Certified Professional for Requirements Engineering

RE@Agile Primer

Lehrplan und Studienleitfaden

Lars Baumann, Peter Hruschka,
Kim Lauenroth, Markus Meuten,
Sacha Reis, Gareth Rogers,
François Salazar,
Hans-Jörg Steffe, Thorsten Weyer

v1.5.0 | 1. Juni 2026

Nutzungsbedingungen

1. Einzelpersonen und Seminaranbieter dürfen den Lehrplan und Studienleitfaden als Grundlage für Seminare verwenden, sofern die Inhaber der Urheberrechte als Quelle und Besitzer des Urheberrechts anerkannt und benannt werden. Des Weiteren dürfen der Lehrplan und Studienleitfaden zu Werbungszwecken nur mit Einwilligung des IREB e. V. verwendet werden.
2. Jede Einzelperson oder Gruppe von Einzelpersonen darf den Lehrplan und Studienleitfaden 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 und Studienleitfaden wurde verfasst von: Lars Baumann, Peter Hruschka, Kim Lauenroth, Markus Meuten, Sacha Reis, Gareth Rogers, François Salazar, Hans-Jörg Steffe, Thorsten Weyer.

Das Dokument wurde überprüft und kommentiert von: Bernd Aschauer, Thomas Emmerich, Dirk Fritsch, Rainer Grau, Andrea Herrmann, Krystian Kaczor, Niko Kaintantzis, Elisabeth Larson, Ladislau Szilagyi, Daniel Tobler, Erik van Veenendaal, Arun Vetrivel, Sven van der Zee.

English Review durch Joy Beatty, Gareth Rogers und Candase Hokanson.

Allen sei für ihr Engagement gedankt.

Die Freigabe der englischen Version wurde am wurde am 2. März 2017 durch das IREB Council auf Empfehlung von Xavier Franch genehmigt.

Zweck des Dokuments

Dieser Lehrplan und Studienleitfaden definiert die Basisstufe (Foundation Level) des Zertifikats „RE@Agile" des International Requirements Engineering Board (IREB). Der Lehrplan und Studienleitfaden dient den Ausbildungsanbietern als Grundlage für die Erstellung ihrer Kursunterlagen. Die Lernenden können sich mit diesem Lehrplan und Studienleitfaden auf die Prüfung vorbereiten.

Inhalt von Lehrplan und Studienleitfaden

Die Basisstufe spricht alle Personen an, die sich mit den Themen Requirements Engineering und agile Entwicklung befassen. Dies schließt Personen aus dem Projekt- oder IT-Management, Fachexperten, Systemanalytiker und Softwareentwickler, Scrum Master, Product Owner sowie Personen die Teil von agilen Organisationen sind, mit ein.

Inhaltsabgrenzung

Die Inspiration zu RE@Agile kommt durch die Sicht von IREB auf agile Werte sowie durch eine agile Sicht auf die Werte des Requirements Engineering. Zum Inhalt gehören Klassifizierung und Beurteilung von Requirements-Engineering-Arbeitsprodukten und -Techniken im Zusammenhang mit Agilität, agilen Arbeitsprodukten und Techniken, Requirements Engineering und wesentlichen Prozesselementen in der agilen Produktentwicklung. RE@Agile zeigt die Motivation für die Verwendung agiler Methoden in Entwicklungsprozessen auf.

Die Synergie zwischen Requirements Engineering und Agilität ist ein sehr wichtiges Thema: Agile Prinzipien in Bezug auf Requirements Engineering und agile Denkweisen in Bezug zu den zentralen Requirements-Engineering-Werten.

Ziel der RE@Agile Zertifizierungen von IREB ist die Unterstützung der folgenden Personengruppen:

- Requirements Engineers, die sich mit agiler Entwicklung befassen und ihre Techniken in dieser Umgebung erfolgreich anwenden möchten.
- Requirements Engineers, die etablierte Konzepte und Techniken agiler Ansätze anwenden und ihre Requirements-Engineering-Prozesse verbessern möchten.
- Fachkräfte für agile Entwicklungsprozesse, die die Werte und Vorteile des Requirements Engineering in agilen Projekten verstehen möchten.
- Fachkräfte für agile Entwicklungsprozesse, welche die agile Entwicklung durch bewährte Requirements-Engineering-Techniken und -Methoden verbessern möchten.
- Personen aus verwandten Disziplinen – IT-Manager, Tester, Entwickler, Architekten und andere Vertreter im Bereich der Entwicklung (überwiegend, aber nicht ausschließlich Softwareentwicklung), die verstehen möchten, wie sie Requirements-Engineering- und agile Ansätze in Entwicklungsprozessen erfolgreich kombinieren können.

Detaillierungsgrad

Der Detaillierungsgrad dieses Lehrplans und Studienleitfadens ermöglicht konsistentes Lehren und Prüfen auf internationaler Ebene. Um dieses Ziel zu erreichen, beinhaltet der Lehrplan und Studienleitfaden Folgendes:

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

Geschlechtsspezifische Formulierungen

Wir haben in diesem Dokument bewusst auf geschlechtsspezifische Formulierungen verzichtet.

Selbstverständlich unterstützen wir als IREB geschlechtssensible Formulierungen. Wir sehen aber auch die Notwendigkeit, komplexe Sachverhalte so zu formulieren, dass sie leicht verstanden werden können.

Texte, die eigentlich eine männliche und weibliche Form fordern, wären weniger gut lesbar und damit schwieriger zu verstehen. Ziel dieses Dokuments ist es jedoch, Inhalte präzise und klar darzustellen und zu vermitteln. Da wir dem Leser helfen wollen, den Fokus auf den Inhalt zu richten, verwenden wir in diesem Dokument bewusst nur die männliche Form von Personen.

Dies soll nicht als Ausdruck von mangelndem Respekt verstanden werden.

Lernziele/Kognitive Stufen des Wissens

Jeder Abschnitt dieses Lehrplans und Studienleitfadens ist einer kognitiven Stufe zugeordnet. Eine höhere Stufe umfasst die niedrigeren Stufen. In den Formulierungen der Lernziele werden für die Stufe K1 das Verb „kennen" und für die Stufe K2 die Verben „können und anwenden" verwendet. Diese Verben werden stellvertretend für die nachfolgend aufgelisteten Verben der gleichen Stufe verwendet:

Allen Modulen und Lernzielen in diesem Lehrplan ist eine kognitive Stufe zugeordnet. Die folgenden Stufen werden verwendet:

- **K1: Kennen** (beschreiben, aufzählen, charakterisieren, erkennen, benennen, erinnern, ...) – Sich an zuvor gelernten Stoff erinnern oder ihn abrufen.
- **K2: Verstehen** (erklären, interpretieren, vervollständigen, zusammenfassen, begründen, klassifizieren, vergleichen, ...) — Bedeutung anhand von gegebenem Inhalt oder Situationen begreifen/herstellen.
- **K3: Anwenden** (spezifizieren, schreiben, entwerfen, entwickeln, implementieren, ...) – Wissen und Fähigkeiten in gegebenen Situationen anwenden.

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 Begriffe, die im CPRE Glossar und SCRUM Guide genannt werden, sind zu kennen (K1), auch wenn sie in den Lernzielen nicht explizit genannt sind.

Das CPRE Glossar steht auf der IREB Homepage zum Download zur Verfügung: https://cpre.ireb.org/de/downloads-and-resources/downloads#cpre-glossary.

Der SCRUM Guide steht zum Download zur Verfügung unter: https://scrumguides.org/download.html

Im Lehrplan und Studienleitfaden wird die Abkürzung „RE" für Requirements Engineering verwendet.

Struktur des Lehrplans und Studienleitfadens

Der Lehrplan und Studienleitfaden besteht aus 4 Hauptkapiteln. Ein Kapitel umfasst eine Lerneinheit (LE). Jeder Haupttitel eines Kapitels beinhaltet die kognitive Stufe des Kapitels, das ist die höchste Stufe der Teilkapitel. Weiterhin werden die Unterrichtszeiten genannt, welche in einem Kurs mindestens für dieses Kapitel aufgewendet werden sollten. Wichtige Begriffe des Kapitels, die im Glossar definiert sind, sind am Anfang des Kapitels aufgelistet.

Beispiel: Kapitel 2 Grundlagen von RE@Agile (K1)
Dauer: 1,25 Stunden
Begriffe: Product Owner, Product Backlog, Sprint Backlog, Epics, User-Stories, Story-Maps

Das Beispiel zeigt, dass in Kapitel 2 Lernziele der Stufe K1 enthalten sind und 75 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 3.1.2
Das Beispiel zeigt, dass das Lernziel LZ 3.1.2 im Unterkapitel 3.1. beschrieben wird.

Die Prüfung

Dieser Lehrplan und Studienleitfaden bildet die Grundlage für die Prüfung zum RE@Agile Primer. Es gibt zwei verschiedene Prüfungen:

- Eine Multiple-Choice-Prüfung von 40 Minuten Dauer unter Aufsicht, die mit einem offiziellen RE@Agile Primer Zertifikat abschließt; diese Prüfung ist mit den Multiple-Choice-Prüfungen für CPRE Foundation Level und CPRE Modul RE@Agile Practitioner vergleichbar.
- Multiple-Choice-Selbstevaluation, die online durchgeführt werden kann und die mit einer Teilnahmebestätigung abschließt.

Prüfungen unter Aufsicht können unmittelbar im Anschluss an einen Kurs, aber auch unabhängig davon (z. B. in einem Prüfzentrum) abgelegt werden. Die von IREB anerkannten Prüfungsanbieter sind auf der IREB Homepage unter https://ireb.org/de/cooperations/certification-bodies aufgeführt.

Die Selbstevaluation wird auf der IREB Homepage zur Verfügung gestellt: https://cpre.ireb.org/de/training-certification/online-self-assesment

Eine Prüfungsfrage kann Stoff aus mehreren Kapiteln des Lehrplans und Studienleitfadens enthalten. Alle Abschnitte (LE 1 bis LE 4) dieses Lehrplans und Studienleitfadens können geprüft werden.

Versions-Historie

Version	Datum	Kommentar
1.0.0	17. Juli 2017	Release-Version, übersetzt aus der englischen Version 1.0
1.0.1	17. August 2017	Typos fixed
1.0.2	16. November 2017	Glossar als eigenständiges Dokument extrahiert. Siehe https://www.ireb.org/de/downloads/#re-agile-glossary
1.1.0	24. September 2020	Detaillierte Informationen über den Produkt-Backlog hinzugefügt; einige Rechtschreib- und Grammatikprobleme behoben. konsequent"Minimum" verwendet (z.B. Minimum Viable Product
1.2.0	1. Januar 2023	▪ Bugfixes ▪ Aktualisierung an den Scrumguide 2020 ▪ Synchronisierung mit dem CPRE Foundation Level Version 33 sowie dem RE@Agile Advanced Level 2022 ▪ Anpassung der kognitiven Stufen an die neue IREB Definition ▪ Anpassung der Lernziele an die kognitiven Stufen ▪ Anpassung der empfohlenen Zeiten der einzelnen Kapitel an die kognitiven Stufen
1.3.0	September 2023	Kleinere Bugfixes
1.4.0	1. Juli 2024	Fehlerhafte Referenzen korrigiert, Schreibweise von Begriffen an das Glossar angepasst, Manifest für agile Softwareentwicklung an RE@Agile Practitioner angeglichen, Überschrift Kapitel 2.7 auf Deutsch, neues Corporate Design umgesetzt
1.5.0	1. Juni 2026	Update wegen Wegfall des RE@Agile Glossar (Aktualisierung des CPRE Glossar) und Aktualisierung der Literaturreferenzen.

Inhalt

Die Vision von RE@Agile

Motivation und Hintergrund

Die Qualität der Anforderungen entscheidet über Erfolg oder Scheitern des gesamten Produktentwicklungsprozesses, unabhängig von der angewandten Entwicklungsmethodologie. Entgegen weit verbreiteter Meinung sind die Techniken und Methoden der Requirements-Engineering-Disziplin in ihrer Verwendung innerhalb bestimmter Entwicklungsmethodologien (etwa wie das Wasserfall-Modell oder Scrum) neutral. Das Requirements Engineering wird jedoch meist als nicht-agile Entwicklungsdisziplin aufgefasst, was zu dem Missverständnis führt, dass der Wissensfundus des Requirements Engineering für den Erfolg agiler Entwicklungsprozesse nicht relevant ist.

In vielen Fällen werden Requirements-Engineering- und agile Ansätze separat und nicht zusammen betrachtet. Während in konventionellen Entwicklungsprozessen das Requirements Engineering mit dedizierten Rollen als eine eigene Disziplin im Lebenszyklus eines Systems erarbeitet wird, unterschätzt man die Bedeutung des Requirements Engineering in der agilen Entwicklung häufig.

Agile Ansätze beruhen auf direkter Kommunikation, der Einfachheit von Lösungen und Feedback. Die schnelle Reaktion auf Veränderungen stellt einen der wichtigsten Werte dar. Somit bilden Änderungen von Anforderungen und deren Prioritäten ein inhärentes Konzept aller agilen Ansätze. Misst man der Requirements-Engineering-Kompetenz in agilen Entwicklungsprozessen einen angemessenen Stellenwert bei, kann dies den Erfolg agiler Projekte tatsächlich steigern und gleichzeitig die Qualität entwickelter Systeme und Produkte nachhaltig verbessern. Umgekehrt kann die Requirements-Engineering-Praxis, unabhängig von der im Einzelfall angewandten Entwicklungsmethodologie, erheblich von einigen sehr nützlichen agilen Prinzipien und Techniken profitieren.

Man trifft heute oftmals auf Experten, die entweder im Requirements Engineering oder in der Anwendung bestimmter agiler Ansätze sachkundig sind. Diese Experten müssen selbst einen Weg finden, um sich die Vorzüge des Einsatzes von Prinzipien und Techniken aus dem jeweils anderen Kompetenzbereich nutzbar zu machen. Zertifizierungen, die auf eine Integration beider Kompetenzfelder ausgerichtet sind, gab es bisher nicht, weder von der Requirements-Engineering- noch von der Agile-Community. Daher ist es äußerst vielversprechend, eine weithin anerkannte Brücke zwischen Requirements-Engineering- und agilen Ansätzen, und damit auch zwischen Requirements Engineers und Experten für agile Entwicklungsprozesse zu schlagen, damit beide Seiten effizient und effektiv miteinander kommunizieren können.

IREBs Antwort auf diesen Bedarf ist die RE@Agile Zertifizierung.

Informationen zu RE@Agile

Die Brücke zwischen Requirements Engineering- und agilen Ansätzen sollte aus zwei unterschiedlichen Richtungen gebaut werden: Einerseits muss die Requirements-Engineering-Community verstehen, wie sie ihre verschiedenen Techniken und Methoden in agilen Entwicklungsprozessen erfolgreich einsetzen und spezifische Techniken aus agilen Ansätzen anwenden kann, um die Requirements-Engineering-Praxis zu verbessern. Andererseits, da agile Ansätze auf eine möglichst frühzeitige Auslieferung werthaltiger Software abzielen, müssen die Experten für agile Entwicklungsprozesse verstehen, wie sie diesen Effekt nutzen können, indem sie bewährte Konzepte und Techniken aus der Disziplin des Requirements Engineering (RE) anwenden.

Die zentralen Prinzipien von RE@Agile sind:

- *Requirements-Engineering- und agile Ansätze können voneinander profitieren*
  Dazu greift RE@Agile zum einen die Verwendung von Arbeitsprodukten und Techniken aus der Disziplin des Requirements Engineering in agilen Prozessen auf. Zum anderen beinhaltet es die Nutzung von Arbeitsprodukten, Rollen und Techniken aus agilen Ansätzen in Requirements-Engineering-Prozessen im Kontext verschiedener Entwicklungsmethodologien.
- *Schlanke und in hohem Maße adaptive Prozesse*
  Basierend auf der Philosophie von RE@Agile ist die Differenzierung zwischen prädiktiven und adaptiven Entwicklungsprozessen von entscheidender Bedeutung. RE@Agile verfolgt die Idee eines schlanken und hochgradig adaptiven Ansatzes für die Durchführung von Requirements-Engineering-Aktivitäten im Rahmen agiler Entwicklungen. Das Requirements Engineering stellt in RE@Agile mehr eine grundlegende Disziplin als einen einfachen Prozessschritt dar: einen kontinuierlichen Prozess, der systematisch durchgeführt werden muss und der ein hohes Maß an Kompetenz und Erfahrung erfordert.
- *Enge Zusammenarbeit mit dem Team und mit zentralen Stakeholdern und „Just-in-Time-Anforderungen"*
  Für den Erfolg agiler Entwicklungsprozesse ist ein häufiger Austausch und die enge Zusammenarbeit zwischen allen Teammitgliedern und für die Gestaltung des Produktes wichtigen Stakeholdern von besonderer Bedeutung. Im Rahmen von RE@Agile ermittelt, analysiert, verfeinert und dokumentiert das Team gemeinsam mit zentralen Stakeholdern die Anforderungen auf hochgradig interaktive Art und Weise. RE@Agile unterstützt Experten in der Praxis beim Auswählen der geeigneten Aktivitäten zum richtigen Zeitpunkt, um qualitativ hochwertige Anforderungen sicherzustellen, bevor diese implementiert werden.
- *Situative und selektive Anforderungsermittlung, Analyse, Spezifikation und Verfeinerung*
  RE@Agile stützt sich auf die Idee, dass nicht jede Anforderung präzise und in einem hohen Detaillierungsgrad spezifiziert werden muss, bevor die Implementierung eines Systems beginnen kann. Vielmehr werden nur solche Anforderungen verfeinert und detaillierter spezifiziert, die übermäßig komplex (d. h. für Stakeholder oder das Entwicklungsteam nicht verständlich) oder kritisch (d. h. bei denen kein

Missverständnis riskiert werden kann) sind. Dem Gesamtprozess liegt die gemeinsame Philosophie zugrunde, dass Änderungen an funktionalen Anforderungen erwünscht und leicht integrierbar sind.

- *Weniger relevante Aktivitäten und Funktionalitäten vermeiden und das Minimum Viable Product sicherstellen*
  Ein Prinzip von Agilität ist „Einfachheit". Nach diesem Prinzip ist die erste Phase einer System- oder Produktentwicklung in agilen Prozessen häufig das MVP (Minimum Viable Product). Das MVP ist ein klar abgegrenztes, bereitstellbares System, das lediglich grundlegende Merkmale aufweist, es bietet jedoch einen hinreichenden betriebswirtschaftlichen Wert für Endbenutzer, um validiertes Lernen zu ermöglichen. Durch den minimalen Umfang eines MVP kann Überflüssiges bei der Entwicklung vermieden und schnelleres Kundenfeedback eingeholt werden. Eine der nächsten Produktphasen umfasst oftmals das MMP (Minimum Marketable Product) – das Produkt mit dem minimal notwendigen Funktionsumfang für die Nutzung und Vermarktung, welches den Benutzerbedürfnissen Rechnung trägt und dadurch einen Marktwert besitzt. RE@Agile liefert Antworten auf zwei sehr wichtige Fragen, die selbst in nicht-agilen Entwicklungsprozessen eine ganz wesentliche Rolle spielen: „Wie lassen sich das Release-Management und der Produktdefinitionsprozess vereinfachen?" und „Wie definiert man das MVP oder das MMP auf Basis der Anforderungen?"

Informationen zur IREB RE@Agile Zertifizierung:

- Als Vorkenntnisse werden Kenntnisse des CPRE Foundation Levels und über agile Entwicklungsprozesse empfohlen.
- Das RE@Agile Primer Zertifikat wurde für Experten aus verwandten Disziplinen konzipiert: Projektmanager, Business-Analysten, Architekten, Entwickler, Tester sowie Fachexperten. Bei diesem Zertifikat liegt der Schwerpunkt auf der Kommunikation zwischen Experten aus dem Requirements Engineering und aus agilen Entwicklungsprozessen sowie auf dem Verständnis der Begrifflichkeiten aus beiden Gebieten. Zertifikatsinhaber können sich mit Experten in agilen Entwicklungsprozessen über Requirements Engineering austauschen und mit Requirements-Engineering-Experten über agile Methoden und agile Entwicklung.

Eine Person mit einem RE@Agile Primer Zertifikat:

- ist mit der relevanten Terminologie von Requirements-Engineering- und agilen Ansätzen vertraut
- versteht die Rolle und die Bedeutung des Requirements Engineering in agilen Prozessen sowie den Wert von Agilität im Requirements Engineering

1 Motivation und Denkweisen (K1)

Dauer: 1,5 Stunden

Begriffe: Werte, Manifest für agile Softwareentwicklung, Verfahren, Aktivitäten, Sprint, agil/Agilität

Lernziele

LZ 1.1.1 Motivation für die Verwendung agiler Methoden kennen
LZ 1.2.1 Die Ziele des Requirements Engineering gemäß IREB kennen
LZ 1.2.2 Zentrale Werte des Manifests für agile Softwareentwicklung und die daraus abgeleiteten Prinzipien kennen
LZ 1.3.1 Unterschied zwischen Prinzip, Verfahren und Aktivität kennen
LZ 1.3.2 Unterschiede zwischen agilen Denkweisen und Requirements Engineering Denkweisen kennen
LZ 1.3.3 Synergien von Denkweisen und Werten mit Blick auf RE@Agile kennen
LZ 1.3.4 Wissen, was „Dokumentation" in einem agilen Kontext bedeutet (in Übereinstimmung mit dem Manifest für agile Softwareentwicklung)
LZ 1.4.1 Vorteile, Stolpersteine und Missverständnisse bei der Verwendung von RE@Agile kennen
LZ 1.4.2 Beispiele für Missverständnisse kennen
LZ 1.5.1 Wissen, dass agile Werte auf konzeptuelle Arbeit übertragen werden können
LZ 1.5.2 Exemplarische Ansätze kennen, die Agilität in konzeptueller Arbeit ermöglichen

1.1 Motivation für die Verwendung agiler Methoden (K1)

Mehreren Studien zufolge (siehe [MeMi2015]) erfährt das IT-Geschäft insgesamt einen grundlegenden Wandel: Informationstechnologie wird zu einem Haupteinflussfaktor in diversen Geschäftsbereichen (z. B. E-Commerce, soziale Medien) und technischen Gebieten (z. B. Automobil- oder Avionikbranche). Folglich müssen die Systeme und Produkte in IT-orientierten Branchen einer konstanten Anpassung unterzogen werden, damit sie mit den veränderten Bedürfnissen von Kunden oder dem Markt Schritt halten. Sobald sich im Markt eine Veränderung ergibt, müssen die Systeme an diese Änderungen angepasst werden.

Vorhandene Entwicklungsmethoden, die auf langfristige Berechenbarkeit und Stabilität ausgelegt sind, wurden nicht für solche Gegebenheiten entwickelt und versagen häufig in sich schnell ändernden Geschäftsumgebungen oder Projektsituationen. Durch das Manifest für agile Softwareentwicklung (siehe 1.2) geprägte agile Methoden schließen nun diese Lücke. Der Begriff „agil" (oder Agilität) ist schwierig zu fassen, kann aber wie folgt definiert werden (siehe [ShYo2006]):

> Eine schnelle Ganzkörperbewegung mit Geschwindigkeits- oder Richtungsänderung als Reaktion auf einen Impuls.

