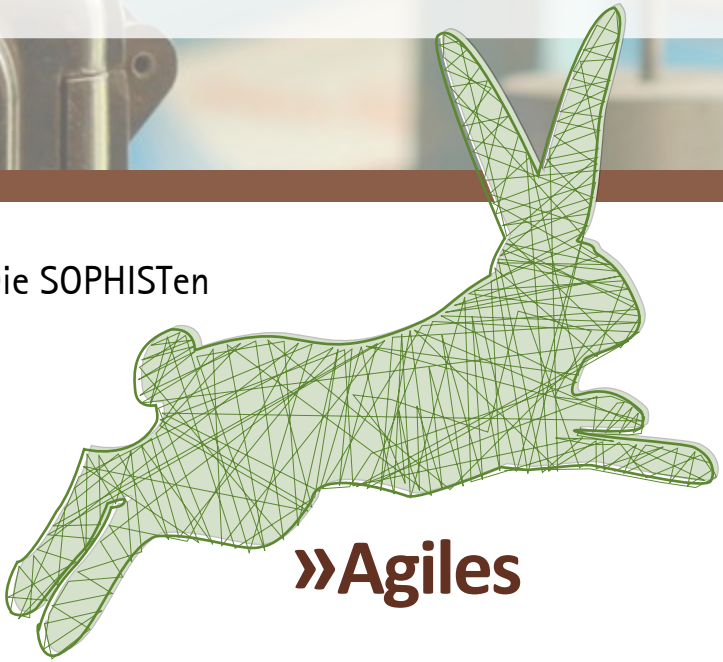
AGILITÄT

Die SOPHISTen



»Agiles Requirements-Engineering«

SOPHIST

Die SOPHISTen

SOPHIST GmbH
Vordere Cramergasse 13
90478 Nürnberg
Deutschland
www.sophist.de

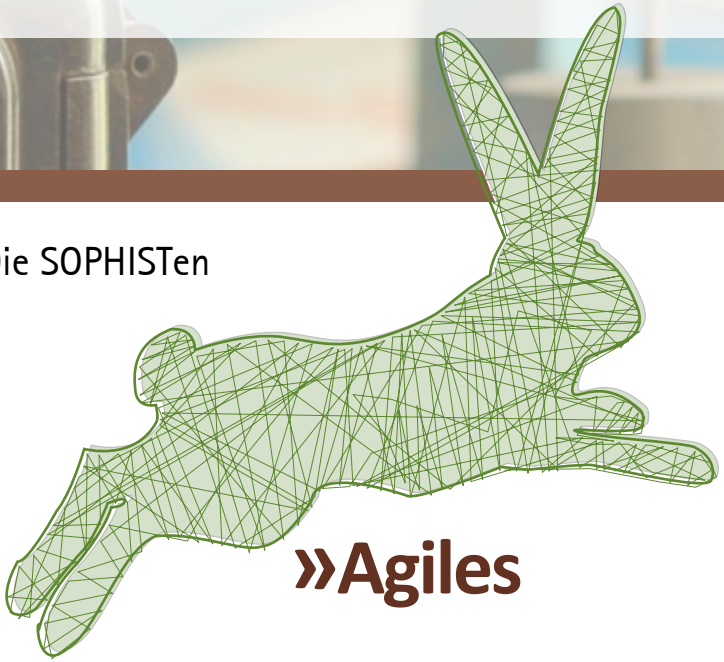
@ SOPHIST_GmbH
@SOPHIST.GmbH
/sophistgmbh
blog.sophist.de

1. Auflage 2021

Copy Editing & Herstellung: Roland Kluge, SOPHIST GmbH
Illustrationen: Assad Binakhahi

AGILITÄT

Die SOPHISTen



»Agiles Requirements-Engineering«

SOPHIST

www.sophist.de

Liebe Leserin, lieber Leser

Beim Verfassen dieser Broschüre haben wir stets auf eine gender-gerechte Formulierung von Begriffen geachtet. Dennoch kommen vereinzelt Bezeichnungen vor, in denen wir die maskuline Form zugunsten der Lesefreundlichkeit und der Verständlichkeit belassen haben (wie z. B.: der Requirements-Engineer, der Product Owner oder der Scrum Master). In diesen Fällen betrachten wir diese als Rollenbezeichnung, nicht als konkrete Person. Wir bitten hierfür um Ihr Verständnis.

1. Einleitung

1.1 Was ist RE?

Sie haben einen Wunsch, den Sie nicht erfüllen können? Sie haben eine Produktvision oder ein Ziel, wissen jedoch nicht wie Sie dieses erreichen sollen? Angenommen Sie wollen ein blaues Auto mit möglichst vielen PS und wenden sich an ein Entwicklungsteam, um dieses Auto zu entwickeln. Als Ihnen dann das Endprodukt vorgestellt wird, folgt jedoch die Ernüchterung: Ihr Entwicklungsteam liefert ein blaues Automobil mit vielen PS, jedoch ist dies ein LKW und kein Sportwagen, wie Sie es sich vorgestellt hatten. Da es bei der Übermittlung, Analyse und Vermittlung von Anforderungen Schwierigkeiten bei der Kommunikation gibt, kommt das Requirements-Engineering ins Spiel. Früher dachte man, dass Requirements-Engineering simple oder überflüssig sei: Der Requirements-Engineer führt Interviews mit Stakeholdern, schreibt daraus resultierende Anforderungen auf und verfasst diese zu einem Lastenheft.

Dies ist jedoch nur teilweise richtig. Der Requirements-Engineer ist für mehr zuständig, als nur das. Er/sie dient als Bindeglied zwischen den Stakeholdern und den Entwicklern. Die folgenden Aufgaben gehören zu seinen/ihren vier Haupttätigkeiten:

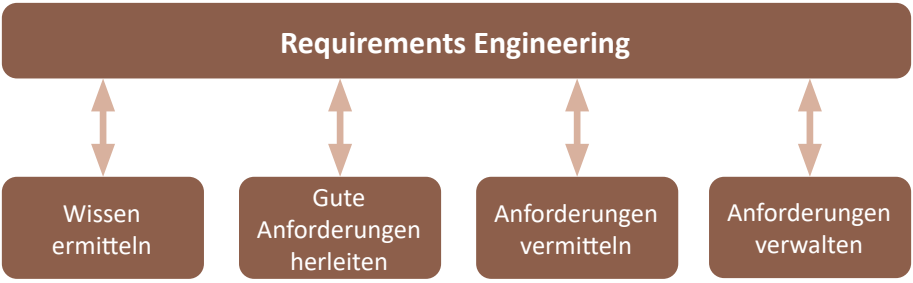


Abbildung 1.1: Die vier Haupttätigkeiten des Requirements-Engineering

Das **Ermitteln des Wissens** der Stakeholder ist seine/ihre wichtigste Tätigkeit. Es ist jedoch nicht so simple wie es klingt, da Wissen in drei Kategorien eingeteilt wird: Das bewusste, das unterbewusste und das unbewusste Wissen. Im oben genannten Beispiel waren Sie sich nur bewusst, dass das Auto viele PS haben und blau sein soll. Diese Eigenschaften werden auch Leistungsfaktoren genannt. Unterbewusst wollten Sie jedoch ein Sportwagen entwickeln, was für Sie einen Basisfaktor darstellt. Vielleicht begeistern Sie sich auch unterbewusst für automatisiertes Fahren in Ihrem neuen Auto. Dies wird als Begeisterungsfaktor bezeichnet. Damit der Requirements-Engineer diese Faktoren ermitteln kann, benötigt er/sie hierfür verschiedene Ermittlungstechniken wie beispielsweise das Führen von Interviews, die Wiederverwendung von vergangenen Produkten oder das Brainstorming.

Damit der Requirements-Engineer sein/ihr ermitteltes Wissen an die Entwicklerteams liefern kann, muss er/sie **gute Anforderungen herleiten**, welche die Entwicklungsteams verstehen und umsetzen können. Wichtig hierbei ist, dass Anforderungen eindeutig, vollständig, notwendig, unter den Stakeholdern abgestimmt und verständlich sind, denn ansonsten könnte das Endprodukt nicht Ihren Wünschen entsprechen. Hierfür gibt es gewisse Qualitätskriterien wie z.B. das INVEST-Prinzip. Wir bei SOPHIST verwenden hierfür z.B. unser *RE*gelwerk oder auch unsere MASTER-Schablonen, um Anforderungen in guter Qualität zu entwickeln.

Die Aufbereitung der Anforderungen ist wichtig, da der Requirements-Engineer die **Anforderungen** an weitere Personen **vermitteln** muss. Hierfür verwendet er/sie verschiedene Vermittlungstechniken, die er/sie situativ einsetzen muss. Beispiele hierfür können Storytelling oder Videos sein. Typischerweise spielt bei der Vermittlung von Anforderungen nicht nur der Inhalt eine Rolle, sondern auch z.B. die Komplexität oder der Umfang des Betrachtungsgegenstands. In diese Haupttätigkeit fällt ebenfalls die Dokumentation der Anforderungen, klassischerweise z.B. über Anforderungsspezifikationen, oder in der agilen Welt mittels des Product Backlogs statt.

Zuletzt muss der/die Requirements-Engineer noch die **Anforderungen verwalten**. Insbesondere diese Tätigkeit dient der Nachvollziehbarkeit und der Änderbarkeit von Anforderungen. Vielleicht müssen Anforderungen angepasst werden, weil Sie doch ein rotes Auto wollen, oder weil Sie festgestellt haben, dass Sie doch nicht den Motor mit den meisten PS einbauen wollen. Durch Versionisierung und Traces kann hierbei ein Überblick über die Änderungen und Abhängigkeiten der Anforderungen behalten werden. Vor allem dient diese Haupttätigkeit der Kommunikationsverbesserung innerhalb und zwischen Entwicklungsteams im Projekt, einer erhöhten Qualität von Anforderungen und somit einer erhöhten Qualität von Produkt und Prozessen, sowie einer vereinfachten Überwachung komplexer Vorhaben während aller Entwicklungsschritte. Und das ist für alle Beteiligten nervenschonender.

Die vier Haupttätigkeiten anhand eines Beispiels finden Sie im folgenden Video:

www.sophist.de/bagd/k1v1

1.2 Was ist Agilität

Was unterscheidet Agilität genau von schwergewichtigeren Methoden, wie dem Wasserfallmodell? Die vier Grundideen von Agilität werden im "Manifesto for agile Software Development", auch das agile Manifest genannt, festgehalten:

Abbildung 1.2: Die vier Grundideen des agilen Manifests

Hierbei ist die Zusammenarbeit mit dem Kunden oder der Kundin wichtiger als die Vertragsverhandlungen. Natürlich sind Vertragsverhandlungen auch wichtig, denn die Zusammenarbeit sollte auch geregelt sein (benötigte Ressourcen, Erwartungen, Ziele, etc.). Aber die gute und regelmäßige Zusammenarbeit bringt mehr Vorteile als ein noch so detaillierter Vertrag.

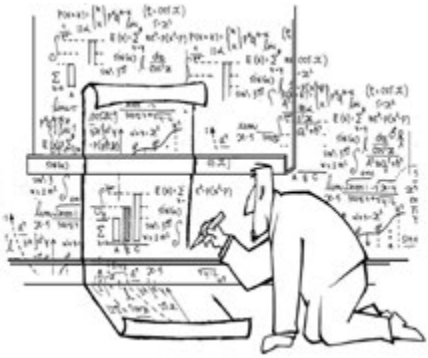
Außerdem ist eine funktionierende Software wichtiger als eine umfassende Dokumentation. Auch dies bedeutet nicht, dass die Dokumentation außer Acht gelassen wird, sondern viel eher, dass die Zeit in eine funktionierende Software investiert werden sollte, statt in eine ausführliche Dokumentation. Ihnen als KundIn bringt es nichts, wenn Ihre Software nicht funktioniert, diese jedoch ausführlich dokumentiert wurde.

Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge. Auch hier sollten die Prozesse und Werkzeuge nicht ganz außer Acht gelassen werden, jedoch wollen wir die Kommunikation zwischen und in den verschiedenen Teams gewähr-

leisten. Selbst wenn Ihrem Entwicklungsteam die besten Werkzeuge zur Verfügung gestellt werden, wird ein gutes Endprodukt nur mit guter Kommunikation zwischen Ihren Teammitgliedern und den zuständigen Personen, wie dem Product Owner, erreicht.

Reagieren auf Veränderungen ist wichtiger als das Befolgen eines Plans. Einen Plan zu haben, ist nicht schlecht. Jedoch ist das Ziel der Agilität spontan auf z.B. Marktveränderungen, die Sie interessieren, zu reagieren und das Produkt nach Ihren Wünschen zu formen. Denn nur so ist Ihr Produkt auf dem neuesten und aktuellen Stand, den Sie sich wünschen.

Die Gründe, warum Produkte agil entwickelt werden, sind vielfältig. Die wichtigsten wollen wir zumindest mal ansprechen:



- Durch die iterative Vorgehensweise in Sprints kann frühzeitig und regelmäßig Feedback durch die Stakeholder eingeholt werden. Dadurch werden Entwicklungen in die falsche Richtung vorgebeugt.
- Die iterativ-inkrementelle Arbeitsweise macht komplexe System beherrschbar.
- Die Zusammenarbeit in einem Team fördert die Motivation der MitarbeiterInnen und sorgt dafür, dass sich alle mit dem Produkt identifizieren können.
- Das bewusste reduzieren von Regeln und Vorgaben sorgt für eine effiziente Arbeitsweise.

