Design the Box. Agile Project Management E-Mail Advisor 2001, http://www.joelonsoftware.com/articles/JimHighsmithonProductVisi.html. Zuletzt besucht im April 2024.

[IREB2019] IREB e.V.: CPRE Advanced Level Elicitation Syllabus, 2019. https://www.ireb.org/de/downloads/tag:al-e-c#top. Zuletzt besucht im April 2024.

[IREB2017] IREB e.V.: CPRE – RE@Agile Primer – Syllabus and Study Guide, 2017 https://www.ireb.org/de/downloads/tag:re-agile-primer#top. Zuletzt besucht im April 2024.

[ISO25010] ISO/IEC 25010:2011: Systems and software engineering –– Systems and software Quality Requirements and Evaluation. ISO/IEC Standard 25010:2011.

[Jeffries2001] Jeffries, R.: Essential XP: Card, Conversation, Confirmation, 2001, https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/. Zuletzt besucht im April 2024.

[Kahneman2016] Kahneman D.: Schnelles Denken, langsames Denken Penguin Verlag, 2016.

[Kniberg2012] Kniberg, H.: Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds https://blog.crisp.se/2012/11/14/henrikkniberg/scaling-agile-at-spotify. Zuletzt besucht im April 2024.

[LaBai2003] Larman, C., Basili, V. R.: Iterative and Incremental Development: A Brief History. IEEE Computer, Vol 36, No. 6, 2003, 47-56.

[Lamsweerde2009] van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.

[Larman2017] Larman, C.: Large-Scale Scrum: Scrum erfolgreich skalieren mit LeSS. dpunkt.verlag, 2017.

[Leffingwell2010] Leffingwell, D.: Agile Software Requirements – Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison Wesley, 2010.

[LeSS] Large-Scale Scrum: https://less.works/. Zuletzt besucht im April 2024.

[Maalej et al.2016] Maalej, W., Nayebi, M., Johann T., Ruhe, G.: Toward Data-Driven Requirements Engineering. IEEE Software (Volume 33, Issue 1), 2016.

[McConnel2006] McConnel, S.: Software Estimation, Demystifying the Black Art. Microsoft Press, 2006.

[Meyer2014] Meyer, B.: Agile! The Good, the Hype and the Ugly. Springer, 2014.

[NEXUS] Scaling Scrum with Nexus™: https://www.scrum.org/resources/scaling-scrum. Zuletzt besucht im April 2024.

[OsPi2011] Osterwald, A., Pigneur, Y.: Business Model Generation – Ein Handbuch für Visionäre, Spielveränderer und Herausforderer. Campus Verlag, 2011.

[Patton2014] J. Patton, J.: User Story Mapping – Nutzerbedürfnisse besser verstehen als Schlüssel für erfolgreiche Produkte O'Reilly, 2014.

[Pichler2011] Pichler, R.: Product Vision Board, 2011 http://www.romanpichler.com/blog/the-product-vision-board/. Zuletzt besucht im April 2024.

[Pichler2016] Pichler, R.: Strategize: Product Strategy and Product Roadmap Practices for the Digital Age. Pichler Consulting, 2016.

[PoRu2015] Pohl, K., Rupp, C.: Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level. dpunkt.verlag GmbH, 2015.

[Reinertsen2009] Reinertsen, D.G.: The Principles of Product Development Flow – Second Generation Lean Product Development. Celeritas Publishing, 2009.

[SAFe] Scaled Agile Framework 4.5 ® http://www.scaledagileframework.com/. Zuletzt besucht im April 2024.

[S@S] Scrum at Scale™: https://www.scruminc.com/scrum-scale-case-modularity/. Zuletzt besucht im April 2024.

[Scrumguide] The Scrum Guide ™ : http://www.scrumguides.org/scrum-guide. Zuletzt besucht im April 2024.

[SofS] Scrum of Scrums , zuletzt besucht im April 2024

[Spotify2012] Scaling Agile @ Spotify https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf. Zuletzt besucht im April 2024.

[Sterling2012] Sterling C.: Affinity Estimating – A How-To https://scrumology.com/guest-post-affinity-estimating-a-how-to/. Zuletzt besucht im April 2024.

[RoRo2012] Robertson J., Robertson S.: Mastering the Requirements Process – Getting Requirements Right, 3rd edition. Addison Wesley, 2012.

[Robertson2003] Robertson, S.: Stakeholders, Goals, Scope: The Foundation for Requirements and Business Models, 2003, https://www.volere.org/wp-content/uploads/2018/12/StkGoalsScope.pdf. Zuletzt besucht im April 2024.

[Wake2003] Wake, B.: INVEST in Good Stories, and SMART Tasks, 2003, https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Zuletzt besucht im April 2024.

[Weerd et al.2006] van de Weerd, I., Brinkkemper, S., Nieuwenhuis R., Versendaal, J., Bijlsma, L.: On the Creation of a Reference Framework for Software Product Management: Validation and Tools Support. International Workshop on Software Product Management (IWSPM 2006).