Diese Definition stammt nicht ursprünglich aus dem Software-Engineering, sondern aus dem Sport, doch sie gibt die eigentliche Motivation für die Verwendung einer agilen Methode wieder: Agile Methoden eignen sich, wenn der Markt oder eine Projektsituation schnelle und

kontrollierte Änderungen erfordert. Eine agile Methode ist selbstverständlich mehr als nur schnelle Entwicklung (siehe 1.2), aber im Wesentlichen konzentrieren sich alle Prinzipien letztlich auf häufige Auslieferungen in einer vorgegebenen Qualität. Dadurch entstehen häufige Feedbackzyklen, was wiederum ein schnelle Anpassung an die Kundenbedürfnisse erlaubt.

Daher muss beachtet werden, dass weder agile Methoden noch Agilität ein Selbstzweck sind [Meyer2014]. Ein Unternehmen muss in der Lage sein, den richtigen Entwicklungsansatz auszuwählen, der zu den Bedürfnissen des Marktes, der Kunden und der Unternehmen passt. Gartner konstatiert sogar, dass die Fähigkeit, IT unter Verwendung des richtigen Ansatzes zu entwickeln, der entscheidende Erfolgsfaktor für das digitale Geschäft ist [MeMi2015].

1.2 Denkweisen und Werte im RE und in agilen Entwicklungsprozessen (K1)

Die Denkweisen und Werte des RE sind in der IREB-Definition von Requirements Engineering (siehe [Glinz2026]) erläutert: The systematic and disciplined approach to the specification and management of requirements with the goal of understanding the stakeholders' desires and needs and minimizing the risk of delivering a system that does not meet these desires and needs.

Im RE sprechen wir von einem System anstelle von einer Software oder einem Produkt. Die Verwendung des Begriffs „System" soll Produkte, andere Arten von Software, oder gar andere Dinge (z. B. Geschäftsprozesse oder Hardware) nicht ausschließen. Im RE wird der Begriff „System" bevorzugt, da er die Tatsache betont, dass ein System eine Gruppe von Bestandteilen oder Elementen ist, die zusammen in einer Umgebung funktionieren. Die Umgebung bezeichnet man im RE als Systemkontext. In diesem Lehrplan und Studienleitfaden verwenden wir durchgängig den Begriff „System", der Produkte sowie alle anderen Arten softwarebezogener Elemente einschließen soll.

Der IREB Foundation Level [CPREFL2022] definiert zudem vier Hauptaktivitäten des RE: Ermittlung, Dokumentation, Prüfung (Validierung)/Abstimmung und Verwaltung von Anforderungen. Diese Liste von Aktivitäten steht nicht für einen bestimmten Umfang von Schritten oder die Reihenfolge, in der diese Aktivitäten ausgeführt werden. Ein zentraler Wert des IREB Foundation Levels ist, dass das RE ein prozessunabhängiger Ansatz ist: Das RE stellt einen umfassenden Wissensfundus aus verschiedenen Methoden bereit sowie eine große Sammlung von Techniken, die in jedem Entwicklungsansatz angewendet werden können. RE verzichtet auf die Empfehlung oder Spezifizierung eines bestimmten Prozesses.

In diesem Lehrplan und Studienleitfaden wird der Begriff „agile Methoden" zur Bezeichnung der vielfältigen Ansätze aus dem Bereich Agilität verwendet (siehe 2). Um agile Methoden von anderen Entwicklungsmethoden (etwa der plangesteuerten Methode oder Wasserfall-Methode) zu unterscheiden, wird in diesem Lehrplan und Studienleitfaden der Begriff „nicht-agile Methoden" verwendet. Diese beiden Begriffe ermöglichen eine Evaluierung der besten Methode – IREB ist der Überzeugung, dass beide Ansätze (agile und nicht-agile) ihren Wert haben.

Die Denkweise von Agilität wird durch das Manifest für agile Softwareentwicklung und die zwölf zugrundeliegenden Prinzipien definiert (siehe [AgileMan2001]):

> **Das Manifest für agile Softwareentwicklung**
>
> Wir erschließen bessere Wege, Software zu entwickeln,
> indem wir es selbst tun und anderen dabei helfen.
>
> Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:
> **Individuen und Interaktionen** mehr als Prozesse und Werkzeuge
> **Funktionierende Software** mehr als umfassende Dokumentation
> **Zusammenarbeit mit dem Kunden** mehr als Vertragsverhandlung
> **Reagieren auf Veränderung** mehr als das Befolgen eines Plans
>
> Die Aspekte auf der rechten Seite haben zweifellos ihren Wert,
> wir messen aber den Aspekten auf der linken Seite einen höheren Wert bei.

Agile Prinzipien

1. Die Zufriedenstellung des Kunden durch frühzeitige und kontinuierliche Auslieferung nützlicher Software hat für uns höchste Priorität.
2. Veränderte Anforderungen werden gerne auch noch in späten Entwicklungsphasen berücksichtigt. In agilen Prozessen werden Veränderungen zum Wettbewerbsvorteil des Kunden genutzt.
3. Funktionierende Software wird regelmäßig innerhalb weniger Wochen bis hin zu einigen Monaten ausgeliefert, wobei die kürzere Zeitspanne bevorzugt wird.
4. Stakeholder und Entwickler müssen während des gesamten Projekts täglich zusammenarbeiten.
5. Projekte werden mit motivierten Einzelpersonen aufgebaut. Sie sollen das Umfeld und die benötigte Unterstützung erhalten sowie das Vertrauen, dass sie ihre Aufgabe erledigen werden.
6. Das persönliche Gespräch ist die effizienteste und effektivste Methode, um Informationen an das und innerhalb des Entwicklungsteams zu vermitteln.
7. Funktionierende Software ist das primäre Maß für den Fortschritt.
8. Agile Prozesse fördern kontinuierliche Entwicklung. Die Sponsoren, Entwickler und Benutzer sollten unbegrenzt ein konstantes Tempo halten können.
9. Stetiges Augenmerk auf technischer Exzellenz und gutem Design sorgen für ein höheres Maß an Agilität.
10. Einfachheit – die Kunst, die Menge an nicht getaner Arbeit zu maximieren – ist wesentlich.
11. Die besten Architekturen, Anforderungen und Designs entstehen aus Teams, die sich selbst organisieren.
12. Das Team denkt in regelmäßigen Abständen darüber nach, wie es effizienter werden kann und richtet dann sein Verhalten danach aus.

Wenn wir die Werte und Denkweisen von RE und Agilität vergleichen, sehen wir keinen einzigen Aspekt, der im Widerspruch zu den jeweils anderen Werten oder Denkweisen steht. Der wichtigste Wert ist dem RE und Agilität gemein: die Endbenutzer des Produkts durch eine Lösung zufriedenstellen, die ihren Bedürfnissen entspricht oder die größten Probleme behebt. Dennoch müssen wir anerkennen, dass die Denkweisen und Werte von RE und agilen Entwicklungsprozessen zu einem gewissen Grad unzusammenhängend sind. Die Schnittmenge von RE und Agilität wird durch RE@Agile definiert, das in der nächsten LE näher erläutert wird.

1.3 RE@Agile - die Brücke zwischen RE- und agilen Prinzipien (K1)

Bevor wir uns mit den Einzelheiten von RE@Agile befassen, möchten wir auf die Terminologie eingehen. In der Softwareindustrie und -forschung gibt es ein umfangreiches Wissen zu Arbeits- und Verhaltensweisen bei der Entwicklung von Software. Dieses Wissen ist auf mehreren Abstraktionsebenen verfügbar. Nachfolgend stellen wir die Differenzierung nach Prinzipien, Verfahren und Aktivitäten als den drei Abstraktionsebenen vor, auf welchen man über die Entwicklung von Software sprechen kann (siehe [Meyer2014]):

- Ein Prinzip ist eine präskriptive Aussage, die abstrakt und falsifizierbar ist.
- Ein Verfahren/eine Technik ist eine Instanziierung eines Prinzips in einem bestimmten Kontext.
- Eine Aktivität ist eine reale oder geplante Ausführung eines Verfahrens.

Die wichtigen Begriffe zur Differenzierung der drei Definitionen sind präskriptiv, abstrakt und falsifizierbar. Präskriptiv bedeutet, dass die Aussage eine Aktion steuert, anstatt eine Tatsache oder eine Eigenschaft darzulegen. Ein Prinzip unterscheidet sich von einem Verfahren durch Abstraktheit. Ein Beispiel: „Eine Softwarefunktion vor der Auslieferung testen" ist präskriptiv, wohingegen „einen Unit-Test für jedes Softwarefeature erstellen" ein Verfahren basierend auf dem vorgegebenen Prinzip ist. Eine Aktivität zu diesem Beispiel wäre die Erstellung eines Unit-Tests für die Suchfunktion eines Bibliotheksystems. Falsifizierbarkeit bedeutet, dass eine Person mit hinreichendem Hintergrundwissen eine andere Meinung zu einem Prinzip haben kann. Das oben angeführte Prinzip („Eine Softwarefunktion...testen") erfüllt dieses Kriterium. Man könnte argumentieren, dass das Testen für sicherheitsrelevante Features unter Umständen nicht ausreichend ist und diese stattdessen anhand mathematischer Hilfsmittel geprüft werden sollten. Eine Aussage, die nicht falsifizierbar ist („nach hoher Qualität streben"), sollte nicht als Prinzip für Verhaltensrichtlinien betrachtet werden.

Das Wissen um die Prinzipien, die unseren Verfahren zugrunde liegen (was wiederum ein Prinzip ist), führt zu bewussten Entscheidungen in Bezug auf unsere Aktionen. Das Kennen verschiedener Verfahren zur Erfüllung von Prinzipien ermöglicht uns, situationsgerecht zu reagieren. Falsifizierbarkeit fördert Gespräche über die Anwendbarkeit eines Prinzips oder eines Verfahrens in einem bestimmten Kontext und vereinfacht so die Entscheidung, ob ein Prinzip angewendet werden soll oder nicht.

Das Kennen von Prinzipien und Verfahren ist nur ein erster Schritt: Die richtige Anwendung eines Verfahrens ist eine eigene Kompetenz. Die Definition von Anwendungsfällen ist beispielsweise ein weithin bekanntes Verfahren zu dem Prinzip „funktionale Anforderungen für wichtige Features spezifizieren". Das Verfassen eines qualitativ hochwertigen Anwendungsfalls ist hingegen eine eigenständige Kompetenz, das reine Kennen einer Anwendungsfallvorlage ist dafür nicht ausreichend.

Wir haben im Wesentlichen drei Kompetenzstufen definiert:

1. Das Kennen von Prinzipien (entspricht der kognitiven Stufe K1)
2. Das Verstehen von Verfahren zum Erfüllen der vorgegebenen Prinzipien (entspricht der kognitiven Stufe K2)
3. Das Anwenden eines Verfahrens im Rahmen eines bestimmten Kontexts (Fähigkeit zum Durchführen einer Aktivität mit hoher Qualität) (entspricht der kognitiven Stufe K3)

Vergleicht man die Disziplin Agilität mit der Disziplin RE (siehe 1.2), so erkennt man, weshalb die beiden gelegentlich als widersprüchlich wahrgenommen werden: Das RE beschäftigt sich mit der systematischen Ermittlung und Dokumentation von Anforderungen als eigenständigen Arbeitsprodukten, während bei agilen Entwicklungsprozessen die funktionierende Software im Vordergrund steht und nicht eine umfassende Dokumentation; auch werden Personen und Kommunikation höher bewertet als Prozesse und Werkzeuge.

Eine überzogene Umsetzung beider Denkweisen kann in der Praxis zu einem Konflikt führen: Eine falsche Auslegung des RE ist etwa, dass es möglich sei, ein vollständiges, konsistentes und abgestimmtes Anforderungsdokument zu erstellen, das sich ohne weitere Änderungen implementieren lässt. Eine ähnlich falsche Interpretation von Agilität ist es, dass ein Entwicklungsprojekt ohne jegliche vorbereitenden Tätigkeiten begonnen werden kann, und dieses ausschließlich dadurch erfolgreich ist, dass Software in regelmäßigen Intervallen ausgeliefert, von Stakeholdern [Anmerkung: Aus RE-Sicht sind Kunden eine Teilmenge der Stakeholder.] geprüft und basierend auf dem Feedback verbessert wird.

Wir behaupten, dass die Denkweisen des RE und der Agilität faktisch nicht im Widerspruch zueinander stehen: Beide Ansätze verfolgen dasselbe Ziel der Auslieferung von Software auf einem klar definierten Qualitätsniveau. Dank agiler Methoden kann funktionierende Software effizient und schnell ausgeliefert werden (geringere Zykluszeit). Das RE stellt geeignete Techniken bereit, mit deren Hilfe ein Verständnis der Wünsche und Anforderungen von Stakeholdern erlangt und die Software mit dem größten Gewinn (Zuwachs an Business-Wert oder Linderung akuter Probleme) entwickelt werden kann.

Das RE vereinfacht Folgendes:

- Das für die Entwicklung nützlicher Software relevante Verständnis der Wünsche und Anforderungen von Benutzern zu erlangen (1. Agiles Prinzip)
- Geeignete Werkzeuge, um Veränderungen im Markt zu erkennen und zum Wettbewerbsvorteil der Stakeholder zu nutzen (2. Agiles Prinzip)
- Passende Werkzeuge und Techniken zur Förderung einer effizienten Zusammenarbeit zwischen Stakeholdern und Entwicklern (4. Agiles Prinzip)

- Geeignete Werkzeuge und Techniken zur Unterstützung verbaler Kommunikation (6. Agiles Prinzip)
- Das für die Minimierung der Entwicklung unnötiger Software relevante Verständnis der Wünsche und Anforderungen von Stakeholdern zu erlangen (10. Agiles Prinzip)

Ein wichtiger Unterschied zwischen der Anwendung von RE in agilen Entwicklungsprozessen und anderen Entwicklungsmethoden ist die zeitliche Abstimmung und der angewandte Prozess. Mit diesem Lehrplan und Studienleitfaden definiert IREB das Gebiet RE@Agile und legt dar, wie das RE im Kontext agiler Methoden anzuwenden ist.

IREB bevorzugt den Begriff RE@Agile gegenüber „Agiles Requirements Engineering", um zu verdeutlichen, dass das RE prozessunabhängig ist.

Das Manifest für agile Softwareentwicklung hebt den Wert eines Inkrements funktionierender Software (oder eines funktionierenden Produkts) gegenüber einer umfassenden Dokumentation hervor. Eine übertriebene Interpretation hatte den falschen Eindruck zur Folge, dass agile Methoden gänzlich auf Dokumentation verzichten. Diese Interpretation ist falsch: Zweckmäßige Dokumentation ist in agilen Entwicklungsprozessen nach wie vor erwünscht und zu empfehlen, allerdings nur dann, wenn sie die Entwicklung unterstützt oder Bestandteil des Produkts ist. In der Vergangenheit wurde es als Problem empfunden, dass in vielen Projekten Dokumentation ohne klar definierten Zweck oder Mehrwert erstellt wurde – diese Art von Dokumentation ist nach den agilen Prinzipien zu vermeiden.

1.4 Vorteile, Missverständnisse und Stolpersteine bei der Verwendung von RE@Agile (K1)

RE@Agile bietet eine Reihe von Vorteilen. Diese Vorteile haben jedoch auch einen Preis: damit verbunden sind potenzielle Missverständnisse und Stolpersteine, die es zu vermeiden gilt.

1.4.1 Vorteile von RE@Agile

RE- und Entwicklungskompetenzen im selben Team können den Aufwand für Übergaben verringern: Arbeitet man in der Praxis mit cross-funktionalen Teams und agilen Methoden, dann muss das Team über alle Kompetenzen verfügen, die es zur Entwicklung eines Produktinkrements basierend auf den ausgewählten Anforderungen benötigt.

Wenn RE-Aufgaben im Team ausgeführt werden, kann das den Bedarf an vorab erstellter, umfassender Anforderungsdokumentation verringern, da sich die Teammitglieder bestimmte Details direkt gegenseitig erklären können. Der Vorteil von Dokumenten in diesem Kontext ist die Dokumentation von Ergebnissen aus den geführten Gesprächen und die Bewahrung von Wissen.

Inkrementelle Entwicklung ermöglicht die Optimierung vorhandener Ideen: Das zentrale Prinzip agiler Methoden ist die inkrementelle Entwicklung von Software in Iterationen. Durch den iterativen Prozess entstehen Arbeitsprodukte (z. B. ein Geschäftsprozessmodell, ein Epic, eine User Story, ein Anwendungsfall, ein UI-Prototyp, eine Prozessbeschreibung oder

die Software), und er verbessert diese Arbeitsprodukte in einer Folge von Entwicklungs- und Review-Aktivitäten. Der Vorteil eines solchen Verfahrens ist, dass die Qualität der Arbeitsprodukte und/oder der Software kontinuierlich verbessert und optimiert wird. Außerdem ermöglichen Inkremente geringeren Umfangs frühzeitige Gespräche mit dem Kunden, und sie minimieren das Risiko, dass zwischen den Erwartungen des Kunden und der Entwicklung große Lücken entstehen.

Verfeinerung ist ein Prinzip zur Ausreifung und Validierung von Anforderungen: In der agilen Entwicklung wurde ein sehr gutes Verfahren entwickelt – die kontinuierliche Verfeinerung. Zu diesem Zweck werden regelmäßig Verfeinerungsmeetings für das Entwicklungsteam angesetzt, um Anforderungen – in enger Absprache mit den Stakeholdern – kontinuierlich durchzusehen und zu detaillieren. Zudem wird das bewährte Verfahren der Definition of Ready als Quality Gate genutzt, um zu validieren, dass eine Anforderung in einer nachfolgenden Iteration zur Implementierung bereit ist.

Das RE vereinfacht das Definieren eines ersten Product Backlogs: Das RE bietet verschiedene Techniken, um die Anforderungen der Stakeholder für das gewünschte Produkt richtig zu verstehen. Es ermöglicht dadurch ein tiefer gehendes Verständnis der Anforderungen, die für die erste Definition des Product Backlog benötigt werden. Es ist wichtig zu erkennen, dass sich mit einer solchen RE-Aktivität keine detaillierte Spezifikation erstellen lässt. Das Ziel dieser Aktivitäten ist vielmehr die Konzentration auf ein umfassendes Verständnis des Produkts auf einer bestimmten Abstraktionsebene (z. B. das Verstehen und Definieren der eigentlichen Anwendungsfälle oder Epics und User Storys). Zum Erstellen eines ersten Backlogs kann ein komplexer Geschäftsprozess auf hoher Ebene mithilfe etablierter RE-Methoden zum Beispiel in Epics, Features und User Storys zerlegt werden [CPREALAGILE2022].

1.4.2 Missverständnisse in Bezug auf RE@Agile

In der Welt der Entwicklung (besonders in der Softwareentwicklung) bestehen einige Missverständnisse und Stolpersteine in Zusammenhang mit dem RE, auf die wir nun eingehen:

Missverständnis – das RE ist lediglich eine vorab erstellte Analyse: Das RE wird sehr oft nur als eine mögliche Vorabaktivität betrachtet. Das RE als Disziplin ist hingegen faktisch prozessunabhängig und erfordert nicht zwingend eine umfassende Vorarbeit. Die RE-Aktivitäten können stattdessen im Rahmen einer agilen Methode auf dieselbe Weise wie andere Aktivitäten (z. B. Codieren oder Testen) durchgeführt werden. Das RE ist eine in jede Iteration integrierte Aktivität.

Missverständnis – vorab ist schlecht: Die Vorbereitung auf eine iterative Entwicklung ist ein wesentlicher Bestandteil jedes nicht alltäglichen Entwicklungsprojekts. Überlegungen im Vorfeld implizieren nicht automatisch einen bestimmten Softwarelebenszyklus oder langwierige Analysephasen und Dokumente: Der Projektkontext bestimmt, wie und wann genau Überlegungen im Vorfeld stattfinden sollen. Anwender sollten die Fachliteratur zu Agilität nicht so interpretieren, dass Überlegungen im Vorfeld an sich eine schlechte Sache sind; das Manifest und die oben dargelegten Prinzipien stützen ein solches Verständnis nicht.

Agile Verfahren, die den Wert von Überlegungen im Vorfeld berücksichtigen, sind Story Mapping (siehe [Patt2014]), Prototyping (siehe [Martin1991]) und das sog. Test Driven Development (TDD) (siehe [Beck2003]).

Missverständnis – RE ist gleich Dokumentation: Das RE wird häufig nur mit den in diesem Zusammenhang erstellten Dokumenten in Verbindung gebracht. Die Dokumente sind allerdings nur ein potenzielles Ergebnis einer Aktivität, die Wissen erzeugt. Bei gutem Requirements Engineering ist man sich stets der Tatsache bewusst, dass selbst das beste Dokument niemals vollumfänglich abgeschlossen ist. Ein Dokument dient stattdessen als Unterstützung des Zwecks, wie in 3.2 definiert: Einhaltung rechtlicher Vorgaben (Compliance), Bewahrung nützlicher Informationen, Vereinfachung der Kommunikation und Unterstützung gedanklicher Vorgänge.

Missverständnis – User Storys sind ausreichend: User Storys sind eine gängige Methode für die Erfassung der Bedürfnisse von Stakeholdern. Sie zielen jedoch darauf ab, die Kommunikation in Gang zu setzen und stellen nicht die vollständige Spezifikation dar. Der Weg von einem unspezifischen Bedarf hin zu einer vollständigen Anforderung ist in dem Verfahren „3C – Card, Conversation, Confirmation" zusammengefasst. Ein weit umfassenderes Bild von Anforderungen lässt sich jedoch durch eine Kombination von User Storys mit anderen anerkannten RE-Techniken wie Kontextdiagrammen, Prototyping, Anwendungsfällen und User Journeys erreichen.

Missverständnis – Dokumentation ist wertlos, nur Code ist von bleibendem Wert: Es ist zwar gut möglich, dass übermäßige Spezifikation in ganz speziellen prozesslastigen Projekten ein Problem darstellen kann, aber es wäre falsch, daraus zu folgern, dass jegliche Dokumentation wertlos ist. Die Anforderungsdokumentation hat ebenso wie die Design-, Test- oder Betriebsdokumentation in bestimmten Kontexten ihre Daseinsberechtigung. Die Dokumente sind alle gleichermaßen gültige und notwendige Arbeitsprodukte, die aus dem Entwicklungsprozess für jedes nachhaltige Softwareprodukt entstehen.

Missverständnis – funktionierende Software ist der einzige Weg zur Validierung von Anforderungen: Im Manifest für agile Softwareentwicklung wird mehr Wert auf Software als auf umfassende Dokumentation gelegt. Eine falsche Schlussfolgerung aus dieser Aussage in puncto Validierung von Anforderungen ist, dass die Anforderungsvalidierung basierend auf funktionierender Software immer der Validierung einer dokumentierten Form von Anforderungen vorzuziehen ist. Software ist als ein Mittel zur Validierung von Anforderungen zu bevorzugen, wenn die mit einem solchen Validierungsansatz verbundenen Kosten und Risiken verglichen mit dem Ergebnis akzeptabel sind. Sind die Kosten und/oder Risiken hoch, stellt RE verschiedene Tools zur Verfügung, die ein schnelles Feedback und eine rasche Anforderungsvalidierung ermöglichen, bevor auch nur eine einzige Zeile Code geschrieben wurde, beispielsweise Modelle für Benutzeroberflächen oder Story Boards.