Zu den bekanntesten Methoden im agilen Kontext zählen:

- Scrum
- Kanban
- Crystal-Familie

Da Agilität häufig in Gruppen mit vielen Entwicklungsteams stattfindet, muss dort in einem Framework gearbeitet werden, welches die Kommunikation und Synchronisation von Abhängigkeiten zwischen den Teams erleichtert. Die wichtigsten und bekanntesten Frameworks der agilen Welt sind:

- SAFe
- LeSS
- Nexus
- Scrum@Scale

1.3 RE in Agilität

Da Agilität ein breitgefächertes Gebiet mit verschiedenen Vorgehensweisen und Frameworks ist, betrachten wir beispielhaft Requirements-Engineering in Scrum.

Um nun das Requirements-Engineering hierbei einschätzen zu können, folgt zuerst der grobe Aufbau von Scrum.

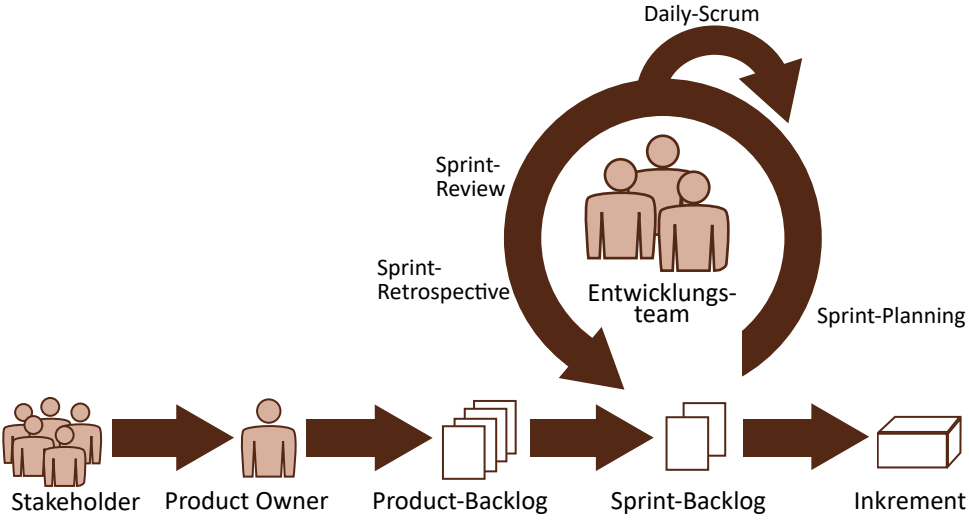


Abbildung 1.3: Der Ablauf einer Entwicklung nach Scrum

Im Scrum teilen die Stakeholder eine Produktidee oder Vision dem Product Owner mit. Der Product Owner notiert die dazugehörigen Anforderungen im Product Backlog und spricht diese mit dem Entwicklungsteam ab. Zum Start eines Sprints (ein Zeitraum zwischen 2-4 Wochen) Nehmen sich die Entwickler aus dem Product Backlog so viele Inhalte heraus, wie sie im Sprint umsetzen können. Sie definieren also das Sprint Backlog. Im Sprint werden die ausgesuchten Inhalte entsprechend der Absprachen zwischen Product Owner und Entwicklern umgesetzt. Hierbei unterstützt der Scrum Master das Entwicklungsteam bei Belangen bezüglich agiler Arbeitsmethoden und Prozesse. Das Ziel des Entwicklungsteam ist die Erstellung eines Produktinkrements in jedem Sprint. Ein Produktinkrement ist ein Teil des zukünftigen Endprodukts, welches potentiell auslieferbar wäre. Das bedeutet es handelt sich dabei um etwas funktionierendes in der geforderten Qualität. Somit wächst von Sprint zu Sprint das Produkt immer weiter.

Rollen und Aufgaben

- Damit ein Projekt erfolgreich ist, müssen die verschiedenen Rollen der beteiligten Personen vorab geklärt werden. Zum direkten Scrum-Team gehören die Rollen des Product Owners, die des Entwicklungsteams und die des Scrum Masters.
- Der Product Owner ist der Requirements-Engineer in der agilen Welt, der jedoch mehr Verantwortlichkeiten als ein klassischer Requirements-Engineer hat. Er/sie ist die Schnittstelle zwischen den Stakeholdern und dem Entwicklungsteam. Mit den Stakeholdern ermittelt er/sie die Vision oder das Produktziel, stimmt dies mit ihnen ab und hält dies mittels Product Backlog Items fest. Außerdem managt er/sie das Product Backlog, indem er/sie Product Backlog Items klar und verständlich formuliert, priorisiert und sortiert. Hierbei soll er/sie auch Rücksprache mit dem Entwicklungsteam halten und kann an das Entwicklungsteam Teile seiner/ihrer Aufgaben abgeben. Er/sie trägt jedoch die Verantwortung für die Wertmaximierung des Produkts, gemessen an der Produktvision oder der Produktziele. Aus diesem Grund gibt er die Reihenfolge in der Backlog Items umgesetzt werden vor, indem er die Backlog Items sortiert.
- Das Entwicklungsteam besteht aus 3 bis 9 Mitgliedern, die für die Realisierung des Produktinkrements verantwortlich sind. Im Endeffekt besteht es aus mehreren ExpertInnen in unterschiedlichen, für die Umsetzung relevanten Fachbereichen, die gleichgestellt an der Umsetzung des Sprint Backlogs arbeiten und dieses in das Endprodukt einfügen. Das Entwicklungsteam ist selbstorganisiert und trägt die Verantwortung für das umgesetzte Produkt. Häufig lassen sich auch Requirements-Engineers in Entwicklungsteams einsetzen, die den Product Owner bei seinen/ihren Tätigkeiten unterstützen.

- Der Scrum Master ist für die Durchführung und das Verständnis von Scrum zuständig. Er/sie ist die verantwortliche Person für die Umsetzung von Scrum in der Organisation. Das heißt, er/sie ist dafür zuständig, dass sich die Mitglieder des Entwicklungsteams und der Product Owner an die Regeln halten, auf die sie sich geeinigt haben und alle Beteiligten ein Verständnis über agile Arbeitsweise erlangen. Er/sie berät außerdem das Entwicklungsteam und den Product Owner bezüglich der Methoden und Techniken, welche in der täglichen Arbeit eingesetzt werden.
- Trotz alledem ist die wichtigste Rolle die der Stakeholder. Diese wird häufig von Personen, Unternehmen oder Organisationen eingenommen. Die Stakeholder bestimmen eine Vision oder ein Produktziel und sind diesbezüglich der Ansprechpartner für den Product Owner.

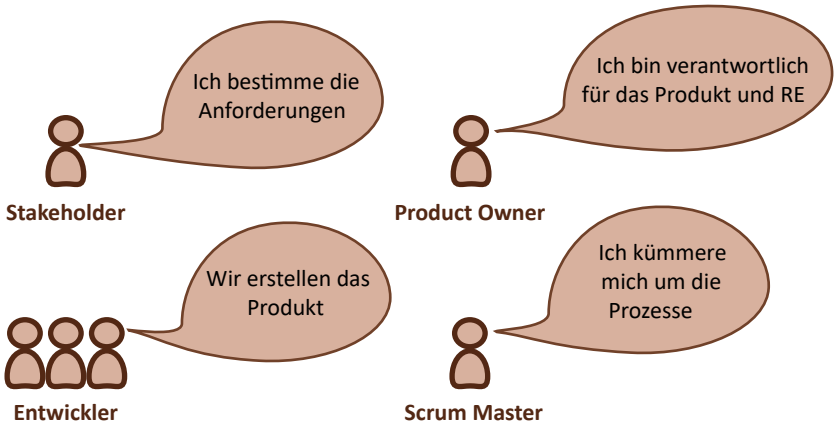


Abbildung 1.4: Die Rollen im Scrum-Team

Events

In Scrum sind ein paar Events festgelegt. Das sind im Einzelnen:

- Der Sprint ist ein Zeitraum von nur wenigen Wochen und am Ende eines jeden Sprints soll ein Teil des Produkts (ein Produktinkrement) umgesetzt sein. So wächst das Produkt von Sprint zu Sprint immer weiter.
- Das Sprintplaning findet zu Beginn des Sprints statt und dient dem Scrumteam festzulegen, was in dem Sprint umgesetzt werden soll. Zusätzlich werden die Entwickler auch planen, wie die Umsetzung im Sprint stattfinden soll.
- Das Review findet am Ende des Sprints statt und dient vor allem dazu, mit den Stakeholdern gemeinsam die Sprintergebnisse anzusehen und dabei auch gleich ein Feedback der Stakeholder einzusammeln.
- Das Daily findet täglich im Sprint statt und dient dazu, dass die Entwickler sich kurz abstimmen, welche arbeiten an diesem Tag anstehen.
- Die Retrospektive schließt einen Sprint ab und lässt die Teammitglieder den Sprint Revue passieren, um Ansätze zu finden, an denen die Arbeit des Scrumteams verbessert werden kann.

Wie das Requirements Engineering in Scrum aussieht, was in den Events passiert und was außerhalb der Events noch alles getan wird, möchten wir im Verlauf dieser Broschüre zeigen.

Artefakte

Scrum definiert drei Artefakte

- Das Product Backlog ist der Ort, an dem wir unsere Anforderungen ablegen werden. Das Backlog ist eine sortierte Liste mit all den Dingen, die in das Produkt eingearbeitet werden sollen (vgl. Kapitel 5.1)

- Das Sprint Backlog beinhaltet alle Backlog Items, welche in dem aktuellen Sprint umgesetzt werden sollen. Das Sprint Backlog wird zu Beginn des Sprints im Sprint Planning von den Entwicklern aus dem Product Backlog befüllt
- Das (Produkt) Inkrement ist das Ergebnis am Ende eines Sprints. Das Inkrement ist ein (kleines) Teil des Produkts, welches in einem Sprint entwickelt wird, so dass über mehrere Sprints hinweg das fertige Produkt entsteht

Es kann sinnvoll sein, weitere Artefakte zu verwenden. Diese drei genannten sind zumindest das Minimum an Artefakten, welche benötigt werden.

Im Kapitel 7 haben wir uns mit der Idee weiterer Artefakte für das Requirements-Engineering in Scrum befasst.

2. Ziele/Vision, Stakeholder, Systemabgrenzung

Bevor Sie als Requirements-Engineer Anforderungen an das zu entwickelnde System erheben können, ist es wichtig, dass Sie hierfür eine Grundlage schaffen. Diese Grundlage besteht darin, dass Sie die Ziele/Vision definieren, die richtigen Stakeholder finden und Kontext sowie Grenzen des zu entwickelnden Systems bestimmen.

2.1 Ziele/Vision

Ziele sind erste grobe Anforderungen, welche die Richtung vorgeben, in die entwickelt wird. Oft wird im Zusammenhang mit Zielen auch der Begriff Vision verwendet. Hierbei handelt es sich um eine grobe und langfristige Zieldefinition. Es ist wichtig, dass die Ziele/Vision dokumentiert werden und alle Beteiligten ein gemeinsames Verständnis des Ziels/Vision haben.

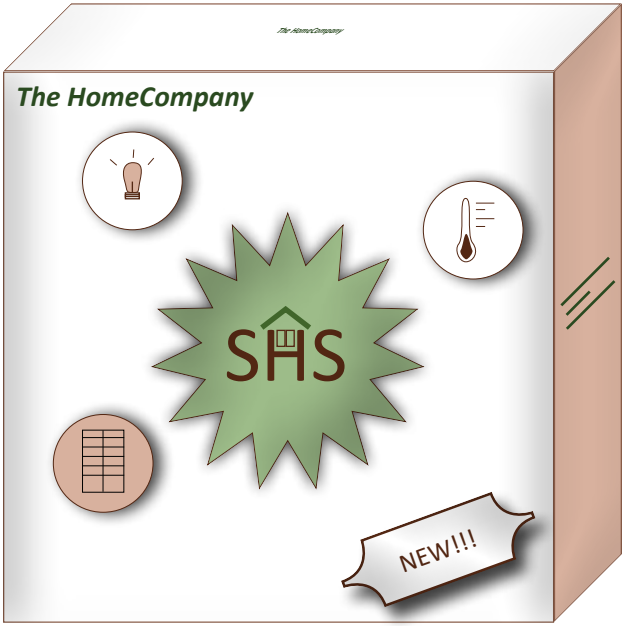


Abbildung 2.1: Beispiel für eine Vision-Box

Denn ohne klar definierte Ziele/Vision, kann es passieren, dass Sie als Requirements-Engineer keine Vorgaben haben, welche Ziele/Vision durch die Anforderungen erfüllt werden müssen und spezifizieren möglicherweise an den eigentlichen Zielen/Vision vorbei.

Und ohne ein gemeinsames Verständnis der Ziele/Vision, ist es möglich, dass einzelne Stakeholder am Ende unzufrieden sind, da das entwickelte System nicht ihren Vorstellungen entspricht.

Eine Unzufriedenheit können Sie vermeiden, indem Sie gemeinsam mit allen Stakeholdern die Ziele/Vision ermitteln und dokumentieren.

Zur Ermittlung können Sie einen Workshop veranstalten, bei dem Sie gemeinsam eine Vision Box erstellen, die Ziele/Vision nach PAM (Purpose, Advantage, Measure), News from the future oder mithilfe eines Canvas [Rupp 21] formulieren.