1.4.3 Stolpersteine von RE@Agile

Stolperstein – Anforderungen als einheitliche Art von Informationen behandeln: Es ist ein typischer Fehler in Bezug auf das RE in allen Implementierungskontexten/-methoden, die „Anforderung" als eine einheitliche Art von Informationen zu betrachten. Die Dokumentation von Anforderungen wird durch dieses Missverständnis in der Regel als Zeitverschwendung angesehen, da sich die Anforderungen so schnell ändern, dass sie schon ungültig sind, sobald sie schriftlich festgehalten wurden. Sie können in verschiedenen Detaillierungsgraden und Formaten sowie auf unterschiedlichen Abstraktionsebenen festgehalten werden. Nehmen wir als Beispiel die Vision oder Ziele für ein System: Dabei handelt es sich um Anforderungen auf einer hohen Abstraktionsebene, die typischerweise eine lange Gültigkeitsdauer haben; ein ausführbarer Prototyp ist ein Mittel zur Validierung einer Menge von Anforderungen oder zur Ermittlung neuer Anforderungen.

Stolperstein – den Zusammenhang aus den Augen verlieren: Agile Methoden werden häufig missverstanden und so umgesetzt, dass der Schwerpunkt nur auf den Themen liegt, die das Team unmittelbar vorliegen hat. Aus Perspektive eines Entwicklers mag dies ein nützliches Prinzip sein, da er seine geistige Energie auf die vorliegende Arbeit konzentriert und nicht durch langfristige Themen abgelenkt wird. Wenn sich aber jeder ausschließlich auf die vorliegende Arbeit konzentriert, geraten der Gesamtzusammenhang und die langfristige Perspektive aus dem Blickfeld. Bei nachhaltigen agilen Methoden werden lang- und mittelfristige Perspektiven in eigens dafür vorgesehenen Sitzungen aufgegriffen (z. B. Verfeinerungs-Meetings, Road-Mapping-Sitzungen oder Workshops zur Produktvision). In diesem Zusammenhang gibt es einen weiteren Stolperstein: Es wird sofort in Lösungen gedacht und diese werden auch so spezifiziert, noch bevor das Problem gründlich untersucht und der Nutzen oder Bedarf des Stakeholders festgestellt wurde.

Stolperstein – Stakeholder mit Informationen überladen: RE- Arbeitsprodukte weisen unter Umständen eine hohe Informationsdichte auf und werden in einem agilen Team sehr schnell erstellt. Dieser Ansatz wird häufig in Projekten für sehr moderne oder Hightechprodukte verwendet, wenn keine Fachexperten greifbar sind. Zieht man jedoch die gesamte Iteration heran, um an RE-Arbeitsprodukten zu arbeiten, so entsteht ein hoher Review-Aufwand für Stakeholder, da diese ausführliche Spezifikationen durcharbeiten müssen. Mit einer Mischung aus Spezifikation und Entwicklung (Prototyping) werden gegebenenfalls bessere Ergebnisse erzielt.

Stolperstein – inkrementelle und iterative Ausarbeitung aller Themen: Nicht jedes Anforderungsthema für ein System sollte bis ins kleinste Detail und inkrementell entwickelt werden. Für eine inkrementelle Entwicklung eignen sich besonders additiv komplexe Themen (siehe [Meyer2014]). Typische Beispiele für solche Themen sind Prozesse, die in unabhängige Elemente unterteilt werden können (z. B. der Einkaufsprozess in einem Onlineshop). In sich hochkomplexe Themen (siehe [Meyer2014]) sind für eine inkrementelle Entwicklung nicht geeignet, da jede neue Erkenntnis zu einem Thema zu einem völlig neuen Verständnis der bereits bekannten Informationen führt. Beispiele hierfür sind Berechnungen mit komplexen Eingabeparametern und einfachen Ausgabeparametern (z. B. Versicherungspolicen oder Komponenten für die Maschinensteuerung).

Stolperstein – inkrementelle Entwicklung unterstützt möglicherweise keine radikalen oder bahnbrechenden Innovationen: Der inkrementelle Prozess agiler Entwicklungsprozesse fördert unter Umständen nicht die Entwicklung innovativer und/oder bahnbrechender Ideen, da ein vorgegebenes Arbeitsprodukt (z. B. die Software oder ein Feature/eine Funktionalität der Software) in der Regel lokal verbessert wird (z. B. Beheben von Fehlern oder Hinzufügen fehlender Elemente), sobald es definiert wurde. Obwohl nach dem Manifest für agile Softwareentwicklung Veränderungen ausdrücklich erwünscht sind, unterstützen die inkrementellen Prozesse agiler Entwicklung typischerweise kontinuierliche Innovationen für Produkte und Services. Radikale oder bahnbrechende Innovationen entstehen, wenn unterschiedliche Ideen berücksichtigt und vorhandene Ideen neu kombiniert werden [LiOg2011]. Bei Software wird das Entwickeln alternativer Ideen in agilen Umgebungen in der Regel als überflüssig empfunden (siehe 10. Prinzip – Maximieren nicht getaner Arbeit). Für radikale oder bahnbrechende Innovationen müssen zusätzliche Verfahren einbezogen werden, beispielsweise Lean-Startup-Ideen oder Design-Thinking-Ansätze, wie in 1.5 dargestellt.

Der zentrale und wichtigste Stolperstein – agile und kulturelle Veränderungen sind unvereinbar: Agile Werte fördern Veränderungen der Art und Weise, wie Unternehmen arbeiten, einen Verlust der Eigentümerschaft an Leistungen und der kollektiven Verantwortung. Agilität fördert darüber hinaus kontinuierliche Retrospektiven zum Verhalten von Teams, um deren Arbeitsweise zu verbessern: Ein Team und gelegentlich das gesamte Unternehmen unterliegen unausweichlich immer wieder Veränderungen. Solche kulturellen (organisatorischen) Änderungen erfordern sowohl Zeit, als auch qualifizierte Mitarbeiter, um die Teams in eine neue Richtung zu führen.

1.5 RE@Agile und konzeptuelle Arbeit (K1)

Agile Entwicklung ist in der Welt des Software-Engineerings entstanden, um Herausforderungen zu begegnen, die der Welt außerhalb des Software-Engineerings entstammten (siehe 1.1). Gleichwohl erlebte nicht nur die Software-Engineering-Welt diese Herausforderungen. Auch andere Industriezweige und die Gesellschaft standen vor ähnlichen Herausforderungen, wie anspruchsvollen Kunden und schnelleren Innovationszyklen. In anderen Bereichen wurden Ansätze für konzeptuelle Arbeit (d. h. das Erarbeiten von Konzepten oder Systemspezifikationen) entwickelt, die der agilen Entwicklung recht ähnlich waren. Einige davon eignen sich aus RE-Perspektive besonders gut für die Entwicklung von Innovationen und Produktvisionen. Diese Ansätze und deren Analogien mit der agilen Denkweise werden hier kurz vorgestellt (siehe 1.2). Nachfolgend präsentieren wir drei Ansätze als Beispiele für Agilität in konzeptueller Arbeit.

Design Thinking (siehe [Dsch2015], [LiOg2011]) ist eine Methode zum Lösen sogenannter „Wicked Problems". Aus RE-Perspektive ist das Design Thinking eine Kombination aus Ermittlungs- und Validierungstechniken.

Im Mittelpunkt dieser Methode stehen (a) ein multidisziplinäres Team, das an dem Problem arbeitet; dieses Team bringt ein breit gefächertes Wissen mit, das für die Problemlösung

benötigt wird; (b) eine Arbeitsumgebung, in der das Team an Ideen arbeiten kann; und (c) ein iterativer Prozess, der sich aus folgenden Phasen zusammensetzt:

- Hineinversetzen („Empathize"): In dieser Phase versetzt sich das Team in die Menschen, die das Problem haben, um ein Verständnis dafür zu entwickeln und das Problem zu lösen.
- Definieren („Define"): In dieser Phase formuliert das Team das Problem neu, um ein gemeinsames Verständnis der Details des zu lösenden Problems zu erhalten.
- Ideen finden („Ideate"): In dieser Phase konzentriert sich das Team auf die Generierung von Ideen. Das Ziel besteht hier nicht darin, die Idee zu entwickeln. Stattdessen entwickelt das Team so viele Ideen wie möglich. Am Ende dieser Phase wählt das Team die aussichtsreichsten Ideen für das Prototyping aus.
- Prototypen entwickeln („Prototype"): In dieser Phase erstellt das Team sehr einfache Prototypen (nicht notwendigerweise Software!) aus den entwickelten Ideen. Das zugrundeliegende Prinzip ist, dass der Prototyp so realistisch und kostengünstig wie möglich sein soll.
- Testen („Test"): In dieser Phase testet das Team den Prototyp mit realen Kunden, um Feedback zu deren Ideen einzuholen. Ein Hauptprinzip für diese Testphase ist „zeigen nicht darüber sprechen", d. h. der Prototyp sollte für sich selbst sprechen können, damit der Benutzer echtes Feedback abgeben kann.

Die Phasen des Design Thinking sind skalierbar und können in Projekten angewendet werden, die von wenigen Tagen bis hin zu mehreren Wochen dauern. Darüber hinaus ist durch die Phasen keine strikte Reihenfolge vorgegeben. Das Team kann in diesem Prozess, wann immer es dies für nötig hält, zurück oder nach vorn gehen. Vor diesem Hintergrund entspricht das Design Thinking dem agilen Wert „das Reagieren auf Veränderungen bewerten wir höher als das Befolgen eines Plans". Das Endergebnis eines Design-Thinking-Prozesses ist eine Reihe von Prototypen, die validierte und innovative Lösungen für ein zu Beginn definiertes Problem darstellen. Daher kann das Design Thinking zur Entwicklung von Ideen für hochwertige Software verwendet werden und unterstützt damit das 1. Agile Prinzip. Wie oben angeführt bilden das multidisziplinäre Team und die Arbeitsumgebung zentrale Elemente des Prozesses, was mit dem 5. Agilen Prinzip in Einklang steht. Ein wichtiges Ziel der Prototypenphase ist die Produktion eines kostengünstigen und schlanken Prototyps, was dem agilen Prinzip der Einfachheit entspricht.

Der **Design-Sprint** [KnZK2016] ist ein fünftägiger Prozess, bei dem Ideen basierend auf den Prinzipen von Design, Prototyping und Tests mit Endkunden entwickelt werden. Aus RE-Perspektive ist der Design-Sprint zudem eine Kombination aus Ermittlungs- und Validierungstechniken. Im Mittelpunkt dieser Methode steht das Arbeiten nach klar definierten Terminen, sogenannten Timeboxes; jeder Tag wird einer der folgenden Aufgaben gewidmet: das Wissen des Teams freisetzen, Ideen skizzieren, entscheiden, für welche Ideen Prototypen erstellt werden sollen, die ausgewählten Prototypen erstellen und schließlich die Ideen mit realen Kunden testen. Der entscheidende Unterschied im Vergleich zu agiler Entwicklung ist, dass der Prototyp keine Software sein muss. Hier ist zu beachten, dass der Begriff „Sprint" sich nicht auf den Scrum-Sprint bezieht.

Lean Startup [Ries2011] ist ein Ansatz zur Entwicklung von Geschäftsideen und zum Managen von Startups, der sich in der Agile-Community sehr gut etabliert hat. Auch aus RE-Perspektive umfasst er einige sehr interessante Ideen. Zwei Beispiele sind der besondere Produktentwicklungsansatz und das Minimum Viable Product. Der Produktentwicklungsansatz wird als „Build – Measure – Learn" (bauen, messen, lernen) bezeichnet, und der Schwerpunkt liegt dabei vor allem auf dem kontinuierlichen Lernen über die Bedürfnisse des Kunden. Mit Minimum Viable Product (MVP) ist sinngemäß eine Version eines neuen Produkts gemeint, die einem Team das Erfassen des maximal möglichen sogenannten validierten Lernens über Kunden bei geringstem Aufwand ermöglicht (siehe [Ries2011]). Ein weiteres wichtiges Konzept von Lean Startup ist der Kurswechsel, d. h. „eine strukturierte Kurskorrektur, um eine neue grundlegende Hypothese über das Produkt, die Strategie und den Wachstumsmotor aufzustellen" [Ries2011]. Aus RE-Perspektive sind diese Ideen eine Kombination aus Ermittlungs- und Validierungstechniken. Anstelle einer Ermittlung und Validierung der Anforderungen basierend auf Konzepten oder Dokumenten, werden diese anhand des realen Produkts durchgeführt, was Ries zufolge in Fällen von äußerster Unsicherheit zu bevorzugen ist. Ries betont, dass das MVP nicht zwingend eine voll funktionsfähige und vollständige Software sein muss. Stattdessen wird in seinem Buch als Beispiel eine einfach gehaltene Website für den Verkauf von Schuhen mit manuellem Versandprozess zur Validierung des Bedarfs für den Onlinekauf von Schuhen vorgeschlagen.

Diese Ansätze zeigen, dass Agilität tatsächlich mehr beinhaltet als Scrum. Die Beispiele zu Design Thinking, Design-Sprint und Lean Startup zeigen, dass es Ansätze für konzeptuelle Arbeit in RE@Agile gibt, die die Denkweise agiler Methoden teilen, und die daher gänzlich mit Unternehmen kompatibel sind, die Software auf agile Weise entwickeln möchten. Diese und andere Herangehensweisen sollten als eine neue Form des Wasserfall-Ansatzes oder von Überlegungen im Vorfeld nicht unberücksichtigt bleiben. Sie können und sollen im Rahmen des Frameworks von agiler Entwicklung (z. B. in einer oder mehreren Iterationen) für das Design dedizierter Aspekte eines Systems verwendet werden. Alternativ können sie als Aktivitäten genutzt werden, die der agilen Entwicklung vorangehen (z. B. eine schnelle Phase von Überlegungen im Vorfeld). So helfen diese Ansätze, das begrenzte Potenzial für Innovationen in agilen Entwicklungsprozessen zu überwinden.

2 Grundlagen von RE@Agile (K2)

Dauer: 2,5 Stunden

Begriffe: Product Owner, Product Backlog, Sprint Backlog, Epics, User-Stories, Story-Maps

Lernziele

LZ 2.1.1 Beispiele für agile Methoden kennen
LZ 2.2.1 Scrum als ein Beispiel kennen: Rollen, Prozesse, Artefakte und deren Relevanz für das Requirements Engineering
LZ 2.2.2 Verantwortlichkeiten eines Scrum Product Owner kennen
LZ 2.2.3 Konzept des Product Backlog und den Unterschied zu einem Anforderungsspezifikations-Dokument kennen
LZ 2.3.1 Unterschied zwischen einem klassischen Requirements Engineer und einem Product Owner verstehen
LZ 2.4.1 Verstehen, weshalb das Requirements Engineering Teil eines kontinuierlichen Prozesses sein sollte
LZ 2.5.1 Wertorientierte Entwicklung kennen (z. B. Priorisierungen von Anforderungen)
LZ 2.5.2 Wissen, dass Wert gleichbedeutend ist mit Risiko- UND Chancenmanagement
LZ 2.5.3 Beispiele für Wert in gewinnorientierten und Non-Profit-Organisationen kennen
LZ 2.6.1 Wissen, wie der Produktdefinitionsprozess vereinfacht und ein Minimum Viable Product definiert werden kann
LZ 2.7.1 Wert kontinuierlicher Prozesse und deren Lernkurve kennen

2.1 Agile Methoden - ein Überblick (K1)

Es wurden zahlreiche Methoden entwickelt, denen die Werte des Manifests für agile Softwareentwicklung gemein sind. In dieser Lerneinheit erhalten Sie einen Überblick über einige dieser Methoden und erfahren, wie vielfältig die Ansätze sind. Die Liste erhebt keinen Anspruch auf Vollständigkeit; die Methoden werden in erster Linie aus Requirements-Engineering-Perspektive betrachtet.

Crystal ist eine Reihe von Methoden, die von Alistair Cockburn [Cock1998] entwickelt wurden. Cockburn zufolge wird für jedes Projekt ein eigens zugeschnittenes Prozessmodell benötigt. Abhängig von Größe, Komplexität und Kritikalität schlägt Cockburn die Verwendung adäquater Rollen, Aktivitäten und Arbeitsprodukte vor. Crystal Clear, eine Methode für kleine und weniger kritische Projekte, ist dem XP sehr ähnlich. Crystal Orange und Crystal Red wurden um einige Formalismen erweitert, um größere Projekte abwickeln zu können. Aus Anforderungsperspektive empfiehlt Cockburn unter anderem das Arbeiten mit wenig formellen Anwendungsfallmodellen und Mock-ups.

Lean Development [Popp2003] und Kanban beruhen auf Prinzipien, die in den 1940er-Jahren zunächst in der Automobilproduktion eingesetzt wurden. Sie wurden im Zusammenhang mit agilen Methoden für IT-Projekte angepasst [Ande2012]. Ziel dieser Ansätze ist es, sieben Arten von Verschwendung im Produktionsprozess aufzudecken (unfertige Vorprodukte, Überproduktion, Mängel,...) und diese bis zur endgültigen Lieferung schrittweise zu beseitigen.

Scrum [Scrum2020] ist ein Framework zum Entwickeln und Erhalten von wertvollen Lösungen als Produkt in komplexen Umgebungen. Das Framework konzentriert sich auf eine iterative, inkrementelle Entwicklung, die auf Erfahrenswerten und Lean Thinking basiert. Es benennt nur drei Schlüsselrollen (in Scrum [Scrum2020] als Verantwortlichkeiten (Accountabilities) bezeichnet): einen Product Owner (zum Managen des Product Backlog, d. h. Definieren der Vision aller relevanten Anforderungen eines Produkts), die Entwickler, die diese Anforderungen in Sprints (Bezeichnung für Iterationen in Scrum) implementieren und einen Scrum Master, der die Anwendung von Scrum sowohl für das Scrum Team als auch für die Entwicklungsorganisation leitet und erleichtert. Im nächsten Abschnitt werden wir Scrum ausführlicher behandeln.

TDD (Test Driven Development) [Beck2003] beruht auf der Idee, dass vor dem Codieren eines Features zunächst der entsprechende Test verfasst wird. Die Testfälle sind sowohl eine exakte, als auch eine detaillierte Spezifikation der Anforderungen, die das Produkt erfüllen soll.

XP (eXtreme Programming) [Beck2004]: Beim XP liegt der Schwerpunkt auf der direkten Kommunikation zwischen einem Kunden und einem Programmierer (der sog. On-Site Customer, der direkt neben dem Programmierer sitzt, ständig Anforderungen bespricht und unmittelbares Feedback in Form von implementierten Features erhält).

2.2 Scrum (plus bewährte Verfahren) als Beispiel (K1)

Scrum ist das am weitesten verbreitete und am häufigsten übernommene Framework für agile Entwicklungsprozesse. Scrum ist ein schlankes Framework, das Menschen, Teams und Organisationen hilft, Werte durch adaptive Lösungen für komplexe Probleme zu schaffen. Der Scrum Guide [Scrum2020] enthält eine Definition von Scrum und dessen grundlegenden Komponenten.

Bei dieser Methode gibt es laut Definition [Scrum2020]:

- Drei Verantwortlichkeiten/Rollen (Scrum Master, Product Owner, Entwickler) Diese drei werden gemeinsam auch als Scrum-Team bezeichnet
- Fünf Ereignisse (Sprint, Sprint-Planung, Daily-Scrum, Sprint Review, Sprint-Retrospektive)
- Drei Artefakte1 und die damit verbundenen Commitments (Product Backlog, Sprint Backlog, Inkrement)

Scrum empfiehlt keinerlei Engineering-Verfahren.

Scrum zufolge sollen Produkte iterativ und inkrementell in einer Folge von Sprints entwickelt werden, in einer Timebox von jeweils höchstens einem Monat. Jeder Sprint führt zu einem nutzbaren Inkrement, das einen konkreten Schritt in Richtung auf das Produktziels darstellt. In diesem Fall ist es die Entscheidung des Product Owners, ob er das Inkrement mit der

1 Arbeitsprodukte gemäß CPRE Terminologie [Glinz2026]

Maßgabe an den Kunden ausliefert, dies in der Regel zu tun, es sei denn, es liegen wichtige Gründe vor (z. B. potenzielle Risiken).

Der Sprint

Der Sprint ist die wesentliche Antriebskraft für die Entwicklung, da er ein iterativer Prozess mit den bekannten Anteilen 'Plan-Do-Check-Act' ist und durch seine kurze Laufzeit schnelle Feedbackzyklen ermöglicht. Jeder Sprint beginnt mit der Sprint-Planung, einem Ereignis, bei dem das Scrum-Team gemeinsam definiert, was im nächsten Produktinkrement entwickelt wird und wie das erreicht werden kann (Zerlegung). Der Ausgangspunkt für diesen Plan wird dem Product Backlog entnommen (eine priorisierte und dynamische Liste aller aktuell bekannten Anforderungen für das Produkt). Im Unterschied zu einem Anforderungsspezifikations-Dokument aus dem klassischen Requirements Engineering, ist ein Product Backlog noch keine fertig ausformulierte Sammlung aller Anforderungen. Oftmals (besonders zu Beginn einer Produktentwicklung) ist es nur eine Liste von grob formulierten Ideen die sich mit der Zeit entwickeln und zu konkreten Anforderungen werden. Je höher ein Element des Product Backlogs priorisiert ist und je näher der Zeitpunkt der Implementierung naht (z. B. in einer der kommenden zwei Sprints), desto genauer erfolgt die Beschreibung eines Elements des Product Backlogs. Der Product Owner ist für das Product Backlog verantwortlich, das sich in seiner aktuellen Struktur am aktuellen Produktziel orientiert. Siehe Details dazu unter 3.1.1 Spezifikationsdokumente vs. Product Backlog.

Die *Sprint-Planung*, die durchgeführt wird, um die Elemente auszuwählen, die im nächsten Sprint implementiert werden, führt zu den Ergebnissen: dem Sprint Ziel (warum), dem Sprint Backlog (was, d.h. die Auswahl der Elemente und ihre Zerlegung in Tasks für den Sprint) und dem Sprint Plan (wie man es erreicht).

Anschließend beginnt das *Scrum-Team* mit der Implementierung des *Produktinkrements*. Die *Entwickler* sind für die Umsetzung der ausgewählten Items in konkrete Ergebnisse verantwortlich. Sie arbeiten mit dem *Product Owner* an der Verfeinerung der *Product Backlog* Items zusammen und geben ihr Feedback zur aktuellen Implementierung ab. Die Entwickler bringen sich jeden Tag während des Daily Scrum auf denselben Stand, um den Fortschritt in Hinblick auf das Sprint-Ziel zu beurteilen und das Sprint Backlog bei Bedarf zu aktualisieren.

Die Arbeit der *Entwickler* wird durch die „Definition of Done" geleitet: eine Definition dessen, was für den Abschluss eines Items im Hinblick auf das Inkrement sowie das Sprint-Ziel erreicht werden muss. Die Definition of Done erleichtert Stakeholdern, den Produktfortschritt eindeutig zu verstehen.

Wenn die Sprint Timebox vorüber ist (maximal 1 Monat), inspiziert das *Scrum-Team* in einem *Sprint Review* das Produktinkrement, und die wichtigsten Stakeholder bewerten die Ergebnisse des *Sprints* um die weiteren Schritte zu besprechen. Das Product Backlog wird entsprechend aktualisiert.

Der Sprint endet mit der *Sprint-Retrospektive*, in der das Scrum-Team inspiziert, wie es effizienter werden kann, was aktuell gut läuft und was verbesserungswürdig ist. Unmittelbar nach Abschluss der *Sprint-Retrospektive* endet der Sprint und der nächste Sprint beginnt.

Bewährte Verfahren

Obwohl im Scrum keine Requirements-Engineering-Techniken eingesetzt werden, hat die Agile-Community einige bewährte Verfahren entwickelt, die gut zu Scrum passen.