Um qualitativ hochwertige Ziele/Vision zu beschreiben, wird auch oft das Akronym SMART verwendet. Die einzelnen Buchstaben stehen dabei für Spezifisch, Messbar, Erreichbar, Relevant und Terminiert und können Ihnen dabei helfen, gute Ziele zu formulieren.

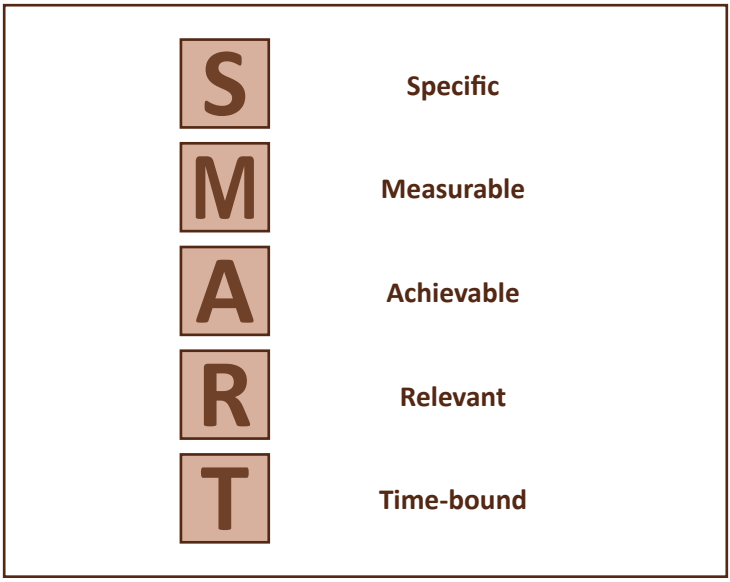


Abbildung 2.2: SMART

2.2 Stakeholder

Neben der Zieldefinition ist es auch wichtig, dass Sie alle relevanten Stakeholder finden. Wie wir bereits (in Kapitel 1.3) erwähnt haben, sind Stakeholder sehr wichtig.

Um alle relevanten Stakeholder zu finden, nutzen wir gerne Stakeholder-Listen. Sobald Sie einen Stakeholder ermittelt haben, sollten Sie die relevanten Informationen zu diesem Stakeholder systematisch dokumentieren. Eine einfache Art ist es, eine Tabelle zu führen.

Bedenken Sie, dass kontinuierlich Stakeholder hinzukommen können und die Stakeholderliste entsprechend gepflegt werden muss.

Im Gegensatz zu klassischen Vorgehensweisen müssen Sie als Requirements-Engineer bei agilen Vorgehensweisen die Stakeholder anders einbinden, da diese fortlaufend mitwirken sollen. Dadurch ergeben sich für den Requirements-Engineer weitere Aufgaben. Beispielsweise muss der Requirements-Engineer die NutzerInnen mit den Produktinkrementen konfrontieren, um Feedback einzuholen oder er/sie muss die Stakeholder dazu bewegen, sich fortlaufend zu beteiligen.

2.3 Systemabgrenzung

Die Grundlage für die Erhebung von Anforderungen ist nun fast vollständig. Nachdem Sie die Ziele/Vision sowie die relevanten Stakeholder identifiziert und dokumentiert haben, fehlt nur noch die Kontextabgrenzung.

Hierbei bestimmen Sie den groben Systemumfang, den Systemkontext und die damit verbundenen Systemgrenzen, um das System von dessen Umgebung abzugrenzen.

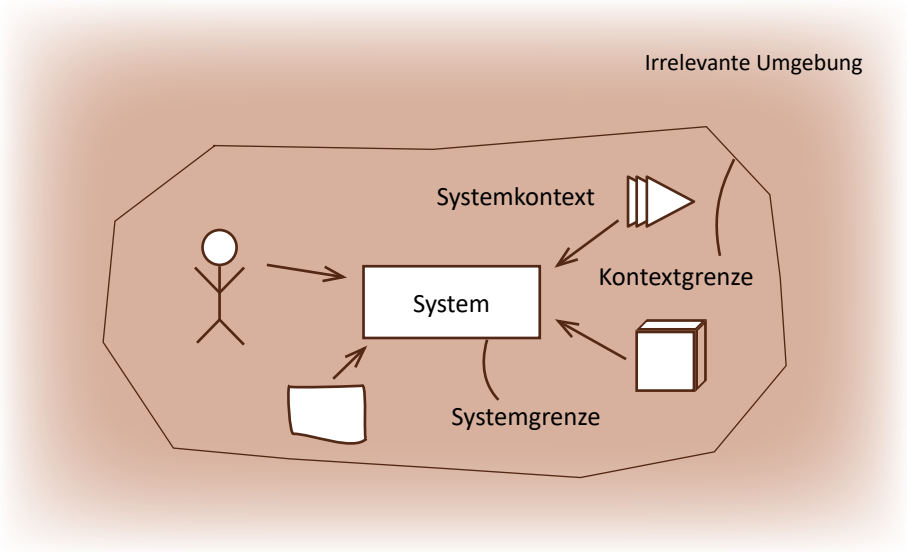
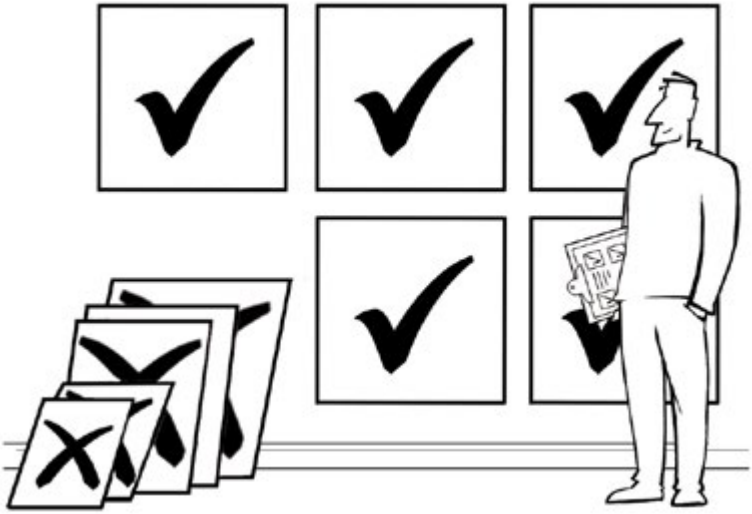


Abbildung 2.3: Kontext- und Systemabgrenzung

Die Systemgrenze wird dadurch bestimmt, dass für die Teile, die sich außerhalb der Systemgrenze befinden, später keine Anforderungen erstellt werden. Bei den Teilen außerhalb der Systemgrenze müssen Sie noch den Systemkontext von dem Bereich abgrenzen, der keine Rolle für die Anforderungen spielt. Die Grenzen zwischen diesen beiden Bereichen, wird Kontextgrenze genannt.

Zur Bestimmung des Systemkontexts müssen Sie alle Objekte identifizieren, die einen Bezug zu ihrem geplanten System und daher Einfluss auf die Anforderungen an das System haben.



Wie bei der Stakeholderliste müssen Sie damit rechnen, dass die Systemabgrenzung nicht final ist. Sie und die EntwicklerInnen entwickeln also das System, obwohl noch nicht die endgültige System- und Kontextgrenze feststeht.

Gerade in der Agilität haben wir das Phänomen, dass die Systemgrenze und somit die Kontextgrenze sich im Laufe der Entwicklung immer wieder ändert. Daher ist es von großer Wichtigkeit, diese immer im Auge zu behalten.

3. Von den Anforderungen zum Produkt

3.1 Das Leben eines Backlog Items

Wie auch in nicht-agilen Arbeitsweisen, müssen wir Anforderungen ermitteln, analysieren und vermitteln. Natürlich gibt es auch Verwaltungsarbeiten, aber diese liegen an dieser Stelle nicht im Fokus.

Betrachten wir zunächst einmal die Ermittlung der Anforderungen, die Tätigkeit, welche logischerweise als Startpunkt gelten kann. Der große Unterschied zu den nicht-agilen Arbeitsweisen ist der Zeitpunkt, zu dem wir Anforderungen ermitteln. Gerade detaillierte Anforderungen werden wir erst kurz vor deren Umsetzung betrachten und alles, was in ferner Zukunft umgesetzt wird, wird vielleicht nur als grobe Anforderung festgehalten. Daraus ergibt sich, dass wenn wir mit den Stakeholdern detaillierte Anforderungen erarbeiten, bereits Anforderungen umgesetzt wurden (außer in den ersten Iterationen) und somit die bereits bestehenden Systemteile als Hilfe für die Ermittlung der Anforderungen herangezogen werden können, bzw. herangezogen werden sollten.

Die ermittelten Anforderungen werden im Product Backlog als Backlog Items (z.B. User-Storys) gesammelt. Es kann hilfreich sein, die Backlog Items nach Themen zu gruppieren. Dies geschieht üblicherweise durch die Zuordnung detaillierter Backlog Items zu groben Backlog Items. Ein Beispiel dafür ist die Zuordnung von User-Storys zu Epics.

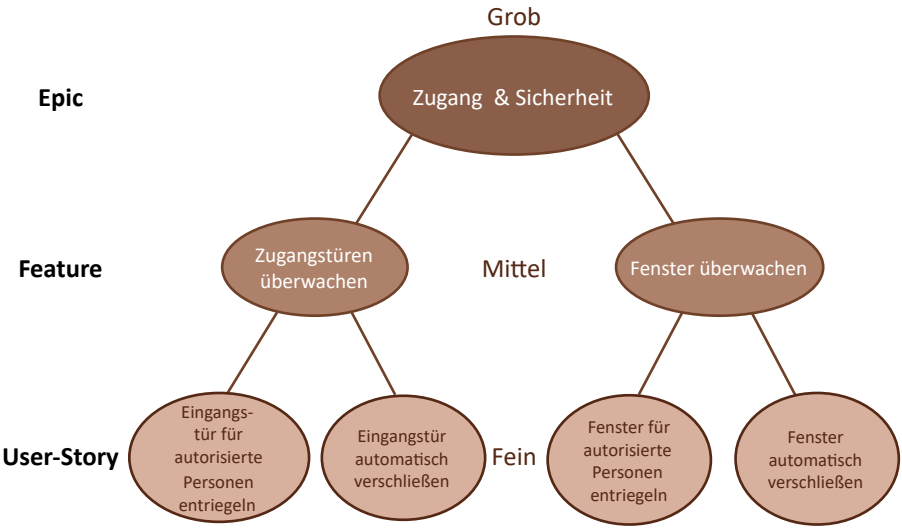


Abbildung 3.1: Beispiel für User-Stories zu einem Epic

Die im Product Backlog gesammelten Backlog Items müssen regelmäßig priorisiert werden. Dafür sind wir als Product Owner zuständig. Das sollte uns aber nicht davon abhalten, uns beraten zu lassen. Wenn die Priorität die Wichtigkeit eines Backlog Items für den/die NutzerIn widerspiegeln soll, dann sollten wir davon Abstand nehmen, selbst zu entscheiden, was für den/die NutzerIn wichtig ist. Außer natürlich wir selbst sind die zukünftigen NutzerInnen des Produkts. Mehr zum Priorisieren der Backlog Items finden Sie in Kapitel 6.4. In unserem Product Backlog müssen wir die Backlog Items entsprechend der Priorität sortieren. Das bedeutet, dass die Items mit der höchsten Priorität ganz oben im Backlog liegen.

Nach der Priorisierung wissen wir nun, welche Anforderungen (Backlog Items) die höchste Priorität haben. Entsprechend der Priorität werden wir die Items auch umsetzen. Doch bevor die Umsetzung starten kann, müssen wir noch einiges tun.

Denn es gilt die Backlog Items in einen Zustand zu bringen, in dem das komplette Scrum Team in der Aussage übereinstimmt, dass das Backlog Item fertig für die Umsetzung ist. Wir sagen auch das Backlog Item ist „Ready“.

Wann ein Backlog Item diesen Zustand erreicht hat, ist in der Definition of Ready (DoR) festgelegt. Die DoR enthält alle Kriterien für fertige Backlog Items und wird gemeinsam vom gesamten Scrum Team abgestimmt. Übliche und auch sinnvolle Inhalte der Definition of Ready sind:

- Das Backlog Item erfüllt das INVEST-Prinzip (siehe Kapitel 6.1)
- Das Backlog Item ist von allen Beteiligten verstanden (siehe Kapitel 6.2)
- Das Backlog Item enthält Akzeptanzkriterien (siehe Kapitel 5.2)
- …

Um diese Kriterien zu erfüllen, müssen wir Backlog Items zerlegen (siehe Kapitel 6.1) und mit Stakeholdern über die Backlog Items sprechen, um detailliertere

Informationen zu dem Backlog Item kennenzulernen. Wir müssen alles tun, um ein gemeinsames Verständnis hinsichtlich der Inhalte eines Backlog Items zu erlangen und festlegen, wann in unseren Augen das Backlog Item fertig umgesetzt ist. Wir analysieren also die Backlog Items im Detail und konkretisieren sie. Allerdings sollten wir es nicht zu weit treiben, denn wir wollen nicht alle Anforderungen bis ins kleinste Detail in den Backlog Items dokumentieren, sondern es sollte für die EntwicklerInnen ein vernünftiges Maß an Freiheit gelassen werden.

Diese Arbeiten erledigen wir immer für die Backlog Items, welche in naher Zukunft umgesetzt werden sollen. Dementsprechend ist das eine fortlaufende Tätigkeit. Denn immer wenn Backlog Items umgesetzt werden, rücken neue Backlog Items an die Spitze unseres Product Backlogs nach.

Ein wichtiger Punkt in der agilen Entwicklung ist das Schaffen eines gemeinsamen Verständnisses. Dazu werden wir regelmäßig mit den EntwicklerInnen (auch zusammen mit Stakeholdern) über die Backlog Items sprechen. Häufig werden dazu Refinement Meetings angesetzt, in denen man gemeinsam über die Backlog Items spricht, welche in naher Zukunft umgesetzt werden. Der Zeitraum zwischen dem Gespräch und der tatsächlichen Einplanung in einen Sprint sollte nicht zu lange sein, da sonst alles, was diskutiert wurde, bis dahin bereits wieder in Vergessenheit geraten ist. Was in solch einem Gespräch zu beachten ist, können Sie in Kapitel 6.2 nachlesen.

Wenn die Backlog Items dann den Status Ready erreicht haben, können diese bei einer der nächsten Sprintplanungen von den EntwicklerInnen in das Sprintbacklog gezogen werden. Dadurch wandern sie aus dem Product Backlog ab und machen Platz für die nächsten Backlog Items, welche nun die gleiche Prozedur durchlaufen. Im Sprint Backlog landen all die Backlog Items, bei denen die EntwicklerInnen denken, sie können diese im Sprint umsetzen. Dabei nehmen sie sich nicht wahllos die Items aus dem Backlog, sondern gehen entlang der Sortierung der Backlog Items im Product Backlog. Daher ist es sehr wichtig, dass wir das Backlog entsprechend sortiert haben.

Am Ende des Sprints werden die Ergebnisse, also das im Sprint erstellte Produktinkrement, in einem Review den Stakeholdern vorgestellt. Hier sehen die Stakeholder, wie sich das Produkt in dem Sprint weiter entwickelt hat und haben die Möglichkeit, Feedback zu geben und neue Anforderungen zu nennen. Das ist üblicherweise der Zeitpunkt, an dem der Lebenszyklus eines Backlog Items endet. Falls das Backlog Item später noch mal benötigt wird, kann es archiviert werden. Ansonsten wird es gelöscht.

Der letzte Akt in einem Sprint ist die Retrospektive. Diese dient dazu, die bisherigen Arbeitsweisen zu reflektieren und Ideen für die Verbesserung zu entwickeln. Die Retrospektive hat dabei die Zusammenarbeit im Team als Fokus und nicht das zu entwickelnde Produkt. Die Retrospektive ist sehr wichtig, da agiles Arbeiten eine kontinuierliche Verbesserung anstrebt.

4. Anforderungen ermitteln

Das Ermitteln der Anforderungen ist eine fortlaufende Tätigkeit. Sie wird bis zum Ende der Produktentwicklung durchgeführt. Gerade zu Beginn der Entwicklung werden viele Anforderungen zunächst grob ermittelt. Sobald sich die Umsetzung der Anforderungen nähert, werden erst dann die detaillierten Anforderungen ermittelt.

Die große Besonderheit in der Agilität besteht darin, dass wir beim Ermitteln der Anforderungen die bereits ermittelten Produktinkremente zur Hilfe nehmen können und auch sollten. Dadurch können die Stakeholder erleben, was aus den bisherigen Anforderungen wurde und spielerisch auf neue Anforderungen kommen. Gerade einige Qualitätsanforderungen (Performanz, Benutzbarkeit, etc.) können so leichter expliziert werden.

Dadurch, dass Stakeholder während der gesamten Produktentwicklung Feedback zu bereits bestehenden Produktinkrementen geben können, neue Anforderungen äußern können und auch Anforderungen ändern können, gibt es ihnen das Gefühl, bei der Entwicklung des Produkts aktiv mit eingebunden zu werden.

In der agilen Welt finden wir auch zunehmend Ansätze, welche die Stakeholder beim ermitteln von Anforderungen stärker involvieren. Schon mit den Stakeholdern gemeinsam eine Story Map [Rupp 21] zu erarbeiten, kann bisher unentdeckte Anforderungen an das Tageslicht bringen. Oder man durchlebt gemeinsam mit den NutzerInnen eine User Journey. Es gibt auch Ansätze, welche verschiedene Techniken kombinieren. Bekannt sind vor allem Design Thinking und Living Labs [Rupp 21]. Sind bestimmte Stakeholdergruppen nicht bekannt oder die Gruppe zu groß, um mit den einzelnen Personen sprechen zu können, dann bieten Personas und Crowd-RE vielversprechende Ansätze, um die Anforderungen dieser Stakeholder zu ermitteln. [Rupp 21]

5. Anforderungen dokumentieren

5.1 Das Product Backlog

Arbeitet man im nicht-agilen Umfeld typischerweise mit einer Anforderungsdokumentation (z.B. einem Lastenheft), findet man im agilen Umfeld die meisten Anforderungen in dem Product Backlog. Demnach ist das Product Backlog die zentrale Sammelstelle der Anforderungen. Allerdings muss nicht alles, was im Product Backlog ist, automatisch eine Anforderung sein. In dem Product Backlog findet man alles, was nach aktuellem Stand irgendwann mal im fertigen Produkt enthalten sein wird. Wichtig ist dabei die Betonung auf dem “aktuellem Stand”, denn nur weil heute etwas im Backlog ist, heißt es noch lange nicht, dass es morgen auch noch so sein wird.

Prinzipiell schreibt man dem Product Backlog folgende Eigenschaften zu.

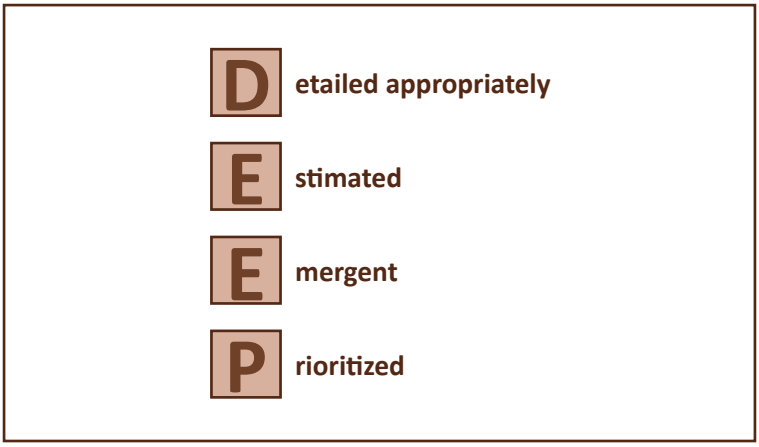


Abbildung 5.1: DEEP

Die Inhalte des Product Backlogs (die Product Backlog Items) sind entsprechend der aktuellen Bedürfnisse detailliert **(Detailed)**. Das bedeutet, dass die Items, die in naher Zukunft umgesetzt werden sollen, den für die Umsetzung benötigten Detaillierungsgrad haben. Andere Backlog Items dürfen zunächst weniger detailliert bleiben. Dies ist ein Unterschied zur klassischen Anforderungsspezifikation, bei welcher der Detaillierungsgrad der Anforderungen nicht vom Realisierungszeitpunkt abhängt.

Die Inhalte des Backlogs sollten geschätzt **(Estimated)** sein. Das ermöglicht den Product Owner eine Vorausplanung auch über einen langen Zeitraum hinaus. Mehr zum Schätzen siehe Kapitel 6.3 und zum Planen Kapitel 6.4.

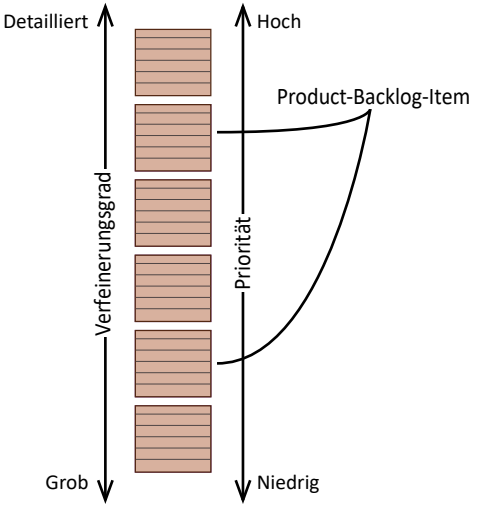


Abbildung 5.2: Ein Product Backlog

GIF www.sophist.de/bagd/k5a1/

Eine wichtige Eigenschaft des Backlogs ist die Emergenz **(Emergent)**. Das bedeutet, das Backlog ist stetigen Veränderungen unterworfen. So können sich jederzeit die Reihenfolge der Backlog Items oder aber auch die Backlog Items an sich verändern. Items können aus dem Backlog verschwinden oder jederzeit neue hinzukommen. Dementsprechend kann laut Definition ein Product Backlog niemals vollständig sein. Das ist unserer Meinung nach der herausragende Unterschied zu einer Anforderungsspezifikation, die zu irgendeinem Zeitpunkt vollständig sein sollte.

Das Product Backlog soll priorisiert **(Prioritized)** sein. Das bedeutet, dass der Product Owner sich des Wertes der einzelnen Product Backlog Items bewusst sein muss, um die Items entsprechend im Backlog zu sortieren. Denn das Product Backlog wird Iteration für Iteration von oben nach unten realisiert. Dementsprechend sollten die Items mit dem größtem Wert ganz oben stehen, damit sie zuerst abgearbeitet werden können.

5.2 Von Product Backlog Items und User Storys

Wie bereits im vorangegangenen Abschnitt beschrieben, finden wir im Product Backlog die Product Backlog Items (PBI).

Was die PBIs alles sein können, ist wie ein bunter Blumenstrauß und reicht von Epics, User Storys, Storys über Bugs, Incidents bis hin zu, naja ganz einfach, Product Backlog Items. Häufig wird der Begriff User Story oder einfach nur Story verwendet, da die Backlog Items, ähnlich wie eine kurze Geschichte eines zukünftigen Systembenutzers, beschrieben werden. Dazu gibt es eine Satzschablone, welche folgendermaßen aufgebaut ist:

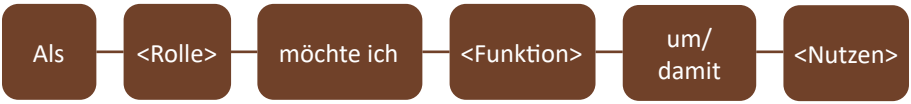


Abbildung 5.3: User-Story Template

Mit dieser Formulierung werden letztendlich drei W-Fragen beantwortet:

- Wer möchte eine Funktion?
- Was ist die gewünschte Funktion?
- Wozu wird diese Funktion benötigt?

Gerade der letzte Teil bereitet vielen angehenden Product Ownern einige Probleme bei der Formulierung. Allerdings ist diese Formulierung durchaus empfehlenswert, da wir uns Gedanken über das „Wozu" einer Funktion machen müssen. Denn immerhin investieren wir Zeit und Geld, um diese Funktion in einer Iteration umzusetzen. Aber nicht nur für die Existenzberechtigung des PBI ist diese Frage wichtig. Auch einen anderen wichtigen Zweck erfüllt die Antwort auf die Frage nach dem „Wozu". Für die Entwickler ist es ungeheuer wertvoll zu verstehen, welches Ziel der/die BenutzerIn mit der Funktion anstrebt. Dadurch ermöglicht diese Art der Formulierung eine zielgerichtete Realisierung des PBI durch das Entwicklungsteam.

Zu einem PBI gehören üblicherweise auch Akzeptanzkriterien. Diese werden vom Product Owner formuliert und beschreiben, nach welchen Kriterien er/sie das PBI am Ende des Sprints abnehmen wird. Akzeptanzkriterien werden konkreter formuliert als eine User Story und ergänzen bzw. detaillieren die Inhalte einer Story entsprechend. Demzufolge sollten die Akzeptanzkriterien nach bestimmten Mustern formuliert werden.

Egal welche Begriffe für die PBIs gewählt werden, eine wichtige Aufgabe des Product Owners ist es, die Backlog Items zu priorisieren und dafür mit einem Wert zu versehen. Der Product Owner muss möglichst viel Wert schaffen und ein gutes Verhältnis zwischen eingesetzten Ressourcen (die Arbeit des Scrum Teams) und dem Wert des Ergebnisses zu schaffen.

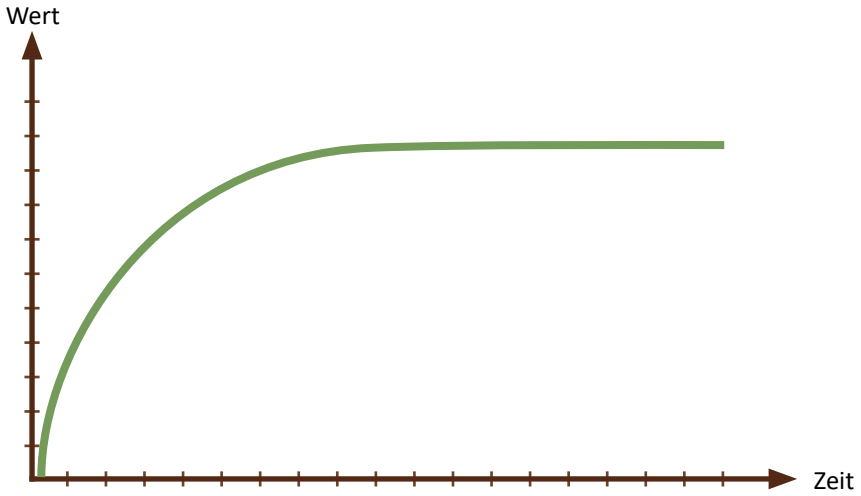


Abbildung 5.4: Wert-Kurve

Wie wir in der Grafik sehen, ist ein wichtiges Ziel des Product Owners in der ersten Zeit der Entwicklung des Produkts möglichst viel Wert zu schaffen. Logischerweise bleiben dann noch PBIs mit geringerem Wert übrig, wodurch die Kurve sich dann abflacht. Spätestens dann, wenn die Kurve keine Steigung mehr hat, sollte man sich überlegen ob nicht mit der weiteren Entwicklung aufgehört werden sollte. Diese Kurve ist allerdings eher idealtypisch als der Realität entsprechend. Durch die Priorisierung sollte ein Product Owner aber sich möglichst diesem Verlauf annähern.