Der Scrum Guide verwendet den Begriff Product Backlog Items für die Items, die im Product Backlog aufgeführt sind. Das ist ein allgemeiner Begriff zur Identifizierung aller Arten von Informationen über das zu entwickelnde Produkt; viele Product Backlog Items sind allerdings tatsächlich Anforderungen oder lassen sich zu Anforderungen umdefinieren.

Die Agile-Community schlägt Folgendes vor:

- Zerlegen der Anforderungen in: Epics, optionale Features und User Storys (Granularitätsgrad) siehe [CPREALAGILE2022]
- Unterscheiden zwischen funktionalen Anforderungen (Fähigkeit), Qualitätsanforderungen (Verhalten) und Randbedingungen (Umgebung)
- Gruppieren der Anforderungen nach Themen

Um die Merkmale des Product Backlogs zu beschreiben, wurde ein (zusätzliches) bewährtes Verfahren entwickelt: Das Backlog sollte „DEEP" (Detailed appropriately, Estimated, Emergent, Prioritized, d. h. angemessen detailliert, geschätzt, dynamisch, priorisiert) [Cohn2004] sein. Innerhalb des Sprints gibt es das Backlog Refinement, ein wesentlicher Bestandteil des agilen Entwicklungsprozesses, bei dem Anforderungen besprochen und verfeinert werden.

Mit der "Definition of Done" (DoD) entwickelt das Team ein gemeinsames Verständnis davon, wann ein Product Backlog Item vollständig und bereit ist, Teil des nutzbaren Inkrements zu werden [Scrum2020].

Ein weiteres bewährtes Verfahren ist die Anwendung der INVEST-Regel [Wake2003]. Das Akronym steht für folgende Kriterien:

- **I**: „Independent of each other" – unabhängig voneinander
- **N**: „Negotiable" – verhandelbar
- **V**: „Valuable" – wertvoll, nützlich
- **E**: „Estimable" – schätzbar
- **S**: „Small enough to fit into one sprint" – klein genug für einen Sprint
- **T**: „Testable" – testbar

Im RE werden ähnliche Kriterien für gute Anforderungen genannt (siehe [CPREFL2022]):

- abgestimmt
- eindeutig
- notwendig
- konsistent
- prüfbar
- realisierbar
- verfolgbar
- vollständig
- verständlich

2.3 Unterschiede und Gemeinsamkeiten von Requirements Engineers und Product Ownern (K2)

Lassen Sie uns nach der Besprechung der Prinzipien des Scrum-Frameworks nun die Rolle eines herkömmlichen Requirements Engineers mit der eines Product Owners vergleichen. Dabei muss erwähnt werden, dass die Rolle des Product Owners mittlerweile von vielen agilen Herangehensweisen (auch außerhalb von Scrum) übernommen wurde. Somit ist die folgende Beschreibung also nicht zwingend nur für Scrum zu verstehen.

Ein Requirements Engineer ist für die folgenden zentralen Aktivitäten verantwortlich [CPREFL2022]:

- Anforderungsermittlung
- Anforderungsdokumentation
- Anforderungen prüfen und abstimmen
- Requirements Management

Wie zuvor erläutert, sind die wichtigsten Verantwortlichkeiten eines Product Owners:

- Sicherstellen, dass das Team einen konstanten Wert (Business Value) liefert: Das bedeutet, dass der Product Owner die längerfristige Vision für das Produkt (Produktziel, Produktvision) und die kurzfristigen Bedürfnissen ins Gleichgewicht bringen muss, das Product Backlog nach definierten Kriterien priorisieren muss und die Ergebnisse des Teams zusammen mit den Stakeholdern am Ende jeder Iteration (Sprint) überprüfen muss.
- Managen aller Stakeholder: Der Product Owner ist dafür verantwortlich, konsistente Anforderungen an das Team weiterzuleiten. Er muss Anforderungen aller Stakeholder erfassen und sicherstellen, dass sich die Anforderungen nicht widersprechen. Es muss etwaige Meinungsverschiedenheiten zwischen Stakeholdern lösen, um die Entwickler davon zu entlasten.
- Kontinuierliche Versorgung des Teams mit den Einträgen aus dem Backlog die den größten Wert (Business Value) besitzen: Die Granularität dieser Anforderungen muss klein genug sein, damit sie sich für eine Iteration (Sprint) eignet. Für alle Fragen, die nach dem Planungsmeeting entstehen, muss der Product Owner zur zügigen Klärung zur Verfügung stehen.

Vergleicht man diese beiden Rollen, so zeigt sich, dass sowohl der Requirements Engineer als auch der Product Owner (gemeinsam mit allen Stakeholdern) die zentralen Aufgaben Ermitteln, Dokumentieren, Validieren und Managen von Anforderungen durchführen muss. Die Notationen und Werkzeuge, die dabei zum Einsatz kommen, sind in agilen Entwicklungsprozessen jedoch in der Regel weniger formell:

- z. B. Story-Cards anstelle von Anforderungsdokumenten
- mehr Gespräche und weniger Schreibarbeit
- stärkere Betonung von aktuellen Statusanforderungen, weniger Betonung auf Versionierung und Verlauf

Es ist zwar wichtig, eine Gesamtverantwortung für Anforderungen von hoher Qualität kontinuierlich aufrechtzuerhalten, dabei ist die Rolle des Product Owners breiter gefächert

als die eines herkömmlichen Requirements Engineers: Ein Product Owner ist für den Erfolg des Produkts insgesamt, das fortwährende Einholen von Feedback von den Stakeholdern sowie für das entsprechende Aktualisieren und Priorisieren des Backlogs verantwortlich.

Im Hinblick auf die Tätigkeiten im Requirements Engineering kann zusammengefasst werden, dass der Product Owner dafür verantwortlich ist. Er kann sich dabei von Personen mit Erfahrung im Requirements Engineering unterstützen lassen oder diese Aufgaben selbst durchführen. Die Verantwortung für das Ergebnis des Requirements Engineering sowie die Verantwortung für das Product Backlog verbleibt jedoch beim Product Owner.

2.4 Requirements Engineering als kontinuierlicher Prozess (K2)

Das Requirements Engineering in agilen Entwicklungsprozessen ist keine klar abgegrenzte Phase während der Entwicklung, sondern eine iterative und laufende Aktivität. Die Ermittlung und Analyse sämtlicher Anforderungen, bevor das Design und die Implementierung beginnen können, ist nicht das Ziel; Anforderungen und Produkte werden gleichermaßen iterativ und inkrementell erstellt.

Das Requirements Engineering ist daher eine fortlaufende Aktivität, die so lange dauert wie die gesamte Entwicklung des Produkts. Gleichwohl hat dieser Prozess gut definierte Zwischenergebnisse: die vorausgesagten Anforderungen, welche den größten Wert entsprechend den definierten Kriterien (Business Value) versprechen, sollten „bereit für die Implementierung" sein (im Sinne der oben beschriebenen Definition of Ready). Andere, weniger dringende Anforderungen werden erst dann verfeinert, wenn die dringenden Anforderungen abgeschlossen sind.

Der „kontinuierliche Prozess" schließt einige wichtige Vorabaktivitäten nicht aus. Selbst wenn für Anforderungen jeweils rechtzeitig auf Basis des Need-to-Know Prinzips geklärt werden, so gibt es einige Aspekte, die dennoch frühzeitig im Projektlebenszyklus behandelt werden müssen. Als Beispiele seien das Definieren von Visionen oder Zielen, das Kennen der Stakeholder und das Erarbeiten des Produktumfangs genannt. Beginnt man die Implementierung ohne diese Aktivitäten, so erhöht sich das Risikolevel erheblich.

2.5 Wertorientierte Entwicklung (K1)

Mit agilen Methoden wird das Ziel verfolgt, dem Endbenutzer kontinuierlich Wert (Business Value) zu liefern. Wert lässt sich in finanzieller Hinsicht, in einem höheren Marktanteil oder in Bezug auf die Kundenzufriedenheit direkt ausdrücken. Dieser Ansatz wird häufig in gewinnorientierten Organisationen verfolgt, aber für Non-Profit-Organisation lässt sich der Wert nicht ganz so klar definieren. Für diese können Kennzahlen wie Nutzungsquoten oder der Zufriedenheitsindex für ein Produkt (d. h. Klickraten auf Websites oder Spenden für eine Non-Profit-Organisation) relevanter sein.

Eine andere Art von Wert ist die Risikoreduktion. Gute agile Ansätze versuchen, ein Gleichgewicht zwischen Business Value und Risikominderung für die Iterationen herzustellen.

Um die Anforderungen mit dem optimalen Wert zu ermitteln, streben agile Methoden häufig Minimum Viable Products (MVPs) oder Minimum Marketable Products (MMPs) an, wie im nächsten Absatz erläutert wird.

2.6 Einfachheit als wesentliches Konzept (K1)

Einfachheit ist eine Möglichkeit, um den Herausforderungen in einem komplexen Umfeld zu begegnen.

- Erarbeiten einer einfachen, potenziell unvollständigen Lösung für ein Problem, wodurch ein kleines Wertinkrement geschaffen wird,
- Entwickeln der Fähigkeit, basierend auf Praxiserfahrungen mehr über den Kontext zu erfahren,
- Anpassen und Erstellen von Iterationen zur Schaffung und Auslieferung von Werten in einem kontinuierlichen Tempo,
- Einsparen von Ressourcen von einer nicht gewinnorientierten Idee, um diese einer neuen Idee zuzuweisen – durch schnelles Scheitern und Lernen.

Einfachheit steht in gewisser Weise im Gegensatz zu „Perfektion". In diesem Sinne bedeutet Einfachheit nicht „schlechte Qualität" sondern „minimaler Umfang" oder „minimaler Service" – jedoch immer mit hoher Qualität. In Bezug auf Qualität gibt es keine Kompromisse.

Das Agile Manifest beschreibt Einfachheit als "die Kunst, die Menge der nicht erledigten Arbeit zu maximieren". Dies bedeutet nicht, dass man sich nicht anstrengen muss, um die Arbeitsbelastung zu verringern, sondern dass man sich über die zu erledigende Arbeit im Klaren sein muss, um einen Mehrwert zu schaffen und bestimmte Ziele zu erreichen. Wenn das Team nicht abgestimmte Anforderungen realisieren würde, dann würde es unbrauchbare Arbeit maximieren – was vermieden werden sollte, um nicht Kosten und Zeit ohne triftigen Grund zu verbrennen.

Häufig werden dabei zwei Arten „einfacher" Produkte definiert: MVPs und MMPs

- Ein Minimum Viable Product (MVP) ist ein Konzept des Lean-Startups (siehe [Ries2011]), und es wird als das kleinste Produkt definiert, das eine Endbenutzererfahrung und damit dem Team Feedback bieten kann. Dieses Feedback ist für die Weiterentwicklung des Produkts äußerst wichtig. Viele Start-ups orientieren sich an dieser Arbeitsweise, da sie eine rasche Investitionsrendite bei einem geringen Risikolevel ermöglicht.
- Der Zweck eines Minimum Marketable Products (MMP) geht noch einen Schritt weiter. Es geht hierbei nicht nur darum, frühzeitig Feedback bereitzustellen und damit die nächsten Anforderungsschritte voranzubringen, sondern es geht um die sofortige Schaffung von Wert. Viele Produkte können bereits in einer einfachen „Version 1" genutzt werden, ohne dass diese Version alle gewünschten Funktionalitäten und Qualitäten umfasst; so erzielt das Produkt bereits Umsatz, mit dem die kontinuierliche Verbesserung (das sog. Continuous Improvement) des Produkts finanziert werden kann. Dabei wird häufig der Lean-Startup Prozess (Build, Measure, Learn) iterativ eingesetzt.

2.7 Inspizieren und anpassen (inspect and adapt) (K1)

In agilen Methoden wird die Bedeutung von häufigem und schnellem Feedback betont. Das Team sollte nach jeder Iteration (manchmal sogar häufiger) besprechen, ob der Entwicklungsprozess für das Team funktioniert oder verbessert werden sollte.

In diesem Feedbackprozess sollte jeder den aktuellen Entwicklungsprozess bereits in einem frühen Stadium inspizieren („inspect"). Das Team sollte die eingesetzten Methoden, die Werkzeuge, die Zusammenarbeit im Team usw. auf den Prüfstand stellen. Jeder wird um seine Meinung zu Fragen wie etwa den folgenden gebeten: Was hat gut funktioniert? Was hat nicht so gut funktioniert? Was sollten wir in der nächsten Iteration ausprobieren?

Es ist wichtig, dass das Produkt oder die Lösung am Ende jeder Iteration besprochen wird. Das Team sollte das Arbeitstempo innerhalb des Teams beleuchten und die geplante Implementierungsgeschwindigkeit für die nächste Iteration entsprechend ändern oder anpassen.

Dies gilt auch für den Anforderungsprozess. Die Konsequenz aus diesen Erkenntnissen sollte die kurzfristige Anpassung („adapt") von Prozessverbesserungsschritten sein.

3 Arbeitsprodukte und Techniken in RE@Agile (K2)

Dauer: 2,5 Stunden

Begriffe: Produktvision, Produkt-Roadmap, Product Backlog, Sprint Backlog, User Story, Akzeptanzkriterien, Feature, funktionale Anforderung, Qualitätsanforderung, Epic, Kontextmodell, Story-Map, Definition of Ready, Definition of Done

Lernziele

LZ 3.1.1 Unterschied zwischen herkömmlichen Spezifikationsdokumenten und Backlogs kennen
LZ 3.1.2 Wert von Visionen und Zielen kennen
LZ 3.1.3 Mehrwert für die Verwendung von Kontextmodellen im RE kennen
LZ 3.1.4 Wissen, wie sich die drei Arten von Anforderungen unterscheiden
LZ 3.1.5 Die verschiedenen Granularitätsstufen bei Anforderungen verstehen
LZ 3.1.6 Unterschiedliche Spezifikationsformate für die verschiedenen Arten von Arbeitsprodukten in agilen Prozessen verstehen (d. h. textuell vs. vorlagenorientiert vs. grafisch)
LZ 3.1.7 Wert von Begriffen, Glossaren und Informationsmodellen verstehen
LZ 3.1.8 Spezifikation von Qualitätsanforderungen und Randbedingungen in agilen Requirements-Engineering-Prozessen verstehen
LZ 3.1.9 Akzeptanz- und Abnahmekriterien kennen
LZ 3.1.10 Verwendung der Definition of Ready und der Definition of Done in agilen Requirements-Engineering-Prozessen verstehen
LZ 3.1.11 Unterschied zwischen Prototyp, Inkrement und Spike kennen
LZ 3.1.12 Unterschiedliche Arbeitsprodukte in RE@Agile-Prozessen kennen (Kontextmodell, Epic, User Story, Backlog, Roadmap, Anforderung, Definition of Done, Definition of Ready)
LZ 3.2.1 Art und Weise der Ermittlung von Anforderungen im agilen Requirements Engineering verstehen
LZ 3.2.2 Art und Weise der Erstellung und Bearbeitung von Backlogs in agilen Requirements-Engineering-Prozessen kennen
LZ 3.2.3 Art und Weise der Validierung und Abstimmung von Anforderungen in RE@Agile kennen
LZ 3.2.4 Art und Weise des Managements von Anforderungen in RE@Agile kennen

3.1 Arbeitsprodukte in RE@Agile (K2)

3.1.1 Spezifikationsdokumente vs. Product Backlog

Um aus Anforderungen Produkte oder Lösungen zu erstellen, können Anforderungen nicht für sich alleine stehen; Sie müssen in Form einer priorisierten Liste organisiert und dokumentiert werden und alles enthalten, was das Produkt später ausmachen soll [Scrum2020]. Unabhängig von jeglicher Methodologie wird die Sammlung von Anforderungen als zentrales Arbeitsprodukt für Requirements Engineers, Business-Analysten, Product Owner oder andere, die für die Anforderungsanalyse verantwortlich sind, betrachtet. In den unterschiedlichen Methoden werden verschiedene Formen und Bezeichnungen für diese Sammlung von Anforderungen verwendet.

Im Requirements Engineering werden sie gewöhnlich als Arbeitsprodukte, als Ergebnis des Ermittlungsprozesses, als Benutzeranforderungsspezifikation, als Systemanforderungsspezifikation, als Lastenheft oder als Softwareanforderungsspezifikation bezeichnet, je nachdem, wer diese Anforderungssammlung verfasst und welchen Detaillierungsgrad sie abdeckt. Es ist nicht notwendigerweise dokumentenbasiert, sondern einfach eine Sammlung von Anforderungen in jeglicher physischen Form (Papier, Repository-basiert, Datenbank usw.).

Agile Methoden, vor allen Dingen Scrum, definieren den Begriff Product Backlog für diese Anforderungssammlung (und andere produktbezogene Informationen, siehe 2.1), die in der Zukunft implementiert werden sollen, und den Begriff Sprint Backlog für diejenigen Anforderungen, die für die nächste Iteration (Sprint in Scrum) ausgewählt wurden [Scrum2020]. Auch hier ist die physische Erscheinungsform des Backlogs nicht relevant. Das Backlog kann aus Karteikarten oder Haftnotizen an einer Wand bestehen oder in einem geeigneten Softwaretool erfasst werden. Auch wenn die Bezeichnungen und die Behandlung unterschiedlich sein mögen, beruhen alle Arbeitsprodukte auf denselben Ideen und bieten eine Basis für die Ermittlung, die Dokumentation, die Abstimmung, die Validierung und das Management von Anforderungen.

In 2.2 haben wir gelernt, dass DEEP eine gute Praxis ist, und dabei das Product Backlog angemessen detailliert, geschätzt, emergent und priorisiert wird. Diese Merkmale sind wichtig für das Product Backlog und sie sind miteinander verknüpft oder voneinander abhängig. Für den Product Owner als "Wertoptimierer" ist die Priorisierung oder die Sortierung des Product Backlog das Wertvollste und Wichtigste. Die Schätzung unterstützt dies und die entsprechende Detaillierung hilft, sich auf die wichtigen Teile zu konzentrieren, sodass sie die Priorisierung und Sortierung unterstützen. Unabhängig von der Art wie Anforderungen dokumentiert werden sollten bestimmte Elemente unbedingt erfasst werden. Dazu zählen alle Arten von Anforderungen: Ziele und Visionen, die Definition des Umfangs eines Systems, funktionale Anforderungen, Qualitätsanforderungen und Randbedingungen sowie ein Glossar (d. h. Definitionen relevanter Begriffe und Abkürzungen). Wie wir später noch behandeln werden, können die Ansätze für Anforderungsspezifikationen in Notationen, in der Syntax und im Detaillierungsgrad variieren. Gar keine Spezifikation zu erstellen (d. h. ohne schriftlich erfasste Anforderungen, ausschließlich auf verbale Kommunikation zwischen Stakeholdern zu setzen), bildet jedoch in der Regel keine Alternative, da schriftliche Dokumente häufig als Basis für Abstimmungen und Akzeptanztests sowie zur rechtlichen Absicherung etc. dienen. Je mehr alle Stakeholder miteinander kommunizieren, umso weniger Schreibarbeit ist erforderlich, doch die Ergebnisse, also die Anforderungen, sollten in einer vertrauten Form erfasst werden (schriftlich oder grafisch). In den folgenden Abschnitten werden die verschiedenen Bestandteile dieser gesamten Sammlung von Anforderungen im Detail behandelt.

3.1.2 Vision und Ziele

Jeder Entwicklungsprozess sollte von Visionen oder Zielen geprägt sein, welche die Produktfunktionen definieren; werden diese erfolgreich implementiert, gilt die Lösung als erfolgreich. Ziele und Visionen, über die sich alle Stakeholder so früh wie möglich einig sind,

sind äußerst wichtig für die Aktivitäten in Verbindung mit den Anforderungen des entsprechenden Systems.

In agilen Entwicklungsprozessen wird häufig der Begriff „Produktvision" verwendet, um zu betonen, dass jedes Ergebnis des Entwicklungsprozesses einen klaren Wert haben sollte, der sich auf die Produktvision bezieht. Um diesen Wert zu definieren muss zuvor gegebenenfalls klar geregelt sein, was die Werte eines Unternehmens sind nach denen es strebt.

Die Ziele verschiedener Stakeholder können im Widerspruch zueinander stehen, das heißt, die Stakeholder müssen sich abstimmen, um eine vereinbarte Vision oder abgestimmte Ziele zu entwickeln. Dies könnte aber auch ein Hinweis dafür sein, dass Varianten des Produkts (z. B. ein kleines und ein großes iPhone) oder sogar völlig verschiedene Produkte benötigt werden (ein iPhone und ein iPad).

Die agile Entwicklung arbeitet häufig mit Zielen von unterschiedlicher Granularität, so etwa Ziele für unterschiedliche Zeithorizonte oder Planungsintervalle. Es kann beispielsweise Ein-Jahres-Ziele geben, um Abstimmungen über Auslieferungen (Zeitpunkt und Inhalt) zu ermöglichen, Drei-Monats-Ziele für die Release-Planung und Iterations-Ziele/Sprint-Ziele für die nächste Iteration/den nächsten Sprint [High2009].

Langfristige Produktvisionen und kurzfristige Ziele sind hilfreich, um die wichtigsten Leistungen hervorzuheben, die innerhalb eines bestimmten Zeitrahmens erreicht werden sollen, und um die Interessen aller Stakeholder auf eine „gemeinsame Mission" einzustimmen. All diese Ziele lassen sich konstruktiv auf einer Zeitachse in der Roadmap darstellen.

Die Produktvision oder -ziele sind die abstrakteste Form von Anforderungen, die nicht ohne weitere Verfeinerung in die Entwicklung übernommen werden können. Sie dienen als leicht verständliche und umfassende Orientierungshilfe für den gesamten agilen Entwicklungsprozess. Jede Anforderung sollte auf die Ziele hin überprüft werden, um den Beitrag zu verifizieren, den sie für die unterschiedlichen Ziele leistet. Wenn eine Anforderung keine Beziehung zu den Zielen hat, kann das auf mangelnden Wert hindeuten. Folglich sind die Ausarbeitung der Produktvision und die abgestimmten Ziele der Stakeholder sehr wichtige Arbeitsprodukte für den Erfolg agiler Entwicklungsprozesse, da sie den Rahmen für alle Entwicklungsaktivitäten vorgeben, ohne die Entwickler unnötig in ihrer Kreativität einzuschränken.

3.1.3 Kontextmodell

Die Darlegung der Vision und die Ziele der Stakeholder beschreiben sämtliche Bedürfnisse, die das System befriedigen soll, um seinen Zweck zu erfüllen. Kontextmodelle stellen dagegen einen anderen Blickwinkel dar, da sie zum Ziel haben, bestimmte Eigenschaften der Umgebung (des Kontextes), in dem das System funktionieren soll, zu beschreiben.

Die Anforderungen für ein System werden häufig unter Berücksichtigung von Annahmen über die Umgebung festgelegt. Kontextmodelle sind eine strukturierte Weise der Dokumentation relevanter Annahmen über den Kontext. Es ist vorteilhaft, solche Annahmen explizit zu formulieren, um eine gemeinsame Sicht der operativen Umgebung des Systems

für das gesamte Team oder andere relevante Stakeholder zu etablieren, mit der alle einverstanden sind.

Bei Unsicherheiten werden die festgelegten Anforderungen nur korrigiert, wenn die in den Kontextmodellen dokumentierten Annahmen sich als wahr erweisen, das heißt, dass die Kontextmodelle den tatsächlichen operativen Kontext des Systems korrekt darstellen.