6. Gute Anforderungen herleiten und vermitteln

6.1 Zerlegen von Backlog Items

Im Product Backlog können wir unterschiedlich detaillierte Product Backlog Items (PBI) unterscheiden. Eine übliche Art und Weise ist die Einteilung der Backlog Items in drei unterschiedliche Granularitätsebenen:

- Epics sind generell grobgranular, d.h. sie sind sehr groß und vage
  Beispiel: Als HausbewohnerIn möchte ich eine Überwachung des Gebäudes, um unerlaubten Zugang zu verhindern.
- Features sind mittelgranular, d.h. sie sind mittelgroß und etwas detaillierter als Epics, aber immer noch zu groß, um sie in einem Sprint umsetzen zu können
  Beispiel: Als HausbewohnerIn möchte ich eine automatisierte Kontrolle der Fenster und unerlaubten Zugang zu verhindern.
- User-Storys sind feingranular, d.h. sie sind klein und detailliert.
  Beispiel: Als HausbewohnerIn möchte ich, dass bei Abwesenheit der BewohnerInnen die Fenster geschlossen werden, damit kein Fenster offen bleibt.

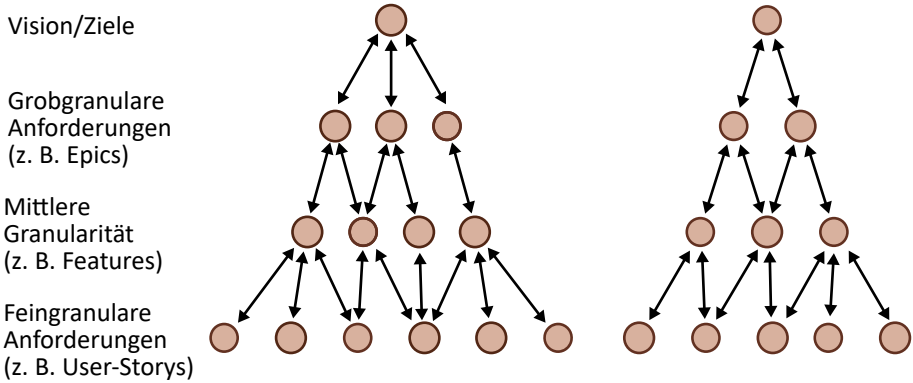


Abbildung 6.1: Eltern-Kind-Beziehung zwischen Anforderungen

Es gibt allerdings keine Kriterien, nach denen Sie Anforderungen in Epics, Features oder User-Storys einordnen können. Vielmehr handelt es sich um eine grobe Einsortierung. Es kann sich manchmal auch als sinnvoll erweisen, nur zwischen groben und feinen Anforderungen zu unterscheiden. Und wie bereits erwähnt, Epics, Features und User-Storys sind nur eine Möglichkeit, die Backlog Items zu unterscheiden.

Die Aufgabe ist es, die Backlog Items, also die Anforderungen, in eine sinnvolle Größe zu zerlegen. Dafür gibt es diverse Gründe:

- Anforderungen werden in Sprints umgesetzt. Daher sollte die Anforderung so detailliert sein, dass sie in einem Sprint umgesetzt werden kann.
- Die Bedürfnisse der Stakeholder sind üblicherweise konkreterer Natur. In unserem Beispiel möchten die HausbewohnerInnen nicht nur irgendein Smart-Home-System, sondern haben sicherlich konkretere Wünsche.
- Das Verständnis über den Inhalt der Anforderungen wir häufig erst klar, wenn sich das Team über die Details Gedanken macht.

Das sind nur ein paar der Gründe, warum wir grobe Anforderungen in detailliertere Anforderungen zerlegen.

Bei der Zerlegung der Backlog Items orientieren wir uns häufig an dem INVEST-Prinzip, welches 2003 von Bill Wake definiert wurde. Das INVEST-Prinzip beschreibt Eigenschaften oder Kriterien, welche Backlog Items erfüllen sollten. Dabei können wir grob sagen, je detaillierter ein Backlog Item ist, desto ernster nehmen wir das INVEST-Prinzip.

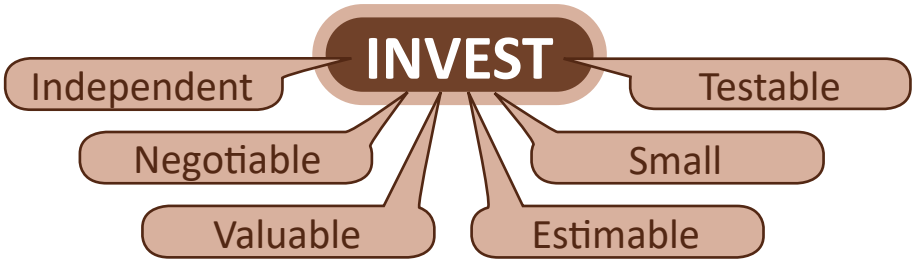


Abbildung 6.2: Das INVEST-Prinzip

Im Einzelnen bedeutet INVEST folgendes:

- **Independent**: Backlog Items sollen unabhängig voneinander sein. Das heißt, für die Umsetzung kann ein Backlog Item für sich alleine betrachtet werden und ohne dass weitere Backlog Items benötigt werden, auch umgesetzt werden.
- **Negotiable**: Backlog Items sind verhandelbar. Sie stellen keinen Vertrag dar, der genauso umgesetzt werden muss, sondern lassen den EntwicklerInnen ausreichend Spielraum, um in einem Sprint gemeinsam mit den Stakeholdern die Details zu erarbeiten.
- **Valuable**: Backlog Items liefern einen Wert. Üblicherweise einen Wert für den/die KundIn, beziehungsweise für den/die zukünftige/n BenutzerIn. Dabei ist es unerheblich, wie groß ein Backlog Item ist. Auch kleinste Backlog Items müssen einen Wert liefern. Falls dies nicht der Fall ist, dann sollten Sie die Existenz sowie die Umsetzung dieses Backlog Items ernsthaft in Frage stellen

- **Estimable**: Backlog Items müssen geschätzt werden (vgl. Kapitel 6.3). Diese Schätzung hilft zum Beispiel das Backlog Item in das Product Backlog einzusortieren, oder aber eine Aussage darüber treffen zu können, wie groß das Backlog Item ist. Damit das Backlog Item geschätzt werden kann, muss es für jeden auch ausreichend verständlich sein. Sonst ist eine Abschätzung nicht möglich.
- **Small**: Backlog Items sollen klein sein. Je kleiner sie sind, desto konkreter werden sie üblicherweise. Aber nicht nur das ist ausschlaggebend, sondern auch die Sprintlänge spielt dabei eine Rolle. Denn das Backlog Item soll ja in einem Sprint vollständig umgesetzt werden können. Dementsprechend müssen wir die Größe so wählen, dass das Item in den Sprint passt und dabei noch ausreichend Puffer vorhanden ist, damit nicht die kleinste Überraschung sofort die Umsetzung im Sprint gefährdet. Aber Vorsicht. Tendieren Sie nicht dazu, die Backlog Items zu klein zu schneiden. Denn dadurch produzieren Sie nur erhöhten Verwaltungsaufwand, ohne einen entsprechenden Mehrwert zu erzielen.
- **Testable**: Ein Backlog Item soll testbar sein. Das bedeutet, dass es ausreichend gut verstanden ist um zu beschreiben, wie überprüft werden kann ob das Backlog Item den Wünschen entsprechend umgesetzt worden ist. Dazu eignet sich die Formulierung von Akzeptanzkriterien zu den jeweiligen Backlog Items

Bei einigen Punkten des INVEST-Prinzips fragen Sie sich sicherlich, wie Sie diese bei groben Backlog Items erfüllen können. Diese Frage ist durchaus berechtigt. Daher gilt das komplette INVEST nur für detaillierte Backlog Items. Gröbere Backlog Items sollten aber zumindest den ersten drei Punkten, also dem INV genügen.

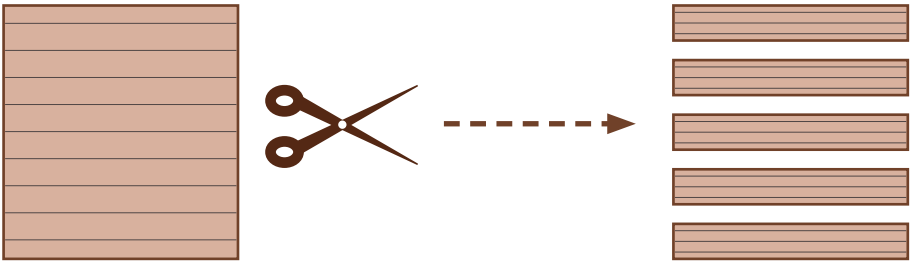


Abbildung 6.3: Vom Groben ins Feine

Sobald der Zeitpunkt näher rückt, zu dem Backlog Items umgesetzt werden sollen, müssen Sie dafür sorgen, dass diese eine vernünftige Größe haben. Allerdings stehen Sie dabei nicht vollkommen alleine da. Natürlich können Sie sich auch Hilfe von EntwicklerInnen holen. Zum Beispiel werden in einem gemeinsamen Backlog Refinement häufig Backlog Items zerlegt. Aber alles können wir nicht immer in der großen Runde machen. Schließlich haben die EntwicklerInnen auch andere Aufgaben. Zum Beispiel aus Backlog Items ein System zu bauen.

Das bedeutet, Sie kommen nicht darum herum, auch selbstständig Backlog Items zu zerlegen oder zu verfeinern – je nachdem welchen Begriff Sie verwenden möchten. Man könnte sagen, dass wir Vorarbeit leisten, bevor wir in der gemeinsamen Runde den Backlog Items den Feinschliff geben. Dabei sollten Sie auf jeden Fall das INVEST-Prinzip im Auge behalten, damit auch die detaillierten Backlog Items gute Backlog Items sind. Es gibt eine Vielzahl von Ansätzen nach welchen Kriterien ein grobes Backlog Item klein geschnitten werden kann. Zum Beispiel können Sie sich nach Workflow Schritten richten, oder nach den Daten etc.. Auf unserer Webseite haben wir ein paar Tipps und Tricks zum Schneiden von User-Storys (also Backlog Items) beschrieben.

Seien Sie sich bewusst, dass ein Ansatz zum Schneiden von Backlog Items, der einmal gut funktioniert hat, nicht automatisch immer funktioniert. Je nach Gegebenheit werden Sie variieren müssen. Allerdings werden Sie im Laufe der Zeit merken, dass das Zerlegen immer leichter fällt. Denn Sie werden immer weiter dazu lernen. Grundsätzlich raten wir aber davon ab, nach System- oder Softwareeinheiten zu schneiden. Es mag für die EntwicklerInnen leichter scheinen, wenn sie Backlog Items für das Frontend und Backlog Items für das Backend haben. Aber Sie verlieren dabei schnell das Ziel aus dem Auge, nämlich ein für den/die zukünftige/n NutzerIn hilfreiches, funktionierendes und qualitativ hochwertiges Produkt.

6.2 Backlog Items besprechen

Product Backlog Items sind entsprechend des 3C-Prinzips (Card, Conversation, Confirmation) ein Kommunikationsversprechen.

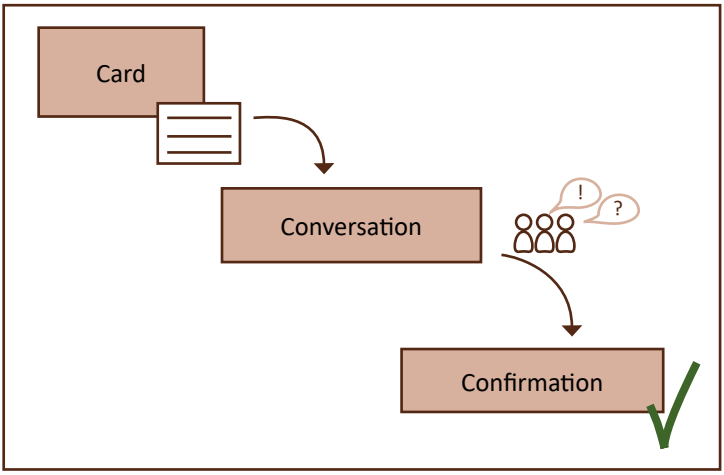


Abbildung 6.4: Das 3-C-Modell

Also primär dafür gedacht, dass man darüber redet. Wichtige Gespräche zu den Backlog Items finden zwischen dem Product Owner und den EntwicklerInnen statt. Denn der Product Owner hat das Wissen zu den Informationen hinter dem PBI und das Entwicklungsteam benötigt dieses Wissen bzw. dieses Verständnis, um das PBI umzusetzen.