Kontextmodelle sind ein leistungsstarkes Arbeitsprodukt, da sie klar zwischen dem zu entwickelnden System und dessen Kontext unterscheiden, der beispielsweise aus Nachbarsystemen und menschlichen Benutzern besteht ([CPREFL2022]). Unter Verwendung dieser Differenzierung können Funktionalitäten zugewiesen werden.

Dadurch kann zwischen den Verantwortlichkeiten des Systems und den Verantwortlichkeiten von Nachbarsystemen oder menschlichen Benutzern im Kontext unterschieden werden, die bei einem Vorgang zusammenarbeiten, um die Gesamtvision zu erfüllen. Daher eignen sich Kontextmodelle auch zur Klärung und Spezifizierung der externen Schnittstellen des zu entwickelnden Systems.

Solche Modelle können in verschiedenen Formaten dokumentiert werden, etwa als Kontextdiagramme für strukturierte Systemanalysen, Grafiken für Anwendungsfälle, Diagramme für SysML-Blockdefinitionen, UML-Komponentengrafiken oder UML-Klassendiagramme. Geeignet sind alle Notationen, sofern sie eindeutig zwischen dem zu entwickelnden System und externen Schnittstellen oder Personen bzw. Systemen in der Umgebung differenzieren. Sie müssen zudem die Beziehungen zwischen diesen Elementen und dem zu entwickelnden System in einem angemessenen Detaillierungsgrad dokumentieren. Selbst einfache, von Hand gezeichnete Kastengrafikdiagramme können pragmatisch und nützlich sein.

Ungeachtet der Dokumentationsform, ist ein Kontextmodell ein sehr nützliches Arbeitsprodukt, das in einem agilen Entwicklungsprozess zu empfehlen ist. Es grenzt den Bereich (im Rahmen des Umfangs) ab, in dem Analysten freie Entscheidungen treffen können, während die externen Schnittstellen (d. h. die Grenze zwischen Umfang und Kontext) mit den Nachbarsystemen abgestimmt werden müssen.

3.1.4 Anforderungen

Anforderungen müssen dann auf Basis der Vision und der Ziele erfasst werden; sie sind durch das Kontextmodell begrenzt. Typischerweise unterscheidet man zwischen drei Arten von Anforderungen: funktionale Anforderungen, Qualitätsanforderungen und Randbedingungen [CPREFL2022].

3.1.5 Granularität von Anforderungen

Stakeholder kommunizieren ihre Bedürfnisse häufig in unterschiedlichen Granularitätsstufen: von groben Anforderungen wie allgemeinen Geschäftszielen, bis hin zu detaillierteren Anforderungen, in denen die Einzelheiten zu erwarteten Systemfunktionalitäten festgelegt sind. Funktionale und Qualitätsanforderungen (siehe 3.1.8) können (und sollten!) daher auf unterschiedlichen Abstraktionsebenen besprochen und dokumentiert werden.

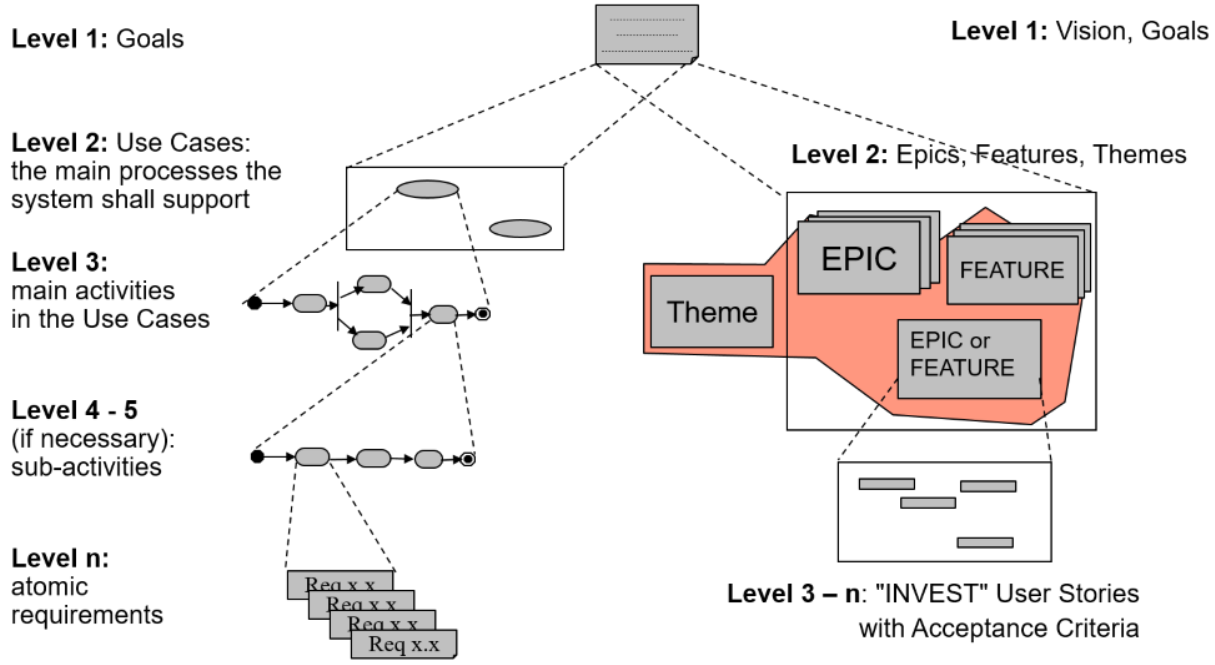

Abbildung 1: Granularität von Anforderungen: Zerlegung eines Anwendungsfalls und agile Varianten, Anwendungsfälle, Spezifikationen für Anwendungsfälle und Aktivitätsdiagramme

Ein typischer Ansatz für nicht-agile Methodologien ist in Abbildung 1 links dargestellt. Bei diesem Ansatz werden Anwendungsfälle verwendet, um Visionen und Ziele zunächst zu verfeinern und eine Unterteilung der erwarteten Systemfunktionalitäten bereitzustellen (siehe Abbildung 1 – Level 2). Die Darstellung zeigt lediglich ein Beispiel für eine Gruppierung von Anforderungen auf einem hohen Abstraktionslevel. Weitere Beispiele umfassen die Gruppierung nach Funktionalitäten, betriebswirtschaftlichen Objekten, Datenflüssen oder nach Komponenten bestehender Lösungen.

Auf der nächsten Granularitätsstufe wird jeder Anwendungsfall ausgearbeitet, um die für die Fertigstellung der Funktion benötigten Schritte zu beschreiben. Je nach Komplexität sind hier verschiedene Alternativen verfügbar:

1. Verbale Beschreibung (reiner Text)
2. Anwendungsfallvorlagen (teilweise strukturiertes Format oder narrative Abläufe)
3. UML-Aktivitätsdiagramm (oder eine andere Art von grafischem funktionalen Modell, z. B. Datenflussdiagramm, BPMN, Kontextdiagramm, Zustandsdiagramm, Geschäftsobjektmodell usw.)

Das Level 3 in Abbildung 1 zeigt Anwendungsfälle, die beispielsweise mit UML-Aktivitätsdiagrammen ausgearbeitet wurden.

Basierend auf der Komplexität des Problems können weitere Granularitätsstufen bereitgestellt werden, so etwa durch das Zerlegen von Aktivitäten in Teilaktivitäten (siehe Level 4 in Abbildung 1) oder durch textuelle Anforderungsspezifikationen einzelner Aktivitäten (siehe Level 5 in Abbildung 1).

Auf diese Weise werden grobe Kundenanforderungen schrittweise in detailliertere Spezifikationen der erwarteten Lösungsfunktionalität verfeinert.

Beachten Sie, dass der obige Ansatz einen Top-down-Prozess der Anforderungsanalyse und -zerlegung impliziert. Diese leicht idealisierte Sichtweise ist nicht immer realistisch. Anforderungen können auch zuerst auf detaillierter Lösungsebene formuliert werden, und die allgemeineren Ziele folgen erst später. Der Prozess kann daher in der Praxis ein Top-down- oder Bottom-up-Prozess sein oder auch eine Kombination aus beiden.

Der entscheidende Punkt ist, dass Methoden und Techniken zur Verfügung stehen, um die Gespräche auf den verschiedenen Abstraktionsebenen zu unterstützen und eine aussagekräftige Hierarchie von Anforderungen zu erstellen, da das Wissen über die Anforderungen insgesamt zunimmt.

Beispielsweise können Anwendungsfallvorlagen zur Beschreibung von Anwendungsfällen geringerer Komplexität verwendet werden, indem die auszuführenden Schritte lediglich anhand einiger Sätze erläutert werden. Für Anwendungsfälle mittlerer Komplexität sind möglicherweise andere Optionen eine gute Wahl, da die Anwendungsfallvorlagen eine Beschreibung komplexerer Prozessschritte ermöglichen, die Teil des spezifischen Anwendungsfalls sind. Dieses teilweise strukturierte Format erleichtert durch eine klare Richtlinie das Verständnis der in einem Anwendungsfall erfassten Anforderungen. Dennoch hat die Anwendungsfallvorlage auch Grenzen, wenn es parallele Aktivitäten und eine hohe Komplexität durch mehrere Szenarien gibt.

Schließlich kann der Requirements Engineer mit Aktivitätsdiagrammen auch ein grafisches Format wählen, bei dem mehrere Szenarien angezeigt werden können. In einem Diagramm lassen sich mehrere parallele und sequenzielle Schrittfolgen darstellen. Diese Form der Anwendungsfallbeschreibung ermöglicht die Darstellung komplexer Abläufe von Anwendungsfällen in einem Diagramm.

Ein Punkt der immer wieder zu Diskussionen führt ist, wann eine Anforderung ausreichend detailliert beschreiben ist. Hier gilt es primär die Zusammenarbeit im Team in den Vordergrund zu stellen. Eine Anforderung ist genau dann ausreichend detailliert beschrieben, wenn das Team (speziell die Personen, die die Anforderung umsetzten, also die Entwickler) verstanden hat was gewollt ist. Empfehlenswert ist es hierfür vorab klare Kriterien zu definieren. Siehe dazu auch Definition of Ready und Definition of Done.

Terminologie aus dem Bereich Agilität: Epics, Themen, Features und User Storys

In Abbildung 1 rechts ist ein äquivalenter Ansatz unter Verwendung von Begriffen aus dem Gebiet der Agilität, etwa Epics, Themes (Themen), Features (Merkmale) und User Storys dargestellt.

Hier werden übergeordnete Geschäftsziele und komplexe Funktionalitäten in Form von Epics oder Features erfasst und nach Themen gruppiert. Die Themen können so komplex sein, dass z. B. ein Scrum-Team zur Entwicklung mehr als einen Sprint benötigt [Griff2015].

Derart komplexe Themen werden in detailliertere User Storys zerlegt. Die Kriterien für „gute" User Storys wurden bereits erläutert (2.2). Sie sollten der Definition of Ready [Kron2008] folgen, z. B. das Erfüllen der INVEST-Kriterien [Wake2003]. Die Punkte „E", „S" und „T" sind besonders wichtig: Sie sollten schätzbar („Estimable"), klein genug für einen Sprint („Small enough to fit into one sprint") und testbar („Testable") sein.

Kurz zur Wiederholung: Ob ein Projekt einem strengen Top-down-Prozess der Verfeinerung von Epics in Features und dann in User Storys folgt, oder ob zuerst einzelne User Storys mit allgemeinen Themen und Epics erfasst werden, die dann in einem Bottom-up-Prozess bearbeitet werden, hängt von Art des Projekts und den beteiligten Stakeholdern ab. In jedem Fall bieten agile Entwicklungsprozesse Arbeitsprodukte, um das Gesamtbild und die entwicklungsbereiten Anforderungen zu besprechen und zu priorisieren.

3.1.6 Grafische Modelle und textuelle Beschreibungen

Unabhängig von der Granularität, besteht immer eine Wahl der Notation für funktionale Anforderungen. Sie können schriftlich in halb-strukturierter Textform (wie User Storys gemäß einer Vorlage) verfasst, in rein natürlicher oder formaler Sprache (wie Gherkin) ausgedrückt werden, wodurch klar festgehalten wird, was die Lösung beinhalten soll.

Da bekannt ist, dass natürliche Sprache manchmal nicht so präzise und eindeutig ist wie nötig, wurden zahlreiche grafische Notationen entwickelt, um diese Mehrdeutigkeit auszuräumen oder um wechselseitige Abhängigkeiten darzustellen, beispielsweise zwischen Daten und Prozessen. Beispiele dafür sind UML-Aktivitätsdiagramme, BPMN-Diagramme, Flussdiagramme, Zustandsdiagramme, Sequenzdiagramme usw.

Bei den meisten dieser auf Grafiken basierenden Notationen liegt die Betonung auf unterschiedlichen Aspekten. Aktivitätsdiagramme oder BPMN-Diagramme eignen sich etwa sehr gut für eher lineare Prozesse, die aufeinanderfolgende Schritte, Alternativen oder Wiederholungen aufweisen. Zustandsdiagramme hingegen sind optimal geeignet, wenn ein Ablauf durch asynchrone Ereignisse beeinflusst werden könnte. Sequenzdiagramme eignen sich perfekt für „Spezifikationen nach Beispiel", da sie konkrete Szenarios von Interaktionen ohne einen Anspruch auf Vollständigkeit darstellen.

Die Modellierung von Prozessen und Daten, Daten und Interaktionen oder Prozessen und Schnittstellen weist positive und negative Aspekte auf. Sie stehen in Wechselbeziehung zueinander (Daten unterstützen den Prozess), sodass niemals eine Entweder-oder-Situation auftreten kann, was nahelegt, dass nur das eine oder das andere Modell verwendet werden sollte. Diese Herangehensweise wäre allerdings recht kontraproduktiv. Textuelle Anforderungen werden zwar von vielen Stakeholdern leichter verstanden, sie können aber auch von denselben Lesern leicht fehlinterpretiert oder missverstanden werden. Grafische Modelle weisen ein höheres Maß an Formalität auf, wodurch unterschiedliche Interpretationen oder Missverständnisse vermieden werden. Die Wahl der Form sollte durch die zentralen Ziele des Requirements Engineering getroffen werden, um einerseits ein gemeinsames Verständnis bei allen Stakeholdern sicherzustellen und andererseits ausreichend Schutz vor Unvollständigkeit und Missverständnissen zu bieten [GoAk2003].

3.1.7 Definition von Begriffen, Glossare und Informationsmodelle

Funktionale Anforderungen sind ohne ein klares Verständnis sämtlicher in diesen Beschreibungen und grafischen Modellen verwendeten Begriffe unvollständig.

Daher ist eine Erfassung aller relevanten betriebswirtschaftlichen Begriffe, die in einem Anforderungsartefakt verwendet werden, ein notwendiges Arbeitsprodukt, das jedoch leicht in Vergessenheit gerät, wenn man sich nur auf Geschäftsanforderungen konzentriert.

Die Mindestform dieses Arbeitsprodukts ist eine Liste von (textuell beschriebenen) Geschäftsbegriffen und Abkürzungen, die zum leichteren Nachschlagen gewöhnlich alphabetisch sortiert werden. Gelegentlich finden sich dafür auch die Bezeichnungen Wörterbuch, Glossar oder Liste von Definitionen.

Wenn das Geschäftsfeld sehr komplex ist, kann das Glossar strukturiert werden, indem man einfache Begriffe in Klassen (oder Entitäten) gruppiert und eine grafische Darstellung einer Übersicht über alle Begriffe erstellt. Solche Arbeitsprodukte werden dann als Datenmodelle, Informationsmodelle, Entity-Relationship-Diagramme oder UML-Klassendiagramme bezeichnet. Neben der Begriffsdefinition enthalten diese Modelle auch relevante Assoziationen zwischen den Entitäten, d. h. statische Beziehungen zwischen den Begriffen.

Wie zuvor erwähnt, besteht das Product Backlog oftmals aus Epics, Features und User Storys, in welchen die benötigten Funktionalitäten hervorgehoben werden, die Definitionen von Geschäftsbegriffen jedoch in den Hintergrund geraten. Dennoch ist selbst in agilen Ansätzen ein klares Verständnis der Geschäftsbegriffe erforderlich, um ein gemeinsames und abgestimmtes Verständnis der in Arbeitsprodukten wie Epics, Features und User Storys verwendeten Begriffe zu schaffen.

3.1.8 Qualitätsanforderungen und Randbedingungen

Neben den funktionalen Anforderungen (Spezifizieren von Funktionen, die ein System bieten muss) sind Qualitätsanforderungen und Randbedingungen von äußerster Wichtigkeit für den Erfolg des entwickelten Systems. Qualitätsanforderungen und Randbedingungen werden traditionell unter dem Sammelbegriff „nicht-funktionale Anforderungen" zusammengefasst [CNYM2000]. Experten im Requirements Engineering verteidigen die Bedeutung dieser „nicht-funktionalen" Anforderungen schon seit Jahrzehnten. Auch wenn der Begriff „nicht-funktionale Anforderungen" in der Praxis immer noch häufig als Überbegriff für Qualitätsanforderungen und Randbedingungen zu finden ist, verwendet das IREB die konkreteren und präziseren Kategorien „Qualitätsanforderungen" und „Randbedingungen" nach [Glinz2026].

Qualitätsanforderungen beziehen sich auf bestimmte Qualitäten, die ein System aufweisen muss, beispielsweise in Bezug auf Performance, Zuverlässigkeit, Sicherheit, Schutz oder Benutzbarkeit [ISO25010]. Liegt das Hauptaugenmerk auf funktionalen Kundenanforderungen in Form von User Storys, dann besteht bei agilen Entwicklungsprozessen das Risiko, dass Qualitätsanforderungen nicht explizit festgehalten werden. Qualitätsanforderungen können nicht als User Storys definiert werden, die innerhalb einer Iteration entwickelt werden können: Sie beschreiben vielmehr ein aus dem entwickelten Produkt entstehendes Attribut und müssen daher für alle User-Storys kontinuierlich getestet werden. Hierfür sind beispielsweise RE-Checklisten für die Qualitätsaspekte (siehe [CPREFL2022]) nützlich. Eine weitere Möglichkeit wäre sämtliche Qualitätsanforderungen in die Definition of Done (DoD) aufzunehmen und so sicherzustellen, dass alle

Qualitätsanforderungen eingehalten werden, damit ein Backlog Item Teil des potentiell einsetzbaren Inkrements wird [CPREALAGILE2022].

Es ist bekanntermaßen schwierig, durch Refactoring Qualitätsanforderungen in vorhandene Systeme zu integrieren; daher ist es umso wichtiger, Qualitätsanforderungen schon frühzeitig im Prozess zu berücksichtigen.

Einschränkungen des Lösungsraums eines zu entwickelnden Systems werden durch Randbedingungen definiert [Glinz2026]. Randbedingungen treten in einer Reihe von Formen auf: organisatorische Randbedingungen (Budgetbegrenzungen, enge Terminpläne, ein vorgegebener Entwicklungsprozess usw.), technische Randbedingungen (erforderliches DB-System, die Nutzung einer bestimmten Programmiersprache, ausgewählte Frameworks usw.) oder Einschränkungen der Umgebung an sich, in deren Rahmen das System betrieben wird (Standards, Normen, Vorschriften usw.).

Ähnlich wie bei Qualitätsanforderungen, wird es unter Umständen auch sehr teuer, wenn man zu spät von zentralen Randbedingungen erfährt, da viele derartige Aspekte nicht inkrementell hinzugefügt werden können. Planungs- und Designentscheidungen hängen von einer guten Kenntnis dieser Probleme ab, daher ist es entscheidend, dass die wichtigsten Randbedingungen früh im Prozess identifiziert werden.

Ähnlich wie funktionale Anforderungen, treten Qualitätsanforderungen und Randbedingungen auf verschiedenen Abstraktionsebenen auf und sollten im Product Backlog dokumentiert, dem Team zugänglich gemacht und in jeder Iteration getestet werden. Daher kann es nützlich sein, Randbedingungen zur Definition of Done hinzuzufügen und deren Validierung in Form von automatisierten Regressionstests zu implementieren.

3.1.9 Akzeptanzkriterien und Abnahmekriterien

Alle Arten von Anforderungen sind nur von begrenztem Wert, wenn ihre Erfüllung nicht validiert, geprüft oder getestet werden kann. Deshalb müssen für alle Anforderungen eine Reihe von testbaren Kriterien vorhanden sein, anhand derer sich prüfen lässt, ob die Anforderung tatsächlich erfüllt wurde. Solche Kriterien erleichtern in der Regel zusätzlich das Verständnis (und unterstützen Feedback), indem sie konkret die Erwartungen beschreiben.

Die Art der verwendeten Kriterien sollte der Granularitätsstufe der Anforderung entsprechen:

- Auf höheren Abstraktionsebenen (Vision, Ziele und Epics) werden gewöhnlich Erfolgskriterien [SAFe] definiert, da man nur messen kann, ob eine benötigte Funktionalität/Fähigkeit bereitgestellt wird oder nicht.
- Auf niedrigeren Abstraktionsebenen können Akzeptanzkriterien verwendet werden, um zu beschreiben, wie die Lösung auf die Anforderung hin geprüft wird, um akzeptiert zu werden.

Nicht-agilen und agilen Methoden ist gemein, dass Anforderungen prüfbar sein müssen. In nicht-agilen Methoden werden oftmals Begriffe wie „Qualitätskriterien", „Abnahmekriterien" oder einfach die guten alten „Testfälle" verwendet; in agilen Prozessen trifft man häufiger

auf Begriffe wie „Akzeptanzkriterien" (für User Storys) oder „Erfolgskriterien" (für Epics oder Themen).

3.1.10 Definition of Ready und Definition of Done

Akzeptanz- und Abnahmekriterien gehören zum Kreis folgender Aspekte und runden diesen ab: Geschäftsanforderungen, Arbeitsprodukte, „Definition of Ready" (DoR) und „Definition of Done" (DoD); sie unterstützen den formalen Prozess der Entwicklung und stellen die Qualität von Anforderungen und Produktinkrementen sicher. Die DoD ist ein offizieller Teil des Scrum Guide [Scrum2020] und wird als Verpflichtung zum Inkrement beschrieben. Sie dient als Quality Gate für den agilen Entwicklungsprozess, während die DoR eine bewährtes Verfahren ist, das die Erstellung nützlicher Anforderungen unterstützt und verhindern soll, dass das Team mit unqualifizierten Anforderungen überladen wird. Die DoR ist hilfreich, um die Anforderungen im richtigen Detaillierungsgrad zu erstellen und ausreichend Informationen für die Abstimmung zwischen Product Owner/Requirements Engineer und den Entwicklern bereitzustellen.

3.1.11 Prototyp vs. Inkremente

Eine andere Möglichkeit des Umgangs mit Anforderungen sind Prototypen, da viele Stakeholder, die Anforderungen an ein System stellen, für die Produktdefinition keine Dokumente schreiben oder lesen wollen – sie möchten stattdessen sofortige Erfolge sehen. Die beste Möglichkeit, um die Bedürfnisse dieser Stakeholder zu verstehen und Feedback zu erhalten, ist die Demonstration von Systemfunktionalitäten oder -Merkmalen in Form eines laufenden Systems.

Dies kann zum einen mit minimalen, inkrementellen Produkten erreicht werden. Im Requirements Engineering stehen dafür zwei Arten zur Verfügung: „horizontale" Produktinkremente, durch die sich mehr Varianten für die Validierung präsentieren lassen („mehr von den richtigen Dingen tun") und „vertikale" Produktinkremente, mit denen geprüft werden kann, ob der Entwicklungsansatz richtig ist („das Richtige tun").

Prototypen können einen sehr guten Beitrag zum Erhalt von Feedback leisten, da sie eine direkte Interaktion mit einem funktionierenden System ermöglichen. Merkmale der Benutzbarkeit, etwa die Reaktionszeit, sind sehr schwer zu spezifizieren, aber ganz leicht zu identifizieren, wenn man funktionierende Software nutzt. Prototypen können jedoch auch eine Frustrationsquelle darstellen – für Benutzer, wenn sie glauben, dass die Entwicklung bereits abgeschlossen ist, und für Entwickler, da sie häufig verworfen und später durch eine bessere Technologie ersetzt werden.

Bei agilen Methoden wird versucht, Prototypen nicht zu verwerfen, indem man sofort Inkremente des „realen" Systems mit einer guten Qualität entwickelt. In einem agilen Umfeld werden kurze Iterationen angestrebt, die nachweisliche Produktinkremente liefern, die wiederum dazu verwendet werden, mehr über die Anforderungen zu erfahren. Auch wenn es nicht die Absicht ist, den entwickelten Code zu verwerfen, so ist das Refactoring doch ein

bewährtes agiles Verfahren, um auf sich ändernde funktionale oder Qualitätsanforderungen basierend auf dem Feedback von Benutzern zu reagieren.

Der Begriff „Spike" wird in agilen Entwicklungsprozessen für eine Entwicklungsiteration verwendet, die den expliziten Zweck hat, einen Komplexitätsbereich (z. B. die Systemarchitektur) zu verstehen und so das Risiko zu verringern. Der Begriff ist zwar nicht präzise definiert, kann sich aber auch auf das Validieren einer Aufgabe oder einer gesamten Iteration beziehen. Das Prototyping ist eine valide Technik innerhalb von Spikes, bei der – anders als bei anderen agilen Iterationen – das primäre Ziel im Sammeln von Wissen besteht, und nicht im Entwickeln von funktionierendem Code.

3.1.12 Zusammenfassung von Arbeitsprodukten

Wie in den letzten Abschnitten ausgeführt, sind einige Arbeitsprodukte für die erfolgreiche Entwicklung sehr wichtig:

- Es ist ein bewährtes Verfahren, mit Visionen oder Zielen zu beginnen.
- Es ist ein bewährtes Verfahren, die wichtigsten Stakeholder immer zu identifizieren und zu kennen.
- Es ist ein bewährtes Verfahren, den Umfang explizit festzulegen und ihn vom Kontext abzugrenzen.

Auch wenn im nicht-agilen und im agilen RE unterschiedliche Begriffe verwendet werden, stimmt man in beiden Richtungen darin überein, dass ein Verständnis der Funktionalitäten ebenso wie der Geschäftsbegriffe notwendig ist, um funktionale Anforderungen zu erfassen (einschließlich Funktionalität und Daten).

Zudem sollten agile und nicht-agile Methoden Qualitätsanforderungen und Randbedingungen aufzeigen, da sie Designentscheidungen stark beeinflussen. Werden diese Qualitätsanforderungen und Randbedingungen ignoriert oder erfährt man zu spät davon, kann das sehr viel Überarbeitungsaufwand und ein Produkt zur Folge haben, das den Erwartungen des Kunden nicht entspricht.

Bei einem guten Requirements Engineering wird sichergestellt, dass keinerlei relevante Aspekte vergessen werden. Bei nicht-agilen Methoden wird deshalb oftmals darauf bestanden, viele potenzielle Interessenbereiche in Bezug auf die Anforderungen für ein System zu dokumentieren.

In agilen Ansätzen nutzt man seltener formale Arbeitsprodukte und ersetzt die fehlende Dokumentation durch direkte Kommunikation und rasche Feedbackschleifen, die durch inkrementelle Produktentwicklung oder Prototypen möglich werden.

Im oft zitierten zweiten Prinzip des Manifests für agile Softwareentwicklung [AgileMan2001] ist genau dargelegt, was wir gerade behandelt haben:

„... Durch diese Arbeit haben wir folgende Werte schätzen gelernt: ... Funktionierende Software bewerten wir höher als umfassende Dokumentation ..."

Wir sind davon überzeugt, dass jede Organisation davon profitiert, bewusste Überlegungen dazu anzustellen, was schriftlich erfasst werden muss, was besprochen und was in Form von

Prototypen oder Inkrementen gezeigt werden kann. Die besten Ergebnisse werden erzielt, wenn Dokumentation, Kommunikation und Prototyping/inkrementelle Entwicklung gemäß den Randbedingungen und der Unternehmenskultur in einem ausgewogenen Verhältnis zueinander stehen. In Kapitel 4 gehen wir näher auf das Thema Ausgleichen von Vorabaktivitäten zu Anforderungen (und der Architektur) und iterativen Aktivitäten ein.

3.2 Techniken in RE@Agile (K2)

In 3.1 haben Sie wichtige Arbeitsprodukte für Anforderungen kennengelernt. In dieser Lerneinheit erfahren Sie mehr über die zentralen Aktivitäten, die im Requirements Engineering auszuführen sind. Das Handbuch zum IREB CPRE Foundation Level [CPREFL2022] strukturiert diese Aktivitäten folgendermaßen:

- Anforderungsermittlung
- Anforderungsdokumentation
- Anforderungsvalidierung und -abstimmung
- Requirements Management

In den folgenden Unterabschnitten werden diese Aktivitäten aus agiler Perspektive beleuchtet.

3.2.1 Anforderungsermittlung

Das Requirements Engineering in agilen Entwicklungsprozessen beruht darauf, dass zur Ermittlung von Anforderungen ein intensiver Austausch zwischen allen Stakeholdern (einschließlich Entwickler) stattfindet. Informelle, direkte Kommunikation zwischen Teammitgliedern kann eine gute Mischung aus Befragungen und Brainstorming sein. Techniken aus dem XP, wie „On-Site Customer", können in Form von Sitzungen zwischen dem Product Owner und Stakeholdern (typischerweise mit den Entwicklern), ebenso erfolgreich sein. Das Ziel besteht in jedem Fall darin, ein eingehenderes Verständnis davon zu gewinnen, was tatsächlich benötigt wird.

Das Requirements Engineering bietet eine weitaus größere Bandbreite an Techniken zur Erkennung und Ermittlung von Anforderungen, als diese normalerweise im Kontext der agilen Entwicklung diskutiert werden. Dazu zählen Frage/Antwort-Techniken (nicht nur Befragungen, sondern auch Fragebögen), Beobachtungstechniken, Artefakt-basierte Techniken (Wiederverwendung, Systemarchäologie usw.) [CPREFL2022] und Kreativitätstechniken wie Brainstorming oder Design Thinking. Diese Techniken unterstützen die Idee, Anforderungen im User Story Format zu erfassen, welche mehr die Basis für strukturierte Gespräche sind als eine Vorgabe für die Implementierung.

Wie in Abschnitt 3.1.11 ausgeführt, sind Prototypen und Produktinkremente eine weitere Möglichkeit, um mehr über Anforderungen zu erfahren. Sobald ein Produktinkrement in einem Sprint Review oder in einem Demo-Meeting präsentiert wird, können neue Ideen entstehen, die direkt in das Product Backlog übernommen und vom Product Owner priorisiert werden können.

Das Requirements Engineering in agilen Entwicklungsprozessen kann in hohem Maße vom Requirements Engineering in nicht agilen Projekten und Produktentwicklungen profitieren, indem man verschiedene Ermittlungstechniken prüft und eine bewusste Entscheidung trifft, welche Mischung aus Techniken genutzt werden soll. Intensive Kommunikation zwischen den Stakeholdern und schnelles Feedback durch Produktinkremente sind zwar hervorragende Ideen, doch hinter der Ermittlung von Anforderungen steckt noch mehr. Wenn es beispielsweise Hunderte oder Tausende von Stakeholdern gibt, ist ausschließlich verbale Kommunikation nicht ausreichend. Wenn man versucht, innovative Anforderungen, oder Anforderungen bei denen sich die Stakeholder der Möglichkeit nicht einmal bewusst sind, ans Tageslicht zu bringen, ist unter Umständen der Einsatz von Kreativitätstechniken erforderlich.

Arbeitet man mit großen zeitlichen Einschränkungen (d. h. es ist nicht genügend Zeit für intensive Gespräche), dann sind Techniken wie Snowcards womöglich am effektivsten. Beim Erschließen neuer Technologien sind Spikes (einfache Inkremente in Timeboxes zum Erschließen potenzieller Lösungen) eine hervorragende Option.

3.2.2 Anforderungsdokumentation

In agilen Entwicklungsprozessen werden Anforderungen in einem Backlog organisiert. Anforderungen können in Form von Story-Cards dokumentiert und mit Anmerkungen zum Wert und der Priorität versehen werden. Diese lassen sich in Story-Maps organisieren oder in einfachere Storys zerlegen. Das Prinzip hinter den Story-Cards ist, dass die notierten Informationen durch die Größe der Karten beschränkt sind, was eine Konzentration auf die grundlegenden Details erleichtert.

Das Dokumentieren von Anforderungen wird nach wie vor als eine wichtige Aktivität zur Förderung der Kommunikation zwischen allen Stakeholdern erachtet. Der Formalismus für die Dokumentation lässt sich minimieren, indem Details verbal kommuniziert werden.

Das Definieren eines adäquaten Maßes an Dokumentation hängt von zahlreichen Faktoren ab: etwa vom Umfang der Projekte bzw. Produkte, von der Anzahl der beteiligten Stakeholder, von rechtlichen Randbedingungen oder der Sicherheitskritikalität. Basierend auf diesen Faktoren versucht man in agilen Entwicklungsprozessen, ein Übermaß an Dokumentation zu vermeiden und einen minimalen Nenner an nützlichen Dokumentationsinhalten zu finden.

Das Arbeiten mit einem „lebenden" Product Backlog ist zwar eine effiziente Weise des Umgangs mit Dokumentation, doch sie ist nicht immer ausreichend. Lassen Sie uns also noch andere Arten der Dokumentation betrachten.

Aus RE-Perspektive unterscheiden wir vier Arten von Dokumentation:

1. **Dokumentation für gesetzliche Zwecke:** Bestimmte Domänen oder Projektkontexte (z. B. Software im Gesundheitssektor oder in der Avionik) erfordern die Dokumentation bestimmter Informationen (z. B. Anforderungen und Testfälle für ein System) für eine bestimmte Zielgruppe, damit eine gesetzliche Genehmigung erteilt wird.

Für die Erstellung von Dokumentation für gesetzliche Zwecke gilt folgendes Prinzip: Gesetzlich erforderliche Dokumentation muss aus den entsprechenden Gesetzen oder Standards abgeleitet werden und ist untrennbarer Bestandteil des Produkts.

2. **Dokumentation zum Zwecke der Bewahrung:** Bestimmte Informationen über ein System haben einen über den initialen Entwicklungsaufwand hinaus anhaltenden Wert. Beispiele dafür sind die Ziele, für deren Erfüllung das System aufgebaut wurde, die zentralen Anwendungsfälle, die es unterstützt oder Entscheidungen, die während der Entwicklung getroffen wurden, beispielsweise Gründe für das Ausschließen bestimmter Funktionalitäten. Die Dokumentation zu Bewahrungszwecken kann zum gemeinsamen Archiv des Teams, eines Produkts oder einer Organisation werden. Sie gewährleistet Unabhängigkeit von der Erinnerung einzelner Teammitglieder, und sie kann Gespräche über vorherige Entscheidungen vereinfachen (z. B. „Warum haben wir uns zu dieser Implementierung entschlossen?").

Das Prinzip ist Folgendes: Das Team entscheidet, was zum Zwecke der Bewahrung dokumentiert wird.

3. **Dokumentation für Kommunikationszwecke:** Effektive und effiziente Kommunikation ist bei agilen Methoden ein wichtiges Werkzeug, da sie interaktiv ist und kurze Feedbackzyklen ermöglicht. In der Praxis gibt es verschiedene Konstellationen, die eine unmittelbare verbale Kommunikation verhindern: geografisch verteilte Teams, Sprachbarrieren oder zeitliche Einschränkungen bei den Beteiligten. Darüber hinaus sind Informationen manchmal so komplex, dass direkte Kommunikation ineffizient oder irreführend sein kann. Ein Prototyp in Papierform oder ein Diagramm eines komplizierten Algorithmus können später beispielsweise noch einmal gelesen werden. Manchmal ziehen Stakeholder einfach schriftliche Kommunikation dem Lesen von Quellcode oder dem Prüfen von Software vor. In diesen Fällen vereinfacht Dokumentation den Kommunikationsprozess zwischen allen Beteiligten und bewahrt die Ergebnisse.

Für das Erstellen von Dokumentation für Kommunikationszwecke gilt folgendes Prinzip: Ein Dokument wird als zusätzliches Kommunikationsmittel erstellt, wenn Stakeholder oder die Entwickler dem Vorhandensein der Dokumentation einen Wert beimessen. Verlief die Kommunikation erfolgreich, sollte das Dokument archiviert werden.

4. **Dokumentation zum Zwecke von Überlegungen:** Es gibt einen Aspekt beim Schreiben eines Dokuments, der häufig vergessen wird: Der Schreibvorgang ist immer ein gutes Mittel zur Verbesserung und Unterstützung der gedanklichen Prozesse des Schreibenden. Selbst wenn das Dokument zu einem späteren Zeitpunkt im Prozess verworfen wird, bleibt der Vorteil der Verbesserung und Unterstützung des Denkvorgangs bestehen. Das Schreiben eines Anwendungsfalls beispielsweise zwingt den Schreibenden, über konkrete Interaktionen zwischen dem System und den Akteuren nachzudenken, einschließlich z. B. Ausnahmen und alternativen Szenarios. Aus diesem Grund kann das Schreiben eines Anwendungsfalls als Werkzeug zum Testen des eigenen Wissens und Verständnisses eines Systems betrachtet werden.

Für das Erstellen von Dokumentation als Teil des Erkenntnisprozesses gilt folgendes Prinzip: Die nachdenkende Person entscheidet über die Dokumentform, die ihre Denkvorgänge am besten unterstützt. Die nachdenkende Person muss ihre Wahl der Dokumentationsform für den Denkvorgang nicht rechtfertigen. Das Dokument kann verworfen werden, wenn der Überlegungsprozess abgeschlossen ist.

Agile Methoden können erheblich davon profitieren, wenn diese vier Arten der Dokumentation im richtigen Kontext identifiziert und eingesetzt werden. Zusammenfassend lässt sich festhalten, dass das Dokumentieren von Anforderungen kein Selbstzweck ist, sondern dass es die Kommunikation zwischen Stakeholdern erleichtern sollte, vor allem zwischen dem Anfordernden (häufig vertreten durch den Product Owner) und den Entwicklern.

3.2.3 Anforderungsvalidierung und -abstimmung

Beim Requirements Engineering liegt der Schwerpunkt der Anforderungsvalidierung auf Methoden wie Reviews, Walkthroughs, Inspektionen oder perspektivenbasiertem Lesen, agile Methoden sind hingegen bestrebt, Anforderungen durch frühes und häufiges Feedback zu werthaltigen Produktinkrementen zu validieren. Ein bewährtes, unterstützendes Verfahren dafür sind automatisierte Regressionstests, die eine kontinuierliche Validierung der Entwicklung und der damit verbundenen Anforderungen ermöglichen. Zum Ziel der Anforderungsvalidierung zählt zudem das Identifizieren fehlender, mehrdeutiger oder falscher Anforderungen sowie strittiger oder widersprüchlicher Anforderungen, bei denen Abstimmungen und Konfliktlösungstechniken angewandt werden können.

Da die iterative, inkrementelle Entwicklung bei agilen Methoden eine zentrale Strategie ist, sinkt der Bedarf an einer formalen Validierung von Dokumenten. Sie wird durch eine konstante Abstimmung unter allen Stakeholdern zu den Anforderungen ersetzt, sodass Konflikte frühzeitig erkannt und gelöst werden.

Die formale Validierung wird auch dadurch reduziert, dass schnelle Ergebnisse in Form von integrierten Produktinkrementen vorliegen. Wenn das Inkrement nicht allen Anforderungen sämtlicher Stakeholder entspricht, wird das Delta in Form von neuen Anforderungen in das Product Backlog aufgenommen und zusammen mit allen anderen Backlog Items bewertet und priorisiert.

Dennoch sind Walkthroughs des Product Backlogs, Besprechungen des Business Values, Besprechungen von Risiken sowie sofortige Abstimmungen von Anforderungen allesamt nützliche Techniken in RE@Agile. All diese Techniken können in Verfeinerungsmeetings angewendet werden, wenn der Product Owner und die Entwickler (und Stakeholder, wenn verfügbar) zusammenarbeiten, um den benötigten Detaillierungsgrad für die Implementierung mithilfe der Prinzipien kontinuierlicher Verfeinerung herauszuarbeiten.

In den ersten Versionen des Scrum Guide wurde das sogenannte „Product Backlog Refinement Meeting", das Verfeinerungsmeeting, nur indirekt erwähnt. In der aktuellen Version [Scrum2020] wird das Refinement explizit als Aktivität für die Ermittlung, Dokumentation und die Validierung von Anforderungen genannt. Dieser Refinementvorgang, bei dem ein Refinementmeetings (Verfeinerungsmeetings) ein essentieller Bestandteil ist, deckt potentielle Probleme frühzeitig auf und verringert später die benötigte Zeit für das Sprint-Planungsmeeting.

3.2.4 Requirements Management

Im herkömmlichen RE beschäftigt sich das Requirements Management (Anforderungsmanagement) mit allen Aktivitäten zur Bearbeitung von Anforderungen im Zeitverlauf. Dies beinhaltet Versionsmanagement, Änderungsmanagement, Konfigurationsmanagement, Verfolgbarkeit und auch das Hinzufügen von Attributen wie Status, Schätzungen, Prioritäten, Verknüpfungen zu widersprüchlichen Anforderungen sowie von Personen, die mit der Erfassung, Prüfung, Unterzeichnung, Implementierung oder dem Testen der Anforderung befasst sind.

Wie bereits angemerkt (siehe 3.1.1), werden die zentralen Arbeitsprodukte für die Bearbeitung von Anforderungen in agilen Entwicklungsprozessen in einem Backlog zusammengefasst. Anders als beim herkömmlichen Requirements Management, ist das Backlog so konzipiert, dass nur die neueste und beste Version aller Anforderungen beibehalten wird, die bis dahin implementiert werden soll. Backlog Items werden typischerweise gelöscht bzw. archiviert, sobald das Produkt, das diese Anforderungen erfüllt, ausgeliefert wird.

Es gibt folgende Aktivitäten des Requirements Managements, die im Backlog stattfinden:

1. **Priorisierung von Anforderungen:** Bestimmung des Werts (Business Value), um zu entscheiden, wann diese Anforderungen implementiert werden sollen. Je höher der Business Value ist, umso höher ist die Priorität der Anforderung, da in agilen Entwicklungsprojekten versucht wird, die Elemente mit dem höchsten Business Value zuerst auszuliefern. Die Einflussfaktoren die den Business Value sowie die daraus abgeleitete Priorisierung bestimmen werden im CPRE Modul RE@Agile Practitioner und Specialist [CPREALAGILE2022] behandelt.
2. **Einschätzung von Anforderungen:** Bestimmen, wie viel Arbeit mit der Erfüllung der Anforderungen in Verbindung steht. Allzu umfangreiche Schätzungen sind für einen Product Owner ein klares Zeichen dafür, dass mehr Arbeit erforderlich ist, bis der Zustand der Definition of Ready (DoR, siehe 2) erreicht ist. Er muss das jeweilige Item also in kleinere Einheiten zerteilen um eine Schätzung zu ermöglichen.

Das soll nicht heißen, dass andere Aktivitäten, die mit den historischen Aspekten des Requirements Managements in Verbindung stehen, in agilen Entwicklungsprozessen nicht stattfinden können. Solche Aktivitäten werden jedoch typischerweise außerhalb des Backlogs aufgezeichnet.

Wenn der Requirements Engineer entscheidet, welche Aktivitäten des Requirements Managements in einem bestimmten Kontext angebracht sind, sollte er einen Ausgleich anstreben: zwischen der Minimierung des Verwaltungsaufwands, der Möglichkeit einer frühen Auslieferung funktionierender Lösungen und den längerfristigen Bedürfnissen der Organisation, etwa die Einhaltung gesetzlicher Vorgaben, Betriebsdokumentation oder Übergaben an neue Mitglieder des Teams.

Fazit

Aktivitäten in Verbindung mit Anforderungen wie Ermittlung, Dokumentation, Validierung sowie das Requirements Management müssen in agilen Entwicklungsprozessen weiterhin generell ausgeführt werden. Die bevorzugten Techniken zur Ermittlung und Dokumentation können bei agiler und nicht-agiler Entwicklung unterschiedlich sein, doch das Lernen von der jeweils anderen Richtung führt zur besten Lösung, da es die Stärken kombiniert und die Verschwendung oder den Verwaltungsaufwand reduziert. Diese Arbeitsweise steht für die fünf Werte von Scrum (Commitment, Fokus, Offenheit, Respekt und Mut) und deren Darstellung in den drei Säulen von Scrum (Transparenz, Überprüfung und Anpassung) [Scrum2020].

Vor allen Dingen sind Offenheit und Respekt hilfreich, wenn das handwerkliche Können des RE in die Welt der Agilität eingebracht wird, und umgekehrt die Ideen und Prinzipien agiler Entwicklungsprozesse in die RE-Handwerkskunst.

In dieser Lerneinheit haben Sie erfahren, dass es selbst in RE@Agile mehr Arbeitsprodukte gibt als nur User Storys im Product Backlog, und dass die entscheidenden Requirements-Engineering-Aktivitäten nicht in Vergessenheit geraten sollten – sondern basierend auf den in 1 erläuterten agilen Prinzipien mit anderem Fokus und anderen Methoden durchgeführt werden können.

4 Organisatorische Aspekte von RE@Agile (K2)

Dauer: 1,5 Stunden

Lernziele

LZ 4.1.1 Zusammenspiel zwischen Organisationsstruktur und RE@Agile verstehen
LZ 4.2.1 Interaktion mit Stakeholdern in agilen Requirements-Engineering-Prozessen kennen
LZ 4.2.2 Wissen, wie Kommunikation und Zusammenarbeit zur Verbesserung von Ergebnissen beitragen können
LZ 4.2.3 Rolle des Managements in agilen Entwicklungsprozessen kennen
LZ 4.3.1 Motivation für Skalierung kennen
LZ 4.3.2 Dimensionen für das Organisieren von Teams kennen
LZ 4.3.3 Ansätze für das Organisieren der Kommunikation zwischen Teams kennen
LZ 4.3.4 Beispiel-Frameworks für Skalierung kennen
LZ 4.3.5 Wichtigste Auswirkungen von Skalierung auf das RE kennen
LZ 4.4.1 Kriterien kennen, um über das Level von Vorab-RE vs. kontinuierlichem Requirements Engineering zu entscheiden
LZ 4.4.2 Richtigen Detaillierungsgrad für Backlog Items kennen
LZ 4.4.3 Wert von Validierung in RE@Agile kennen
LZ 4.4.4 Richtige Aktualisierungszyklen für das Product Backlog kennen
LZ 4.4.5 Wissen, wie man den richtigen Zeitplan für den Entwicklungszyklus findet

4.1 Der Einfluss von Organisationen auf RE@Agile (K2)

Agile Entwicklungsprozesse gehen auf die herstellende/verarbeitende Industrie und die empirische Prozesssteuerung zurück (siehe [TaNo1986]). Agile Prinzipien sind leicht verständlich, und agile Verfahren und Frameworks wie zum Beispiel Scrum lassen sich in Greenfield-Umgebungen wie Startup- oder kleinen Unternehmen einfach einsetzen. In größeren Organisationen, die sich wie lebende Organismen verhalten, die jeden „Eindringling" abwehren, sind sie dagegen schwer zu implementieren. Da Organisationen die Vorteile der Anwendung agiler Prinzipien und Verfahren zunehmend erkennen, versuchen sie diese zu integrieren, wie im Gesetz von Conway [Conw1968] beschrieben. Dies resultiert in einem wachsenden Interesse an Themen wie agilem Management (siehe [ACP/PMI]) und agilen Organisationen (siehe [Denn2015], [Appel2011]), wodurch Gespräche über Agilität angeregt werden, die häufig über die (Software-) Entwicklung hinausgehen.