Für diese Gespräche gibt es in der agilen Welt verschiedene Zeitpunkte, zu denen diese stattfinden können. Zum Beispiel im Rahmen einer Sprintplanung oder wahrscheinlich intensiver in einem Refinement Meeting zwischen PO und EntwicklerInnen.

Ziel dieser Diskussionen ist es, dass

- PO und Entwicklungsteam ein gemeinsames Verständnis zu dem PBI haben
- Offene Punkte zu dem PBI identifiziert wurden
- Im Falle eines PBI, welches im kommenden Sprint umgesetzt werden soll, ob die Details zu dem PBI allen bekannt sind

Meistens werden diese Diskussionen in der Art geführt, dass der Product Owner das Backlog Item öffnet, kurz beschreibt und dann wird diskutiert. Möglicherweise haben Sie bereits die Erfahrung gemacht, dass diese Diskussionen nicht ganz zufriedenstellend verlaufen.

- Schnell driftet man in der Diskussion in Themen ab, die nicht in das PBI gehören
- Es wird bereits ausführlich die Lösung diskutiert, obwohl die Problemstellung noch gar nicht bewusst ist
- Nach einer halben Stunde Diskussion weiß das Team schon nicht mehr, welche Entscheidungen zu Beginn getroffen wurden
- Die Diskussionsrunde verliert das Ziel des Gesprächs aus den Augen

Es kann also viel Zeit investiert werden, das Ergebnis aber nicht angemessen ausfallen. Deswegen sollten Sie sich für diese Gespräche im Vorfeld schon Zeit nehmen und sie auch vorbereiten.

Haben Sie schon mal überlegt, die Diskussion mit einer methodisch gut vorbereiteten Geschichte einzuleiten (Storytelling [Rupp 21]), oder sogar Videos [Link auf Blogserie]einzusetzen? Vielleicht möchten Sie die Diskussion dadurch strukturieren, indem Sie gemeinsam mit dem Team ein weiteres Entwicklungsartefakt (z.B. Testfälle) erstellen? Dies sind nur ein paar Beispiele der verschiedenen Methoden für ein gelungenes Durchführen eines Meetings zur Diskussion der PBIs, welche wir erfolgreich in unseren Projekten einsetzen. Wie bei uns die Vermittlung der Backlog Items beispielhaft ausgesehen hat, können Sie in ein paar kurzen Videos hier in dieser Blogserie erfahren.

6.3 Schätzen

Das Schätzen von Backlog Items ist per se keine Requirements-Engineering Tätigkeit. Allerdings benötigen wir die Schätzung für andere Requirements-Engineering Aufgaben (z.B. Prüfen der Anforderungen auf Verständlichkeit) und daher widmen wir dem Schätzen auch ein paar Zeilen.

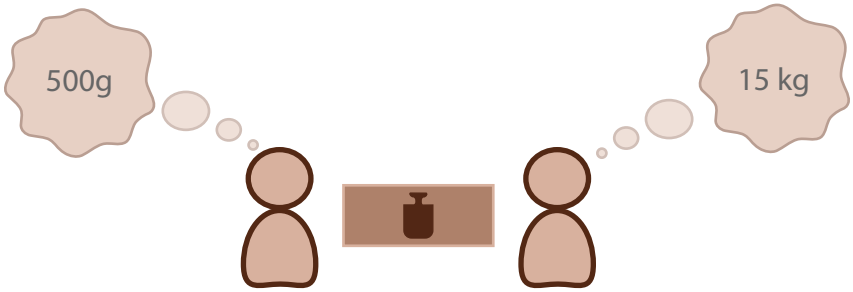


Abbildung 6.5: Schätzen

Die Backlog Items müssen regelmäßig auf ihre Größe geschätzt werden. Das dient nicht nur der Erfüllung der Eigenschaften eines Backlogs (DEEP siehe Kapitel 5.1), sondern hat auch andere Gründe. Zunächst erlaubt die Schätzung eine Aussage darüber, ob ein Backlog Item klein genug ist, dass es in einem Sprint umgesetzt werden kann oder ob es noch weiter zerlegt werden muss. Außerdem lässt sich eine Aussage treffen, wie einfach oder kompliziert ein Backlog Item ist. Auch kann mittels einer Schätzung eine Releaseplanung erstellt werden (siehe Kapitel 8).

Der wohl wichtigste Grund, der für eine Schätzung spricht, ist nicht das Schätzergebnis an sich, sondern das Schätzen selber. Die Schätzung wird im Agilen nicht von einzelnen Personen, sondern immer von Gruppen durchgeführt. Sinnvollerweise von jenen, welche die Backlog Items umsetzen müssen, also den EntwicklerInnen. Wenn nun mehrere Personen ein Backlog Item schätzen, kommt es logischerweise zu Abweichungen in den Schätzungen der einzelnen Personen untereinander. Sind diese Abweichungen größer, dann ist dies ein Indiz dafür, dass in der Gruppe unterschiedliche Vorstellungen existieren, was sich hinter einem Backlog Item verbirgt. Dies ist für uns ein klarer Hinweis, dass hier noch Klärungs- und Abstimmungsbedarf existiert.

Es ist einfacher, relativ zueinander zu schätzen, als absolut. Stellen wir uns zwei Personen auf einem Foto vor. Wenn wir diese Personen nicht kennen, können wir nicht sagen, wie groß diese Personen sind. Aber wir können Aussagen darüber treffen, wie sich die Größe der Personen zueinander verhält. Zusätzlich nimmt man durch ein relatives Schätzen die Sorge aus dem Spiel, dass man einen absoluten Aufwand schätzen muss, an dem man nachträglich gemessen wird. Dementsprechend schätzen wir nicht den Aufwand der sich hinter einem Backlog Item verbirgt, sondern die Komplexität.

Beim Schätzen wird die Komplexität eines Backlog Items festgelegt. Diese wird dann in Story-Points, T-Shirt Größen oder anderen Einheiten festgehalten, damit sie nicht als Aufwand gesehen wird und keine Rückschlüsse auf Leistung und ähnliches gezogen werden können.

Für das Schätzen gibt es verschiedene Methoden, wie Planning Poker, T-Shirt Größen, Magic Estimation oder Wall Estimation, um die bekanntesten zu nennen.

6.4 Priorisieren

Im Kapitel 5.1 „Das Product Backlog“ haben wir gesagt, dass die Product Backlog Items sortiert im Product Backlog liegen. Die Sortierung der Backlog Items ist eine Aufgabe des Product Owners und verfolgt zwei maßgebliche Ziele

- Die Priorität eines Backlog Items sichtbar zu machen
- je weiter oben ein Backlog Item im Backlog liegt, desto höher ist dessen Priorität
- Die Konzentration auf die wichtigen Backlog Items zu lenken

je weiter oben ein Backlog Item im Backlog liegt, desto stärker befassen wir uns mit diesem

Für die Sortierung müssen wir also Kenntnisse über die Priorität eines Backlog Items haben. Als Product Owner stehen wir aber nicht alleine da, was die Bestimmung der Priorität betrifft. Je nachdem was die Priorität bedeutet, können wir uns sinnvolle Hilfe von Stakeholdern oder den EntwicklerInnen holen. Es liegt nur in unserer Verantwortung, dass die Priorität bestimmt wird.

Um eine Priorität festzulegen, müssen wir uns der Kriterien bewusst werden, anhand derer wir die Priorität bestimmen werden. Die Kriterien ermitteln wir anhand der Frage, was uns im Moment denn wichtig ist. Das könnte zum Beispiel sein, dass wir dem/der zukünftigen NutzerIn des Systems möglichst hilfreiche Funktionen zur Verfügung stellen wollen oder aber, dass uns bestimmte Qualitätsanforderungen sehr wichtig sind. Dies sind nur zwei Beispiele, was mit der Priorität ausgedrückt werden soll.

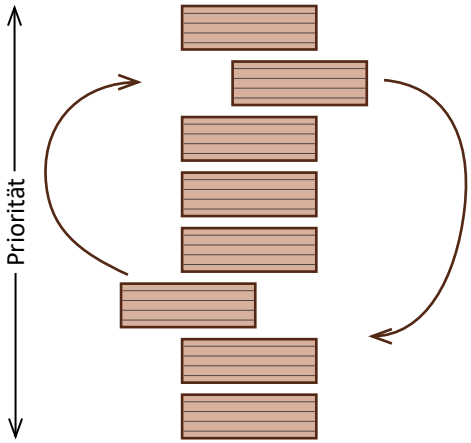


Abbildung 6.6: Sortieren des Product Backlogs entsprechend der Priorität

Prinzipiell ist das oberste Gebot, dass wir die Priorität nach dem Wert für den/die zukünftige/n NutzerIn ausrichten. Denn das sind diejenigen, für die wir das Produkt entwickeln. Natürlich kann es Fälle geben (und wir sind uns sicher die wird es geben) bei denen andere Kriterien eine größere Rolle spielen. Aber immer wenn es keinen besonderen Grund gibt, davon abzuweichen, sollte die Nutzer-/Kundenzufriedenheit im Vordergrund stehen.

Für die Priorisierung selbst bieten sich diverse Techniken an. Bekannte Techniken sind MoSCoW, Kano-Modell, lineare Priorisierung oder Weighted Shortest Job First.

Wichtig ist, dass die Priorisierung keine einmalige Sache ist, sondern regelmäßig durchgeführt wird. Denn jeden Tag können neue Anforderungen auftauchen, die dann entsprechend priorisiert werden müssen und es kann sich das Kriterium ändern, nachdem wir die Priorität festlegen wollen.

7. Product Backlog und weitere Artefakte

Ein zentrales Artefakt in der Agilität ist das Product Backlog. Dort finden wir alle aktuell bekannten Anforderungen. Wenn wir ehrlich sind, aber nicht alle. Einige Anforderungen sind an anderer Stelle besser aufgehoben, zum Beispiel können übergreifende Qualitätsanforderungen oder Constraints in der Definition of Done plaziert werden. Auch zusätzliche Anforderungsdokumente gerade für diese Anforderungen sind durchaus sinnvoll.

Neben den Anforderungen gibt es noch weitere Möglichkeiten Wissen zu dokumentieren. So kann neben dem Product Backlog möglicherweise ein Datenmodell, in Form eines Klassendiagramms, hilfreich sein, Vielleicht helfen bei der Diskussion über die Backlog-Items auch Prozessbeschreibungen um die Backlog Items in ihrem Kontext zu sehen. Diese könnten mit Business Process Model and Notation oder Aktivitätsdiagrammen dokumentiert sein. Es könnte für die Entwickler hilfreich sein zu verstehen, was für Menschen die späteren Nutzer sind. Dazu könnten eine Reihe dokumentierter Personas dienen. Dies ist nur ein kleiner Ausblick auf diverse zusätzlicher Dokumentationen zum Product Backlog. Erlaubt und erwünscht ist alles, was hilft und sinnvoll ist, um das gemeinsame Verständnis zu den Bedürfnissen der Stakeholder zu stärken und zu fördern.

8. Roadmap, Releaseplanung

8.1 Roadmap

Durch das Arbeiten in Sprints, liegt der Fokus der Arbeiten häufig auf der aktuellen und den kommenden 1-2 Iterationen. Das bedeutet wir können Aussagen darüber treffen, was in der aktuellen Iteration bearbeitet wird und was der Scope der kommenden Iteration sein wird. Möglicherweise ist auch der Scope der übernächsten Iteration absehbar, aber üblicherweise sehen wir nicht weiter in die Zukunft.

Allerdings kann auch die ferne Zukunft interessant für uns sein. Möglicherweise gibt es Einschränkungen hinsichtlich des Budgets oder der Termine. NutzerInnen möchten Wissen, wann sie mit welchen Features im Produkt rechnen können. EntwicklerInnen interessieren sich dafür, was in naher und ferner Zukunft noch alles auf sie zukommt. Das sind nur einige wenige Gründe, warum wir auch in der agilen Welt hin und wieder einen Blick in die Zukunft werfen müssen.

Um das zu ermöglichen brauchen wir

- Die Backlog Items
- Die Priorität der Backlog Items
- Eine Schätzung zu den Backlog Items
- Die Velocity der EntwicklerInnen

Mithilfe dieser vier Elemente können wir nun die Backlog Items auf einem Zeitstrahl anordnen und einigermaßen sinnvoll den Zeitpunkt zu dem wir die Umsetzung eines Backlog Items vermuten, abschätzen. Dazu kann es hilfreich sein, den Zeitstrahl in überschaubare Abschnitte einzuteilen, zum Beispiel ein Jahr in vier Abschnitte.

Diesen Abschnitten werden dann die Backlog Items zugeordnet. Dabei berücksichtigen wir die Priorität des Backlog Items, die Schätzung und die Velocity.

Dadurch erreichen wir eine ungefähre Planung der Backlog Items auch über einen längeren Zeithorizont hinweg.