Die Ideen, den Kunden in den Mittelpunkt zu stellen, von sich selbst organisierenden Teams, vom Ermöglichen und Ermächtigen von Einzelpersonen und von kontinuierlicher Verbesserung sind auch in einem weiter gefassten geschäftlichen Kontext von Bedeutung. Das Scrum-at-Scale-Framework ist eine minimale Erweiterung des zentralen Scrum-Frameworks, das die modulare Struktur im Kern des Scrum-Frameworks beibehält und zudem erlaubt, dass sich Scrum-Implementierungen maßgeschneidert auf die einzigartigen Bedürfnisse des Unternehmens skalieren lassen.

In der Welt der Softwareentwicklung scheitern viele Implementierungen von agilen Entwicklungsprozessen in der Konzeption, da der Rest der Organisation nicht zu Veränderungen in der Lage ist, die zur Unterstützung der agilen Teams erforderlich wären.

Warum ist eine organisatorische Veränderung erforderlich?

Sehen wir uns zwei Beispiele dafür an, weshalb auch das übrige Unternehmen sich ändern muss, damit agile Entwicklungsprozesse unterstützt werden können:

1. Die nachfragende (Unter-) Organisation muss genügend gute Anforderungen liefern („gut" bedeutet hinreichend aber nicht zu sehr detailliert – gemäß der Definition of Ready), damit die Entwicklung in gleichmäßigem Tempo voranschreiten kann. Bei Anforderungen, die sich häufig ändern, müssen dafür kontinuierlich Bedarfe weitergegeben werden.
2. Die Personalabteilung muss verstehen, welche Mitarbeiter sie einstellen muss, damit agile Teams die geeignete Unterstützung erhalten. Schlechte Beispiele stellen immer wieder Stellenausschreibungen für Scrum Master dar, in denen Programmierkenntnisse und -zertifikate vorausgesetzt werden, was von einem falschen Verständnis der Prinzipien der drei Scrum-Rollen zeugt.

In 4.2 erläutern wir die organisatorischen Aspekte der Integration einer agilen Organisationseinheit, z. B. ein Scrum-Team, in nicht-agilen Umgebungen. Auch wenn agile Prozesse bereits in der Produktentwicklung eingeführt werden, arbeitet in vielen Fällen nicht unbedingt dann auch das gesamte Unternehmen nach agilen Prinzipien.

Die Auswirkungen, die es auf die Organisation hat, wenn mehrere agile Teams zur Lösung eines komplexen Problems benötigt werden, besprechen wir in 4.3. Das Hauptaugenmerk richten wir dabei wieder auf die IT-Organisation und deren Schnittstellen zum übrigen Geschäft. Der Übergang eines Unternehmens in eine durchgängig agile Organisation wird hier nicht explizit betrachtet, da dies über den Rahmen der Behandlung des RE hinausgehen würde.

In 4.4 liegt der Schwerpunkt auf den organisatorischen Aspekten des Timings, wo vor allem die Frage analysiert wird, wann RE-Aktivitäten stattfinden sollten.

4.2 Agile Entwicklung in einer nicht-agilen Umgebung (K1)

4.2.1 Interaktion mit Stakeholdern außerhalb der IT-Organisation

Es ist Aufgabe der Entwicklungsorganisation in einem Unternehmen, Lösungen und Services für Unternehmenskunden bereitzustellen (sowohl innerhalb als auch außerhalb der Organisation).

Agile Entwicklungsprozesse stellen den Kunden in den Mittelpunkt der Produktentwicklung. Das bedeutet, dass der Kunde während des gesamten Lebenszyklus der Produktentwicklung einbezogen wird, dass aktiv Feedback zu gelieferten Inkrementen eingeholt wird und neue

Anforderungen aufgenommen werden, die in Einklang mit den Anforderungen des Unternehmens stehen.

Durch die laufende Einbeziehung des Kunden wird auch das RE zu einem kontinuierlichen Prozess (siehe 2.4). Die verantwortliche Person, beispielsweise der Product Owner, sollte sich in Form einer offenen und direkten Kommunikation an die Kunden wenden, neue Bedürfnisse und veränderte Erwartungen erfragen und diese in einem Backlog erfassen.

In der Praxis kann diese Kommunikation auf verschiedene Weise stattfinden. Hat der Kunde tatsächlich täglich Zeit für die Zusammenarbeit mit dem Team, so kann die Kommunikation de facto direkt und überwiegend informell stattfinden, und es müssen nur Ergebnisse oder Entscheidungen dokumentiert werden.

Es ist wichtig, dass auch Faktoren berücksichtigt werden, die außerhalb der Kontrolle der Entwickler liegen (z. B. geografische Verteilung der Mitarbeiter oder einfach mangelnde Verfügbarkeit), und die bedeuten, dass eine direkte Interaktion nicht immer praktikabel ist. Dafür sollten effizientere Formen der Kommunikation geplant werden, entweder indem man Kundenvertreter zu regelmäßigen Planungs- und Review-Meetings einlädt oder bereits in einer frühen Phase intensive Sitzungen in Timeboxes durchführt (z. B. Design Thinking oder Design Sprint, siehe 1.5) und die Ergebnisse im Backlog erfasst.

4.2.2 Produkt- vs. Projektorganisation

Agilität verbessert direkte, offene und nicht-hierarchische Beziehungen innerhalb der Organisation und ermöglichen mehr Flexibilität bei der Entwicklung von Produkten im Laufe der Zeit. In größeren Unternehmen, die traditionell in Top-down-Managementstrukturen geführt werden, wird großer Wert auf Planung und Berechenbarkeit gelegt. Die Projekt- und Ressourcenplanung sind beispielsweise Gegebenheiten, denen sich ein Unternehmen stellen muss.

Die Softwareentwicklung ist von jeher oft projektbasiert, das heißt, sie findet in Form einer Reihe von vorübergehenden Verpflichtungen zur Herstellung von eindeutigen Produkten, Services oder Ergebnissen statt (siehe [PMI]). Gruppen zusammenhängender Projekte mit einem gemeinsamen Zweck oder Ziel werden in der Regel als Programme bezeichnet, die Planung und Steuerung der Gesamtheit von Projekten und Programmen innerhalb einer Organisation als Portfoliomanagement.

Getreu seiner Wurzeln in der Produktentwicklung, ist der agile Ansatz stärker produktorientiert. Dabei wird ein Backlog mit Produktverbesserungen bearbeitet, die in einem iterativen Prozess kontinuierlicher Entwicklung implementiert werden. Agilität an sich definiert allerdings kein eigentliches Enddatum. So lange noch Verbesserungen ausstehen oder Vorteile zu realisieren sind, sollte die Arbeit im Prinzip fortgeführt werden, wenn die Vorteile gegenüber dem Aufwand bzw. den Kosten überwiegen.

Diese Ansätze schließen sich zwar nicht gegenseitig aus (im Umfang eines Projekts kann die Auslieferung eines bestimmten Produkts enthalten sein, für spätere Verbesserungen werden dann zusätzliche Projekte aufgesetzt), die Unterschiede in der Perspektive oder

Terminologie können jedoch Spannungen und Missverständnisse zwischen agiler Softwareentwicklung und nicht-agilen Organisationen verursachen.

Ein Ansatz zur Auflösung solcher Spannungen ist, dass die eigentliche Softwareentwicklung zwar strikt nach agilen Prinzipien erfolgt, die Funktionen des Portfolio- und Programmmanagements dagegen ein höheres Level an Planung und Kontrolle bieten, für die Ansätze aus beiden Richtungen verwendet werden (siehe auch 4.3). Ein solcher Ansatz funktioniert nur, wenn konzeptuelle Lücken zwischen der geplanten Erfüllung von Geschäftszielen und Funktionalitäten auf Portfolio- und Programmlevel sowie einer iterativen und flexiblen Lieferung einzelner Softwarefunktionalitäten überbrückt werden können.

Das RE stellt Konzepte und Methoden zur Differenzierung von Anforderungen auf diesen unterschiedlichen Abstraktionsebenen mit Anforderungen auf Geschäftsebene auf den Portfolio- und Programmebenen und abgeleiteten, detaillierten Softwareanforderungen, die für das Entwicklungs-Backlog geeignet sind, bereit. Die Herangehensweise an Anforderungen verschiebt sich von detaillierteren und präziseren Anforderungen, die als exakte Aufträge für die Entwicklung genutzt werden, hin zu Anforderungen, die als allgemeine Basis für Besprechungen und „Just-in-Time"-Anpassungen verwendet werden, die so detailliert sind, wie sie für das aktuelle Planungslevel gerade benötigt werden.

4.2.3 Die Rolle des Managements in einem agilen Kontext

Disziplinen und Teams: Mitarbeiter in den IT-Abteilungen sind traditionell nach Disziplinen organisiert: Entwickler, Tester, Requirements Engineers, Business-Analysten, Projektmanager usw. Aus diesen verschiedenen Kompetenzen werden für die feste Dauer des Vorhabens Projektteams zusammengestellt. Agile Methoden, insbesondere Scrum, verfolgen die Idee von verstärkt cross-funktionalen Teams. Abgesehen von den Expertenrollen von Product Owner und Scrum Master sollten alle Teammitglieder in der Lage sein, in unterschiedlichen Kompetenzbereichen zu arbeiten, mit den verschiedenen Einzelpersonen, die Requirements-Engineering- oder Testaktivitäten unterstützen. Auch aus diesem Grund werden diese Teammitglieder in Scrum allgemein „Entwickler" genannt. Ziel des Teams ist es, Kundenanforderungen vollständig zu liefern, unabhängig davon, welche technologischen oder organisatorischen Elemente daran beteiligt sind. Das ist möglich, wenn RE-Kompetenzen im Team (bevorzugte Lösung) oder außerhalb des Teams vorhanden sind (in der Praxis häufig als Unterstützung für den Product Owner genutzt), die dann offiziell nicht Bestandteil des Scrum-Frameworks sind.

IT-Manager: Es ist die Aufgabe des IT-Managers (in [SAFe] als „People Developer", also Personalentwickler, bezeichnet), in seiner Organisation das richtige Gleichgewicht zwischen den Kompetenzen von Spezialisten und Generalisten zu finden, und Teams dabei zu unterstützen, diese Kompetenzen in einer angemessenen Anzahl von Teams zu organisieren. Nach dem Gesetz von Conway [Conw1968] ist die Teamstruktur ein Spiegel der Produktstruktur (Komponenten) oder in der IT der Systemstruktur. Dies gewinnt bei der Skalierung der Anzahl der Teams noch mehr an Bedeutung. Wenn ein Unternehmen seine Teams nach Komponenten oder Systemen organisiert, ist eine Skalierung durch Steigerung

der Anzahl von Teams nicht hilfreich, da dies noch mehr Abhängigkeiten erzeugt. Skalierung wäre nur in Bezug auf die Anzahl von Teammitgliedern möglich, aber auch nur bis zu einem gewissen Punkt, da der Kommunikationsaufwand dann ebenfalls erheblich steigt.

Cross-funktionale Teams, die alle Fähigkeiten zur Auslieferung vollständiger Inkremente vom Frontend bis zum Backend übernehmen, lassen sich leicht skalieren – doch der Aufbau solcher Teams ist nicht einfach und kann sehr zeitintensiv sein.

Product Owner vs. Projektmanager: Wie bereits besprochen, tragen Product Owner mehr Verantwortung als Requirements Engineers; sie übernehmen die Verantwortung für geschäftliche Prioritäten, die in den Anforderungen an die Produktentwicklungsteams gerichtet wurden. Product Owner müssen befähigt (Wissen) und dazu ermächtigt sein, Geschäftsentscheidungen zu treffen. Einige Entscheidungen, die zuvor von Projektmanagern getroffen wurden, sind in agilen Ansätzen nicht mehr erforderlich, da die Entwickler sich in ihren organisatorischen Grenzen selbst organisieren, und nur die Arbeit tun müssen (Anforderungen in einem Detaillierungsgrad, der die Entwicklung in einer Iteration erlaubt), um ihre Arbeit selbst zu organisieren (Aufgliederung und Zuweisung der Aufgabe innerhalb des Teams).

In einem agilen Umfeld müssen IT-Projektmanager ein klares Setting der Vision für die agile Entwicklung entwerfen und die kulturellen Voraussetzungen klar kommunizieren, damit dieser Ansatz erfolgreich sein kann.

Das richtige Gleichgewicht zu finden und gleichzeitig die Erwartungen des geschäftlichen Umfelds zu erfüllen ist nicht einfach. Einige wichtige Kriterien für den Erfolg werden nachfolgend in 4.4 erläutert.

Die Relevanz von Requirements Engineering (RE): Die zuvor genannten Muster gelten beispielsweise auch für Scrum. Personen, die nach RE-Prinzipien arbeiten, können direkt als Teil des Teams existieren und werden daher nicht extra aufgeführt. Alternativ können sie selbst ein Team bilden und einen oder mehrere Product Owner als ein Team unterstützen. Beide Ansätze haben Vorteile und können zusammen genutzt werden, ohne agile Prinzipien zu verletzen. Das RE wird zum Backbone erfolgreicher agiler Entwicklungsprozesse werden.

4.3 Der Umgang mit komplexen Problemen durch Skalierung (K1)

4.3.1 Motivation für die Skalierung

Die agilen Werte direkter, täglicher, nicht-hierarchischer Kommunikation spiegeln sich typischerweise in kleinen, fest miteinander verwachsenen Teams wider, etwa Scrum-Teams, für die nicht mehr als 10 Scrum-Teammitglieder empfohlen werden (Entwickler + Product Owner + Scrum Master). Die Teammitglieder sollten sich im Idealfall physisch am selben Standort befinden und in Bezug auf betriebswirtschaftliche und technische Kompetenzen cross-funktional aufgestellt sein. In größeren Organisationen ist diese ideale Sicht auf die agile Entwicklung möglicherweise aus folgenden Gründen nicht umsetzbar:

- Bei komplexen Problemen kann die Einbeziehung von Stakeholdern und Wissen aus verschiedenen Unternehmensbereichen erforderlich sein, die sich in einem einzigen Team nicht so einfach vereinen lassen.
- Bei komplexen Problemen kann zudem die Einbeziehung einer Reihe von technischen Experten und Wissen erforderlich sein, die sich in einem einzigen Team nicht so einfach vereinen lassen.
- Der bis zu einem festgelegten Einführungstermin erforderliche Umfang geht über das mit der erreichbaren Geschwindigkeit eines einzelnen Teams Mögliche hinaus.
- Mitarbeiter in globalen Unternehmen arbeiten unter Umständen an verschiedenen geografischen Standorten.

Der Begriff „Scaled Agile" (skalierte Agilität), wird zur Beschreibung von Situationen verwendet, in denen mehrere Teams zusammen an einem Produkt/einer Lösung mit gemeinsamen Zielen arbeiten. Bei Scaled-Agile-Ansätzen müssen Entscheidungen zur Organisation von Teams getroffen werden und dazu, wie die Kommunikation unter den Teams koordiniert werden soll. Ziel dabei ist es, einen effizienten Ansatz für den Umgang mit komplexen Problemen zu erreichen und dabei möglichst viele Vorteile agiler Methoden beizubehalten.

4.3.2 Ansätze für das Organisieren von Teams

Hier ist lediglich die Frage, wie die Organisation sich selbst in Teams mit einem Umfang organisieren soll, der eine cross-funktionale Zusammensetzung (Mindestgröße) nicht ausschließt, dem Team aber gleichzeitig noch eine effiziente (maximale Größe) Kommunikation und Abstimmung erlaubt. Das Organisieren nach funktionalen Linien (z. B. ein Team, das sich auf einen definierten Geschäftsbereich oder sogar eine Funktion innerhalb eines Geschäftsbereichs spezialisiert) hat den Vorteil, dass sich das betriebswirtschaftliche Wissen in einem einzigen Team konzentriert. Das kann die Ermittlung von Anforderungen erleichtern, beispielsweise indem die Anzahl der direkt in das Team einbezogenen geschäftlichen Stakeholder reduziert wird.

Ein Nachteil dieses Ansatzes ist, dass die Bereitstellung von umfassenden Funktionalitäten in einem Geschäftsbereich sehr wahrscheinlich unterschiedliche technische Besonderheiten involviert, wie UI-Design, Prozessmodule, Datenbanken und zentrale Plattformen wie ein ERP-System oder Mainframes, durch die man leicht an die Grenzen der maximalen Größe eines effizienten Teams stößt.

Alternativ ließe sich die Entwicklung auch nach technischen Linien organisieren, d. h. in Teams, die sich etwa auf technische Komponenten oder Plattformen spezialisieren. Der Vorteil von nach Komponenten oder Plattformen organisierten Teams ist, dass diese über tiefgehende Kenntnisse in ihren jeweiligen technologischen Gebieten verfügen. Ein Nachteil dieses Ansatzes sind die Abhängigkeiten, die zwischen den Teams entstehen, die zusammenarbeiten, um das gesamte Produktinkrement termingerecht zu liefern,

Die Skalierungs-Frameworks (z. B. in 4.3.4) zeigen, wie man mit dieser Situation umgehen kann. Requirements-Engineering-Aktivitäten müssen auf das Framework ausgerichtet werden, um eine gemeinsame Sicht auf die lieferbaren Ergebnisse zu erreichen.

In diesem Szenario spielt das RE eine besonders wichtige Rolle: zuerst bei der Aufgliederung der Geschäftsziele und Anforderungen in einzelne Untersystemanforderungen, die dann Teams zugewiesen werden können, und zweitens bei der Begleitung der Entwicklung, um sicherzustellen, dass das Ergebnis einer integrierten Lösung erreicht und damit der höchste Business Value geliefert wird.

Die beste und pragmatischste Lösung kann eine Mischung aus allen Arten von Teams darstellen, um von den Ideen sog. Feature-Teams zu profitieren und gleichzeitig unternehmensspezifische Einschränkungen zu berücksichtigen (für Details siehe [CPREALAGILE2022]).

4.3.3 Ansätze für das Organisieren der Kommunikation

Zur Beantwortung der Frage, wie viele Teams effizient zusammenarbeiten können, unterscheiden wir zwischen zwei verschiedenen Ansätzen:

1. Methoden aus dem Vorgehen mit einem Team verwenden
2. Zusätzliche Konzepte zur Organisation der Kommunikation und Verantwortlichkeiten einführen

Methoden aus dem Vorgehen mit einem Team verwenden: Der erste Ansatz verfolgt die Idee, dass keine zusätzlichen Arbeitsprodukte und Rollen erforderlich sind, und die Kommunikation zwischen mehreren Teams mit den verfügbaren Techniken von Ein-Team-Ansätzen unterstützt werden sollte. Zusätzliche Rollen und Arbeitsprodukte würden einer agilen Denkweise widersprechen und zu noch mehr Komplexität in der Organisation führen. Nach dem grundlegenden Prinzip „keep it simple" (einfach halten) sollte durch die neuen Rollen kein Verwaltungsaufwand erzeugt werden. Die Kommunikation und Koordination zwischen den Teams wird gewöhnlich durch die Product Owner der Teams initiiert, dann jedoch von den Teamvertretern ausgeführt. Über Konstruktionen wie „Communities of Practice" können die Teammitglieder über die Teams hinweg Erfahrungen austauschen und Gesamtprozesse koordinieren.

Zusätzliche Konzepte einführen: Dieser zweite Ansatz empfiehlt, größere Probleme in kleinere Probleme zu unterteilen und Verantwortlichkeiten durch unterschiedliche Rollen für verschiedene Abstraktionsebenen zu managen. Durch das Unterteilen sollten weitere Arbeitsprodukte für die unterschiedlichen Abstraktionsebenen eingeführt werden (z. B. Business Epics, Architectural Epics, Investment-Themen, Features, User Storys).

Abhängig von dem verwendeten Framework werden auch noch weitere Rollen mit Verantwortung für die unterschiedlichen Abstraktionsebenen eingeführt (z. B. Portfolio-Manager, Produktmanager, Chief Product Owner). Aufgrund der steigenden Komplexität sind außerdem zusätzliche Arbeitsprodukte und Rollen erforderlich, um die Planung und Kommunikation unter den verschiedenen Teams zu managen und integrierte Ergebnisse für jede Iteration zu erzielen (z. B. Roadmaps und Release-Manager). Darüber hinaus müssen spezielle Meetings organisiert werden, um die Kommunikation zwischen neuen und bereits etablierten Rollen zu fördern. Das RE bietet zahlreiche Techniken, die dabei helfen könnten,

größere Probleme zu unterteilen und die neuen Rollen auf den unterschiedlichen Abstraktionsebenen zu unterstützen (z. B. Kontextmodellierung, Zielmodellierung).

Beide Ansätze haben ihre Vor- und Nachteile, und jeder Benutzer/jedes Unternehmen muss, am besten basierend auf dem Business Case, seinen eigenen Weg finden.

4.3.4 Beispiel-Frameworks für das Skalieren von RE@Agile

Frameworks, die eine Skalierung von Scrum und agilen Entwicklungsprozessen unterstützen: Es gibt zahlreiche verschiedene Frameworks, die diese Ansätze unterstützen, und aufgrund der zunehmenden Bedeutung der Skalierung agiler Entwicklungsprozesse steigt auch deren Anzahl sehr schnell. Nachfolgend finden Sie eine Auswahl der bekanntesten Frameworks:

- **Scaled Agile Framework (SAFe)**
  SAFe ist eine Wissensdatenbank mit bewährten Erfolgsmustern für die Implementierung von Lean-Agile-Software- und -Systementwicklungen für Unternehmen. [SAFe]
- **Large-Scale Scrum (LeSS)**
  LeSS ist die Anwendung von Scrum auf zahlreiche Teams, die zusammen mit einem Product Owner an einem Produkt arbeiten. [Less]
- **Nexus**
  Nexus ist ein Framework, das auf das Zentrum der Skalierung abzielt: Cross-Team-Abhängigkeiten und Integrationsprobleme. [Nexus2015]
- **Scrum of Scrums**
  Scrum of Scrums ist eine Technik bei der Scrum zum Koordinieren von mehreren Teams verwendet wird. [SofS] Jedes Team benennt dabei eine Person (einen Botschafter), die es bei den Koordinierungssitzungen vertritt, die normalerweise zwei- oder dreimal pro Woche stattfinden [CPREALAGILE2022].
- **Scrum@Scale**
  Das Scrum-at-Scale-Framework ist eine minimale Erweiterung des zentralen Scrum-Frameworks, das die modulare Struktur im Kern des Scrum-Frameworks beibehält und zudem erlaubt, dass sich Scrum-Implementierungen maßgeschneidert auf die einzigartigen Bedürfnisse des Unternehmens skalieren lassen [SatS].