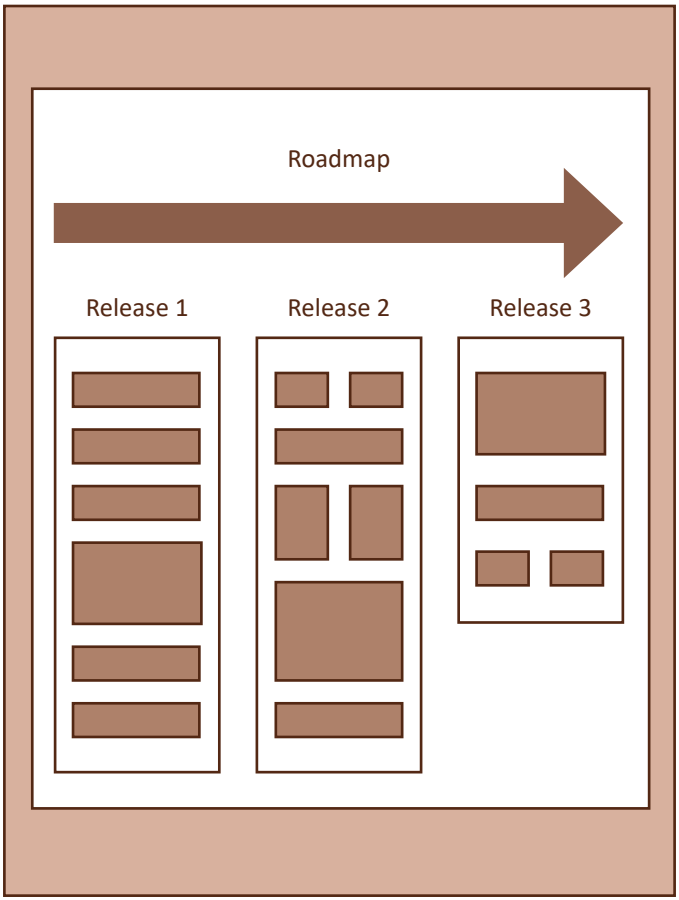


Abbildung 8.1: Releaseplanung mit Roadmap

Wichtig: Diese sogenannte Roadmap spiegelt nur die aktuelle Einschätzung wieder. Sie muss regelmäßig aktualisiert werden und darf nicht als eine Verpflichtung angesehen werden, die das Scrum Team eingeht.

Prinzipiell gilt, je weiter der Plan in die Zukunft reicht, desto ungenauer wird er sein. In einem Jahr kann schließlich viel mehr Unvorhergesehenes passieren, als in den nächsten zwei Wochen. Daher ist es wichtig, den Plan auch regelmäßig zu prüfen und gegebenenfalls zu aktualisieren. Nur so erkennen Sie frühzeitig, ob der Plan aufgehen wird oder nicht.

Falls der Plan nicht aufgeht, dann ist das ein Zeichen dafür, dass Sie sich in der Realität befinden. Wobei wir bei dieser Aussage davon ausgehen, dass Ihr Plan weiter in die Zukunft reicht, als vielleicht 1-2 Monate.

Sobald wir merken, dass der Plan nicht aufgeht, haben wir verschiedene Optionen darauf zu reagieren:

- Verändern des Umfangs des Produkts
- Verschiebung des Fertigstellungstermins
- Weitere Teams für die Umsetzung einsetzen

Hierbei sind die ersten beiden Optionen üblicherweise einfacher zu bewerkstelligen, da das Hinzuziehen weiterer Mitarbeiter natürlich eine Einarbeitungszeit benötigt. Das trifft aber nicht in allen Fällen zu. Manchmal ist einfach der Endtermin unverrückbar vorgegeben.

8.2 Entwicklungsstrategien

Direkt im Zusammenhang der Roadmap steht eine mögliche angestrebte Entwicklungsstrategie. Denn nicht immer ist es sinnvoll, die Backlog Items in genau der Reihenfolge abzuarbeiten, welche die Priorität eigentlich vorgibt bzw. vorgeben würde.

Möglicherweise fangen wir zunächst mit den Backlog Items an, die wir am schnellsten umsetzen können. Oder wir wollen so früh wie möglich den Stakeholdern etwas zeigen können. Vielleicht wollen wir aber erst einmal etwas ausprobieren, um dann herauszufinden, wie die zukünftigen NutzerInnen damit umgehen werden.

Es gibt also verschiedene Gründe sich für eine bestimmte Reihenfolge der Abarbeitung von Backlog Items zu entscheiden. Je nachdem welcher Grund für uns eine Rolle spielt, werden wir eine bestimmte Entwicklungsstrategie wählen.

Übliche Entwicklungsstrategien sind:

- Minimum Viable Product (MVP)
- Minimum Marketable Product (MMP)
- Low Hanging Fruits/Quick Wins
- Weighted Shortest Job First (WSJF)
- Risk Reduction

9. Einführung von Agilität

Stehen Sie vor der Herausforderung, überhaupt erst agile Entwicklungspraktiken zu etablieren, haben Sie einiges an Arbeit vor sich.

Zunächst gilt es, sich der Frage zu stellen, wie agil es denn werden soll? Denn nicht immer arbeiten wir komplett agil. Manchmal verwenden wir hybride Ansätze, um Vorteile der Agilität zu nutzen, wenn wir gleichzeitig aufgrund der Rahmenbedingung nicht vollständig agil arbeiten können.

Ein weiterer Aspekt der interessant ist, ist die Frage, wie führen wir die Agilität ein? Wir können einen Top-Down Ansatz wählen, bei dem die Agilität vom Management ausgehend eingeführt wird. Oder wir wollen lieber den Bottom-Up Ansatz, bei dem die Mitarbeiter selber an der Gestaltung des agilen Arbeitens beteiligt sind. Gerade letzteres bietet auch die Chance, die Einführung der Agilität an sich agil zu gestalten. Hier gibt es neuere interessante Ansätze wie zum Beispiel das Open Space Agility (OSA).

Eine große Hürde bei der Einführung der Agilität ist häufig die Änderung des Mindsets bei den Betroffenen. EntwicklerInnen arbeiten jetzt selbstorganisiert, im Fokus steht das Produkt und zwar das für den/die NutzerIn/KundIn beste Produkt. Wir haben höhere Transparenz bezüglich wer an welchem Thema arbeitet. Es geht stärker um ein gemeinsames Verständnis als konkrete Vorgaben, was genau gebaut werden soll. Diese Veränderung in der Denkweise hinzubekommen, ist schon eine Herausforderung.

Eine weitere Herausforderung ist die Umgebung. Damit meinen wir die Menschen, welche zum Beispiel nicht Teil des Scrum-Teams sind. Nehmen wir zum Beispiel die Stakeholder. Für sie kann die Agilität komplett fremd sein, weil sie möglicherweise selbst sehr stark Wasserfall getrieben arbeiten.

Da wir häufig bei der Einführung von Requirements Engineering und in letzter Zeit gerade im agilen Kontext beteiligt sind, haben wir unsere Erfahrungen in einem Innovationprojekt zusammengetragen, um ein Paket an Maßnahmen zu haben, mit dem die Einführung gelingen kann.

Im folgenden Video erklären wir Ihnen das agile Change-Management:

10. Agiles Arbeiten mit mehreren Teams

Damit Teams gut zusammenarbeiten können, dürfen sie nicht zu groß sein. Man sagt, dass ab ca. 9 Personen ein Team zu groß wird, um effektiv arbeiten zu können. Wenn das zu entwickelnde Produkt einen Umfang hat, bei dem ein Team relativ viel Zeit benötigen würde, um das Produkt zu bauen, dann kann es sinnvoll sein, das Produkt mit mehreren Teams gleichzeitig zu erstellen.

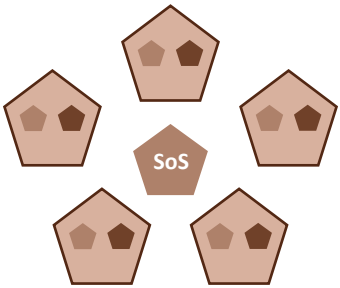
Arbeiten mehrere Teams an einem Produkt, kommen zusätzliche Herausforderungen auf uns zu:

- Zuständigkeiten der Teams müssen geklärt werden
- Abhängigkeiten zwischen den Teams müssen sichtbar gemacht und reduziert werden.
- Umgang mit teamübergreifenden Themen (z.B. einheitliches Design der Benutzeroberfläche)
- Überblick über das Produkt als Ganzes wahren
- Integration der Ergebnisse der einzelnen Teams zu einem Produkt

10.1 Scrum@Scale

Das Metamodell [Scrum@Scale] nutzt Scrum und skaliert es für mehrere Teams. Dabei haben die EntwicklerInnen dieses Metamodells darauf geachtet, möglichst wenig zusätzliche Regeln als in Scrum selbst festzulegen.

Scrum@Scale legt die Teamgröße auf maximal 5 EntwicklerInnen, einen Product Owner und einen Scrum Master fest. Mehrere solcher Teams bilden wiederum eine Einheit, wobei eine Einheit wiederum bis zu 5 Teams hat. Bei mehr als 5 Teams werden mehrere Einheiten gebildet (vgl. Abbildung 10.1).



Scrum@Scale mit 5 Teams

Abbildung 10.1: Scallierung in Scrum@Scale

Nehmen wir den Fall, wir haben bis zu 5 Teams die an einem Produkt arbeiten. Dementsprechend haben wir auch 5 Scrum Master und 5 Product Owner. Die Scrum Master bilden wiederum ein Scrum Master Team und die Product Owner ein Product Owner Team. Wir sprechen hier vom Scrum of Scrum, oder kurz SoS. Das Scrum Master Team arbeitet gemeinsam daran die Zusammenarbeit zu verbessern und das Product Owner Team sorgt dafür, dass das Product Backlog gefüllt, verfeinert und priorisiert wird. Falls es im Product Owner Team schwierig wird, Entscheidungen zu fällen, gibt es einen Chief Product Owner, der dann das letzte Wort hat.

10.2 LeSS

[LeSS] steht für Large Scaled Scrum. Die Idee dahinter ist es, das klassische Scrum für mehrere Teams zu erweitern und dabei möglichst wenige Änderungen oder Einschränkungen vorzunehmen. Das Ziel ist die Flexibilität und die Prinzipien von Scrum zu behalten, aber auf die Arbeit mit mehr Personen zu skallieren.

Einführung in das LeSS-Framework:

Das Framework ist für bis zu acht Teams vorgesehen. Für den Fall, dass mehr Teams am gleichen Produkt arbeiten müssen, gibt es LeSS Huge, eine Erweiterung von LeSS, die zusätzliche Rollen einführt.

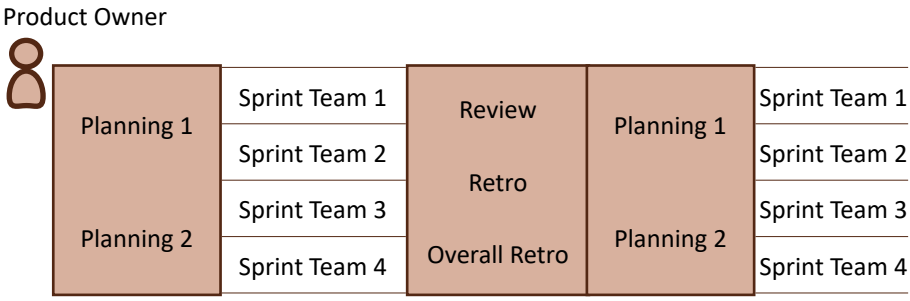


Abbildung 10.2: Arbeiten mit dem LeSS-Framework

Die Teams bei LeSS sind so zusammengesetzt, wie bei einem herkömmlichen Scrum. Allerdings haben alle Teams den gleichen PO und arbeiten mit dem gleichen Product Backlog. Die Scrum Master können je nach Bedarf ein oder mehrere Teams betreuen. In Scrum entwickelt ein interdisziplinäres Team ein Produkt. In LeSS entwickeln mehrere interdisziplinäre Teams gemeinsam ein Produkt, wobei sich jedes Team auf eine andere, vom Kunden gewünschte, Funktionalität (Feature) konzentriert. Daher wird in LeSS auch von Feature Teams gesprochen. Die Arbeit der Feature Teams läuft synchronisiert ab, indem alle die gleiche Sprintlänge und den gleichen Start- und Endzeitpunkt für die Sprints haben.

Das Sprint Planning findet in zwei Phasen statt. Im Sprint Planning 1 treffen sich die Teams oder Repräsentanten der Teams mit dem Product Owner, um das Sprintziel zu klären und zu entscheiden, welche Product Backlog Items von welchem Team übernommen werden sollen. Jedes Team führt ein eigenes Sprint Backlog und plant im Sprint Planning 2, wie es die gewählten Product Backlog Items umsetzen wird.

Product Backlog Refinement sollte nach Möglichkeit mit allen Teams oder zumindest mehreren Teams gemeinsam geschehen, da es das gemeinsame Verständnis der Produktvision und die Kommunikation zwischen den Teams fördert. Wenn sich alle Feature Teams zu einem gemeinsamen Refinement versammeln, können bei LeSS bis zu 72 Leute zusammenkommen. Hinzu kommen noch Product Owner und, je nach Bedarf, Stakeholder, Nutzer oder Fachexperten, um Rückfragen zu klären und Anforderungen zu präzisieren. Damit ein Workshop in dieser Größe funktioniert, bedarf es einer guten Vorbereitung und Moderation. Gerne beraten wir Sie bei der Planung agiler Workshops und helfen Ihnen dabei herauszufinden, welche der vielen Techniken und Methoden zum Wissenstransfer in Gruppen und zur Vermittlung und Dokumentation von Anforderungen für Sie geeignet sind.