4.3.5 Auswirkungen der Skalierung auf RE@Agile

Die oben ausgeführten Abstraktionsschichten bedeuten, dass es mehr Rollen für das Management der RE-Aktivitäten gibt, so etwa Chief Product Owner, Produktmanager, Portfolio-Manager oder Business-Analysten. Jede Rolle erstellt für ihren jeweiligen Problembereich entsprechende RE-Arbeitsprodukte (Epics, Features, User Storys und die Verfolgbarkeit zwischen diesen). Es sind zusätzliche Meetings für RE-Aktivitäten erforderlich (z. B. Story-Times, Feature-Times, Systemdemos), und die Kommunikation mit Stakeholdern wird von den unterschiedlichen Rollen auf den verschiedenen Ebenen in der Organisation ausgeführt.

Da Scrum für sich allein nicht so leicht skalierbar ist, spielt das RE bei der Bestimmung dessen, wie umfassende Anforderungen zerlegt und angemessen unter den zahlreichen agilen Teams verteilt werden, eine entscheidende Rolle, um sich Skalierungsmethoden noch offenzuhalten. Die RE-Techniken helfen bei der Strukturierung von Problemanalysen und bei der Verfeinerung von groben in detailliertere Anforderungen, beispielsweise Features und User Storys für einzelne Teams. Ein strukturierter Ansatz, bei dem angemessene Abstraktionen und verschiedene Analyseperspektiven zum Einsatz kommen, bietet eine fundierte Grundlage für eine weitere Unterteilung der Aufgabe unter teilweise autonomen, agilen Teams.

Das gilt vor allem für Abhängigkeiten innerhalb der Anforderungen, die so früh wie möglich aufgedeckt und besprochen werden müssen, um Überflüssiges in der Entwicklung zu vermeiden.

Wenn Teams oder andere Rollen an unterschiedlichen Standorten arbeiten, kommt eine weitere Herausforderung hinzu. Die Komplexität beim Management der Kommunikation nimmt nicht nur aufgrund möglicher Zeitverschiebungen oder sprachlicher Unterschiede zu. Die größte Herausforderung liegt meist in den kulturellen Unterschieden. Da Kommunikation eine der Hauptaufgaben von Product Ownern und Produktmanagern ist, hat dies einen erheblichen Einfluss auf die erforderlichen Kompetenzen der RE-relevanten Rollen.

4.4 Ausgleichen von Vorab- und kontinuierlichen Aufgaben des Requirements Engineering im Zusammenhang mit Skalierung (K1)

Auf einer sehr abstrakten Ebene lassen sich agile Methoden als ein kontinuierlicher, iterativer Prozess charakterisieren, in dem ein System basierend auf den Backlog Items inkrementell entwickelt wird. Aus Requirements-Engineering-Perspektive lassen sich fünf Parameter (siehe folgende Unterkapitel) identifizieren, die diesen Prozess prägen:

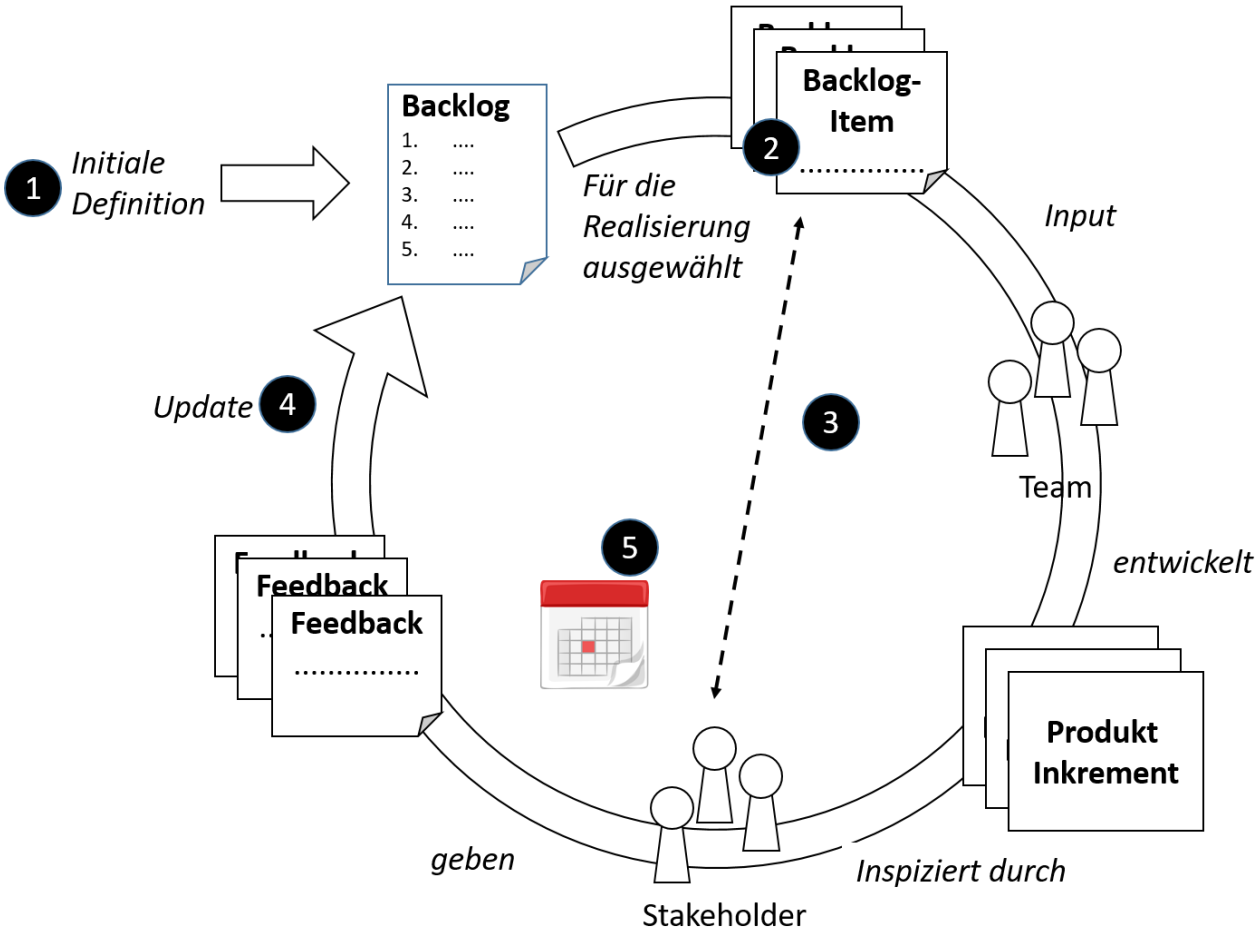

Abbildung 2: Kontinuierlicher Requirements-Engineering-Prozess

4.4.1 Definition erster Anforderungen

Bevor der kontinuierliche Entwicklungsprozess beginnen kann, muss ein erstes Backlog erstellt werden (siehe [TaNo1986]). Diese erste Definition des Backlogs wird häufig als Vorabdefinition bezeichnet. Das Requirements Engineering wird manchmal so verstanden, dass diese Vorabdefinition eine vollständige und detaillierte Spezifikation aller Anforderungen sein soll. Das mag unter Umständen für einige Vorgehensmodelle oder Domänen der Fall sein, es muss allerdings nicht immer so sein und ist in agilen Entwicklungen auch eher unüblich.

Die Basislinie für die anfängliche Definition des Backlogs ist die Summe der Informationen, die für den Beginn der ersten Iteration des kontinuierlichen Entwicklungsprozesses benötigt wird. Der Start des Iterationsprozesses ist eine äußert wichtig Entscheidung. Abhängig von System und Kontext können Unsicherheiten in Bezug auf die Anforderungen (wenn mit unzureichendem Wissen begonnen wird) zu zusätzlichen Verzögerungen, Kosten oder möglicherweise zum Scheitern des Projekts führen. Im Gegenzug kann sich ein zu später Start negativ auf die Markteinführungszeit auswirken und auf die Eignung für die Bedürfnisse der Endbenutzer zu einem bestimmten Zeitpunkt.

Das Requirements Engineering besagt, dass Anforderungen, von denen bekannt ist, dass sie große Auswirkungen auf die Architektur, auf die Realisierbarkeit der Lösung insgesamt oder auf zentrale Entscheidungen in Bezug auf Infrastruktur und Hardware haben, zu einem früheren Zeitpunkt ermittelt und detailliert erfasst werden sollten. Solche architekturrelevanten Probleme sollten tatsächlich vor der ersten Entwicklungsiteration ermittelt und analysiert werden. Anforderungen mit geringeren Auswirkungen können dann im Verlauf der Iterationen in einem kontinuierlichen Prozess verfeinert werden.

In den iterativen und inkrementellen Prozessen agiler Entwicklung wird das Requirements Engineering zu einem kontinuierlichen Prozess, der Anforderungen just in time liefert und verfeinert, und zwar genau in der für den Entwicklungszyklus erforderlichen Detailtiefe. Der Prozess erinnert an einen Trichter zum Zerkleinern von Steinen, bei dem zunächst große Steine in den Trichter gelangen, die dann auf mittlere Größe zerkleinert werden, bis sie am Ende des Prozesses schließlich zu kleinen Steinen geworden sind.

4.4.2 Detaillierungsgrad für Backlog Items

Der Detaillierungsgrad eines Backlog Items beschränkt die Freiheit der Entwickler bei der Realisierung eines Backlog Items (siehe [Pich2010]). Alle Aspekte eines Backlog Items, die nicht definiert worden sind, liegen in der Entscheidungsgewalt des Teams, was dem Team – innerhalb der definierten Geschäftsgrenzen – mehr Kreativität erlaubt.

Die bei den Entwicklern vorhandenen Kompetenzen dienen hier als Faustregel. Wenn die Entwickler über ausreichende Kompetenzen und Erfahrung für die Entscheidung über die Details eines Backlog Items verfügen (z. B. wenn ein Experte für Authentifizierungen und Berechtigungen unter den Entwicklern ist), dann sollte diese Entscheidung den Entwicklern überlassen werden. Siehe dazu auch 3.1.5.

4.4.3 Validität von Backlog Items

Kennt man die Validität eines Backlog Items vor Beginn der Implementierung, dann hilft dies, unnötige Implementierungsarbeit zu minimieren (siehe [Denn2015]). Darauf zu warten, dass man eine falsche oder unvollständige Anforderung in der implementierten Software feststellt, ist ein sehr teurer Ansatz der Anforderungsvalidierung.

Die Bestimmung der Validität eines Backlog Items ist eng mit seinem Detaillierungsgrad verbunden. Die Validität eines Backlog Items kann nur bestimmt werden, wenn es genügend Details enthält. Daher muss man den Aufwand für die Ausarbeitung und Validierung einer Anforderung vor der Implementierung dem voraussichtlichen Aufwand für die Implementierung der Anforderung und die anschließende Validierung in der ausgelieferten Software gegenüberstellen.

Wenn die Präferenz eines Stakeholders in Bezug auf ein Backlog Item mit tragbarem Aufwand ermittelt werden kann, sollte eine derartige Anforderungsvalidierung vor der Entwicklung des Backlog Items stattfinden.

Nachfolgend finden Sie einige typische Beispiele für solche Arten von Anforderungen (einschließlich eines beispielhaften Validierungsansatzes): das Design der

Benutzeroberfläche insgesamt (z. B. mit UI-Mock-ups), Berechtigungs- und Authentifizierungsmechanismen (z. B. mit Reviews von Anwendungsfällen), in dem System zu speichernde Datenstrukturen (z. B. mit Reviews von Datenmodellen) und Anforderungen für Schnittstellen zu vorhandenen Systemen (z. B. mit Reviews von Aktivitätsdiagrammen).

Backlog Items mit hohem Risiko (z. B. kritische Geschäftsfunktionen, sicherheitsrelevante Funktionen, innovative Funktionalitäten) oder solche mit hohen Testkosten für die Implementierung (z. B. wenn die Software in einem teuren Prototyp getestet werden muss) sollten vor der Implementierung validiert werden.

4.4.4 Feedback zum Backlog und dessen Aktualisierung

Backlogs werden häufig basierend auf Feedback aus einer Inspektionsaktivität, wie z. B. dem Sprint Review in Scrum, aktualisiert. Ein solcher Ansatz ist für Anforderungen mit geringer Skalierung oder detaillierte Anforderungen möglich, für die Auswirkungen einer Änderung schnell analysiert und erfasst werden können, und zwar in einer Umgebung mit ein oder zwei kleinen Teams. Es ist nicht ratsam, Anforderungen, die komplexer sind oder die vielfältige Abhängigkeiten aufweisen, kurzfristig zu ändern. In diesen Fällen sind Änderungen des Backlogs zeitaufwendiger, Stakeholder sind unter Umständen nicht greifbar und es können zusätzliche Analysen nötig sein.

Ein weiterer Faktor, der sich auf die Änderung von Backlog Items auswirken kann, ist der Entscheidungsfindungsprozess der Organisation. In Organisationen, wo wesentliche Entscheidungen einige Zeit beanspruchen können (z. B. wenn das verantwortliche Gremium nur alle drei Monate zusammenkommt), muss das Prinzip kontinuierlicher Verfeinerung konkret in Meetings umgesetzt werden, an denen alle betroffenen Stakeholder teilnehmen. Das RE dient zur Vorbereitung und als Entscheidungshilfe für solche Meetings.

4.4.5 Zeitplan für den Entwicklungszyklus

Der endgültige Parameter für den Entwicklungsprozess ist der Zeitplan oder die Dauer der Iteration. Diese wirkt sich erheblich auf die Requirements-Engineering-Aktivitäten aus, die während der Arbeit an Backlog Items ausgeführt werden müssen, die aktuell nicht entwickelt werden. Darüber hinaus bestimmt die Iterationsdauer die Häufigkeit, mit der Ergebnisse an Stakeholder ausgeliefert werden oder für Kunden zur Durchsicht verfügbar sind.

Folglich steigern kürzere Iterationszeiten das Arbeitspensum für die Stakeholder und zwar aus drei Gründen (vgl. z. B. [Rein1997]):

1. Stakeholder müssen während der gesamten Iteration verfügbar sein, um am Backlog mitzuarbeiten und Informationen für die nächste Iteration zu liefern.
2. Stakeholder müssen die vom Team erarbeiteten Ergebnisse durchsehen und Feedback dazu abgeben.
3. Stakeholder müssen immer wieder den Arbeitskontext wechseln, d. h. zwischen ihrem Tagesgeschäft und der Projektarbeit.

Längere Iterationszeiten verringern den Druck, reduzieren aber auch die Möglichkeiten für die Produktentwicklung, das Backlog zu beeinflussen.

Die Iterationsdauer muss unter Berücksichtigung der Verfügbarkeit der Stakeholder festgelegt werden. Stakeholder sind in der Regel nicht zu 100 % für Entwicklungsaktivitäten verfügbar, da sie in der Organisation, für die das System entwickelt wird, andere Verpflichtungen haben.

Als Faustregel gilt: Eine kürzere Iterationsdauer ermöglicht häufiges Feedback und mehr Möglichkeiten, Fehler frühzeitig zu erkennen; daher beschleunigen kürzere Iterationszeiten die Entwicklungsaktivitäten tendenziell. Wenn eine kurze Zykluszeit ein inakzeptables Pensum für die Stakeholder darstellt, sollte für die Iterationsdauer ein Kompromiss gefunden werden, auch wenn diese im Verhältnis zur Projektpriorität kürzer ausfallen kann.

Ein weiterer Faktor, der sich auf die Zykluszeit auswirken kann, ist der durchschnittliche Umfang und die Komplexität von Backlog Items. Größere oder komplexere Backlog Items erfordern mehr Zeit zum Verständnis und für die Analyse. Daher kann die Zykluszeit verlängert werden, um größere oder komplexe Backlog Items innerhalb einer einzigen Iteration abzuwickeln. Wenn das entwickelte System sich beispielsweise in einer frühen Phase befindet, kann unter Umständen ein längerer Zyklus ratsam sein, damit das Team mehr Zeit hat, ein erstes Verständnis von dem System zu gewinnen. Dieser Faktor und das Ziel kürzerer Zykluszeiten müssen dennoch ausgeglichen sein, um häufiger Feedback zu erhalten. Die Entscheidung für längere oder kürzere Zykluszeiten muss gemeinsam im Team getroffen werden, unter Abwägung der Zeit, die für die Analyse benötigt wird, mit dem Ziel, frühzeitig Feedback zu erhalten. Veränderte Zykluszeiten beruhen immer auf dem Prinzip „Inspect and Adapt", unter Berücksichtigung dessen, dass bisherige Erfahrungen nicht immer eine Garantie für die Zukunft bedeuten. Die Änderung der Zykluszeit sollte vor, aber niemals während einer Iteration stattfinden. Darüber hinaus sollte die Zykluszeit so einheitlich wie möglich sein, um die Komplexität zu reduzieren und eine erreichbare Prognosefähigkeit zu schaffen. Eine zu häufige Änderung der Dauer kann zu Instabilität führen, das Engagement des Teams beeinträchtigen und höchstwahrscheinlich ein praktikables Tempo der Produktentwicklung verhindern.

5 Begriffsdefinitionen, Glossar (K2)

Das CPRE Glossar und der SCRUM Guide definieren die für den RE@Agile Primer relevanten Begriffe.

- Das CPRE Glossar steht auf der IREB Homepage zum Download zur Verfügung: https://cpre.ireb.org/de/downloads-and-resources/downloads#cpre-glossary
- Der SCRUM Guide steht zum Download zur Verfügung unter: https://scrumguides.org/download.html

6 Literaturverzeichnis

[ACP/PMI] Agile Certified Practitioner: https://www.pmi.org/certifications/agile-acp. Zuletzt besucht im Juni 2026.

[AgileMan2001] Agile Manifesto: http://agilemanifesto.org/iso/de/manifesto.html, 2001. Zuletzt besucht im Juni 2026.

[AmLi2012] Ambler S.; Lines, M.: Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. IBM Press, 2012

[Ande2012] Anderson, D.J.: Lessons in Agile Management: On the Road to Kanban. Blue Hole Press, 2012

[Appel2011] Appelo J.: Management 3.0. Addison-Wesley Professional, 2011

[Beck2003] Beck, K.: Test-Driven Development by Example. Addison Wesley – Vaseem, 2003

[Beck2004] Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, 2004

[CNYM2000] Chung, L.; Nixon, B. A.; Yu, E.; Mylopoulos, J. : Non-Functional Requirements in Software Engineering. Springer Science & Business Media, 2000

[Cock1998] Cockburn, A.: Surviving Object-Oriented Projects. Addison-Wesley, 1998

[Cohn2004] Cohn, M.: User Stories Applied For Agile Software Development. Addison Wesley Professional, 2004

[Conw1968] Conway, M.: How Do Committees Invent? Datamation 14(4):28-31, 1968. Article available at http://www.melconway.com/Home/Conways_Law.html. Zuletzt besucht im Juni 2026.

[CPREFL2022] IREB e.V.: Syllabus CPRE Foundation Level, Version 3.3.0. https://cpre.ireb.org/de/downloads-and-resources/downloads#cpre-foundation-level-syllabus. Zuletzt besucht im Juni 2026.

[CPREALAGILE2022] IREB e.V.: Syllabus CPRE Practitioner | Specialist, Version 2.3.0. https://cpre.ireb.org/de/downloads-and-resources/downloads#cpre-re-agile-syllabus. Zuletzt besucht im Juni 2026.

[Denn2015] Denning, S.: How To Make The Whole Organization Agile. http://www.forbes.com/sites/stevedenning/2015/07/22/how-to-make-the-whole-organization-agile/#658d3f65135b, 2015. Zuletzt besucht im Juni 2026.

[Dsch2015] d.school: An Introduction to Design Thinking – Process Guide. https://web.stanford.edu/~mshanks/MichaelShanks/files/509554.pdf, 2015. Zuletzt besucht im Juni 2026.

[Glinz2026] Glinz, M.: A Glossary of Requirements Engineering Terminology, https://cpre.ireb.org/de/downloads-and-resources/downloads#cpre-glossary. Zuletzt besucht im Juni 2026.

[Griff2015] Griffiths, M.: PMI-ACP Exam Prep. Rmc Publications, 2015

[GoAk2003] Gordijn, J.; Akkermans, J.M.: Value-based Requirements Engineering: exploring innovative e-commerce ideas. Springer, 2003

[High2009] Highsmith, J.: Agile Project Management: Creating Innovative Products. Addison-Wesley Professional, 2009

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

[KnZK2016] Knapp, J.; Zeratsky, J; Kowitz, B.: Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days. Simon & Schuster, 2016

[Kron2008] Kronfaelt, R.: Ready-ready: the Definition of Ready for User Stories going into Sprint planning. https://scrumftw.blogspot.com/2008/10/ready-ready-definition-of-ready-for.html, 2008. Zuletzt besucht im Juni 2026.

[Less] Large Scale Scrum: https://less.works/resources/LeSS-brochure.pdfLast visited June 2026.

[LiOg2011] Liedtka, J.; Ogilvie, T.: Designing for Growth: A Design Thinking Tool Kit For Managers. Columbia Business School Publishing, 2011

[Martin1991] Martin, J.: Rapid Application Development. Macmillan Coll Div, 1991

[Meyer2014] Meyer, B.: Agile! – the good, the hype, and the ugly. Springer, 2014

[MeMi2015] Mesaglio, M., Mingay, S.: Bimodal IT: How to Be Digitally Agile Without Making a Mess, Gartner 2015, https://www.gartner.com/en/documents/2798217. Zuletzt besucht im Juni 2026.

[Nexus2015] Nexus Guide https://www.scrum.org/resources/nexus-guide,. Zuletzt besucht im Juni 2026.

[Patt2014] Patton, J.: User Story Mapping. O'Reilly, 2014

[Pich2010] Pichler, R: Make the product backlog deep. http://www.romanpichler.com/blog/make-the-product-backlog-deep/,. Zuletzt besucht im Juni 2026.

[PMI] PMI Project Management Institute. http://www.pmi.org/. Zuletzt besucht im Juni 2026.

[Popp2003] Poppendieck, M.: Lean Software Development: An Agile Toolkit. Addison-Wesley Professional, 2003

[Rein1997] Reinerstsen, D. G.: Managing the Design Factory – A Product Developer's Toolkit. Simon & Schuster, 1997

[Ries2011] Ries, E.: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Publishing Group, 2011

[SAFe] SAFe – Scaled Agile Framework. http://www.scaledAgileframework.com. Zuletzt besucht im Juni 2026.

[SatS] Scrum@Scale framework: https://www.scrumatscale.com/scrum-at-scale-guide/. Zuletzt besucht im Juni 2026.

[Scrum2020] Schwaber, K. & Sutherland, J.: The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game, July 2020. https://scrumguides.org/download.html. Zuletzt besucht im Juni 2026.

[ShYo2006] Sheppard, J. M.; Young W. B.: Agility literature review: Classifications, training and testing. Journal of Sports Sciences 24(9): 919-932, 2006

[SofS] Scrum of Scrums https://scrumguide.de/scrum-of-scrums. Zuletzt besucht im Juni 2026.

[TaNo1986] Takeuchi, H.; Nonaka, I.: The new new product development game. Harvard Business Review 64(1), January/February 1986, p.137-146

[Wake2003] Wake, B.: Invest in Good Stories and Smart Tasks, http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Zuletzt besucht im Juni 2026.