Ähnlich wie das Sprint Planning, findet auch die Retrospektive in zwei Teilen statt. Zunächst führt jedes Team eine eigene Retrospektive, wie im klassischen Scrum, durch. Neben der Zusammenarbeit im eigenen Team, wird dabei aber auch über die Arbeit zwischen den unterschiedlichen Teams gesprochen, um übergreifende Hindernisse zu identifizieren. Anschließend findet die Overall Retrospektive statt, bei der sich PO, Scrum Master, Repräsentanten der Teams und, bei Bedarf, auch des Managements, treffen. Thematisiert werden teamübergreifende Hindernisse in der Zusammenarbeit oder strukturelle Probleme in der Organisation, die die Teams an der Ausübung ihrer Aufgaben hindern. Auch Erfolgsrezepte und Best Practices, die ein Team entdeckt hat, können hier ausgetauscht werden.

LeSS Huge – Erweiterung von LeSS auf mehr als 8 Teams

Um die Kommunikation und einen reibungslosen Ablauf bei der Zusammenarbeit mit vielen Teams sicher zu stellen, führt LeSS Huge eine neue Rolle ein. Der Product Owner bekommt Unterstützung durch mehrere Area Product Owner (APO), die jeweils für einen Teilbereich der geplanten Funktionalität des Gesamtprodukts verantwortlich sind. Wie bei den einzelnen Product Backlog Items, werden auch die Themenbereiche für die Zuweisung der APOs nach Fachlichkeit geschnitten, um eine kundenorientierte Funktionalität als Ganzes zu betrachten und umzusetzen.

10.3 Nexus

Ähnlich wie LeSS baut auch [Nexus] direkt auf Scrum auf und macht relativ wenige Änderungen. Im Gegensatz zu LeSS werden aber ein paar mehr neue Rollen, Artefakte und Events definiert.

Nexus arbeitet mit 3-9 Entwicklerteams, welche alle den gleichen Sprintstart und Ende haben. Für alle Teams gibt es genau einen Product Owner. Der wird häufig durch Requirements Engineers, die in den Teams sitzen, unterstützt.

Eine Besonderheit des Nexus ist das Nexus Integration Team (NIT). Dieses Team besteht aus dem Product Owner, einem Scrum Master und weiteren Personen, welche auch aus den Entwicklerteams stammen können. Die Aufgabe des NIT ist es, die Integration der Ergebnisse der einzelnen Teams in ein gemeinsames Produkt zu unterstützen. Das NIT hat dabei eine eher coachende Rolle.

Die Sprintplanung besteht aus zwei Teilen: dem Nexus Sprint Planning für alle Teams, in dem Backlog Items auf die Teams aufgeteilt und priorisiert werden, und dem individuellen Sprint Planning der einzelnen Teams im Anschluss

Neben dem Product Backlog und den Sprint Backlogs (für die einzelnen Teams) gibt es in Nexus auch noch das Nexus Backlog. Das Nexus Backlog wird dazu verwendet, um Abhängigkeiten zwischen den Aufgaben der Teams im Auge zu halten und enthält nur die Backlog Items, welche Abhängigkeiten haben.

Eine zusätzliche Erweiterung ist das Nexus Daily Scrum, bei dem aus jedem Team ein Repräsentant auftritt, um kurz über mögliche Abhängigkeiten bei den aktuelle Arbeiten zu sprechen.

Genauso wie bei LeSS ist auch bei Nexus die Retrospective zweigeteilt. Eine Retrospective für jedes Team und eine gemeinsame Retrospective, um die Zusammenarbeit der Teams zu beleuchten.

10.4 SAFe

Das [SAFe] Framework ist darauf ausgelegt, ein ganzes Unternehmen agil zu gestalten. Es sieht vor, dass mehrere Produkte mit vielen Teams entwickelt werden und definiert mehrere Ebenen, um die Zusammenarbeit zu steuern und zu synchronisieren. In dieser Beschreibung starten wir unten auf der Team Ebene und arbeiten uns langsam nach oben bis zum Management.

Team Ebene

- Mehrere Scrum-Teams arbeiten zusammen an einem Produkt. Es gibt ein gemeinsames Product Backlog aber jedes Team hat ein eigenes Sprint Backlog, einen eigenen PO und SM.
- Es gibt keine Beschränkung wie viele Teams es gibt.
- Mehrere Sprints werden zu einem Produktinkrement von meist 3-4 Monaten zusammengefasst. Am Ende eines Produktinkrements muss ein integriertes Inkrement stehen.

Programm Ebene

Der Program Manager entwickelt aus den Epics der höheren Ebenen mittelgroße Anforderungen (Features). Diese werden priorisiert und dann in die Programm Inkremente eingeplant. Dem Program Manager stehen weitere Rollen als Unterstützung zur Seite, zum Beispiel System Architekten oder Test Manager.

Large Solution Ebene

Diese Ebene wird benötigt, wenn der Betrachtungsgegenstand ein größerer ist. Wenn wir zum Beispiel eine Smarte Fabrik bauen wollen, dann wird diese Ebene verwendet, um verschiedene Programm Ebenen miteinander zu koordinieren.

Portfolio Ebene

Die oberste Ebene im Unternehmen – hier werden üblicherweise strategische Entscheidungen getroffen, welche in Form von Epics in das Product Backlog gekippt und priorisiert werden. Häufig wird diese Rolle vom Unternehmensvorstand erfüllt.

11. Quellenverzeichnis

[LeSS] https://less.works

[Nexus] https://www.scrum.org/resources/nexus-guide

[Rupp 21] Rupp C. und die SOPHISTen: Requirements-Engineering und -Management. Das Handbuch für Anforderungen in jeder Situation, 7. Auflage, München 2021

[SAFe] https://www.scaledagileframework.com/

[Scrum@Scale] https://www.scrumatscale.com/scrum-at-scale-guide/

SOPHIST Eigenproduktionen

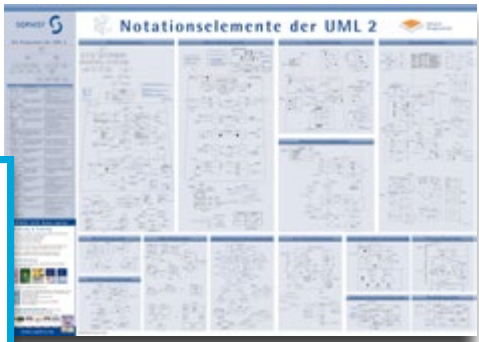
Wissensträger mal anders!

Kostenfrei!

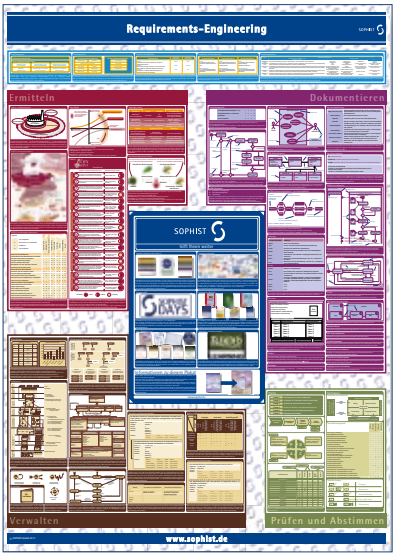
WISSEN for free

Poster/Plakate

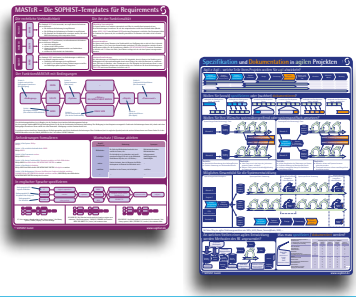
Das SOPHIST UML-Plakat



Das SOPHIST RE-Plakat ...



... und seine „Kerne“



WISSEN for free

Broschüren

www.sophist.de/wissen-for-free

SOPHIST

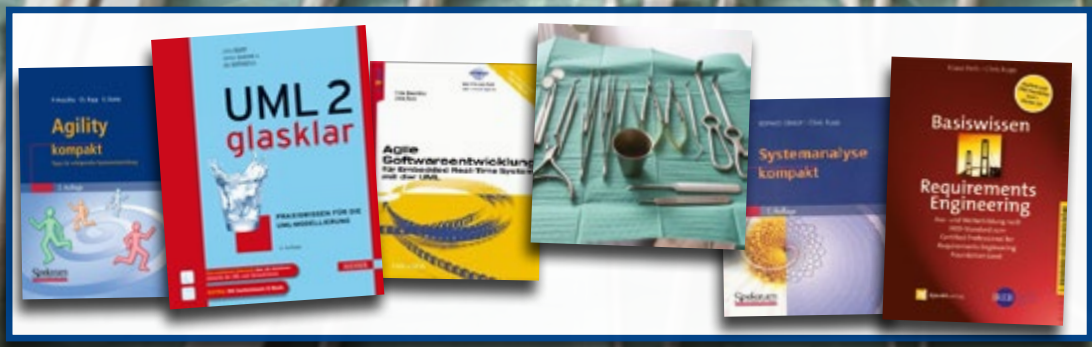
Kompetenz und Fachwissen

par excellence

Methodenerfinder

Speaker

Buchautoren



Berater Coach Trainer

Das bieten wir Ihnen an:

Wir unterstützen Sie kompetent, tatkräftig und zielführend sowohl bei der Anpassung Ihrer Entwicklungsprozesse und -methoden als auch bei der Durchführung Ihres Projekts.

Zu unseren Kunden gehören viele weltbekannte Unternehmen. Die Vielzahl an positiven Meinungen und Projektberichten unserer zufriedenen Kunden spricht für sich. Werfen Sie doch einen Blick auf **www.sophist.de/referenzen**

Unsere Leistungen:

- Verbesserungspotenziale unter Berücksichtigung der Randbedingungen in Ihrer Organisation identifizieren, ausschöpfen und einführen
- Anforderungen und Architekturen angemessen erheben, analysieren sowie vermitteln und dokumentieren
- Unterwegs in einfachen Software-Anwendungen bis hin zu komplexen Systemen
- agil und angepasst zu arbeiten

All das und noch viele weitere Themen aus der Welt des Requirements- und Systems-Engineerings bieten wir Ihnen in Form von Beratung, Coaching, Trainings und Vorträgen an.

Wie können wir Ihnen helfen?

Gerne arbeiten wir mit Ihnen ein Konzept aus, um Sie bestmöglich in Ihrem Vorhaben zu unterstützen.

Kontaktieren Sie uns unverbindlich:

+49 (0) 911 40 900 - 0

heureka@sophist.de

SOPHIST Trainings

UPDATE

Was ist daran neu?

Alle Trainings von SOPHIST sind nach neuestem Stand der Wissensvermittlung konzipiert. Unsere Trainer folgen unter anderem den Grundsätzen der Methode „... from the Back of the Room“, um Trainingsinhalte nachhaltig zu vermitteln.

Eine aktivierende Lernumgebung – mehr Bewegung, weniger Text, mehr Interaktion mit den Teilnehmern und überraschende Übungskonzepte – sorgt für Spaß und Effizienz beim Lernen.

Ihre Vorteile?

Sie profitieren nicht nur von dem Know-How der Methodenführer, sondern auch von einer didaktischen Umsetzung, die ihre Spuren hinterlässt.

Neugierig geworden?

Kontaktieren Sie uns unverbindlich:

+49 (0) 911 40 900 - 0

heureka@sophist.de

Online/Remote Trainings

Ihre SOPHIST Weiterbildung - an nahezu jedem Ort der Welt

SOPHIST Online/Remote Trainings sind speziell für die Vermittlung von Wissen und Können über das Internet entwickelt worden. Die besondere und sorgfältig durchdachte Ausarbeitung – inhaltlich und didaktisch – sowie eine Teilnehmerzahl von maximal 12 Personen gewährleistet einen perfekten Online-Wissenstransfer.

Für **Einzelpersonen** und **kleinere Teams** eignen sich unsere „Offenen Trainings“ perfekt. Und durch den modularen Aufbau, der Kombination verschiedener Schwerpunkte und der Möglichkeit der individuellen Anpassung, eignen sich Remote/Online Trainings auch perfekt für firmeninterne Weiterbildungen **kompletter Teams.**

Natürlich gibt es auch unsere bekannten CPRE-Zertifizierungstrainings in diesem Format.

... egal wo

Requirements Engineering in der Agilität

Die Disziplin des Requirements Engineering ist seit vielen Jahren das Kernthema der Firma SOPHIST. Waren die früheren Jahre des Requirements Engineering vor allem durch wasserfallartig abgewickelte Projekte geprägt, so wird heutzutage gerade in der Softwareentwicklung zunehmend agil entwickelt. Das wirkt sich auch auf die Techniken und Arbeitsweisen im Requirements Engineering aus, denn die Rahmenbedingungen sind hier völlig andere. Das heißt aber nicht, dass in agiler Entwicklung kein Requirements Engineering mehr gemacht wird, es sieht häufig nur anders aus und wird zu anderen Zeitpunkten durchgeführt als in den klassischen Vorgehen.

In der vorliegenden Broschüre beschäftigen wir uns mit den Bereichen der Agilität in denen es um das Ermitteln, Vermitteln und das Herleiten guter Anforderungen geht. Also den Haupttätigkeiten des Requirments Engineering. Schlagworte, welche sich in dieser Broschüre finden, sind:

- User Storys
- Product Backlog
- Priorisierung
- Refinement
- Schneiden von User Storys
- Roadmaps
- Skalierung von Agilität
- uvm

Bei all diesen Begriffen immer den Hauptaugenmerk auf unser Kernthema gerichtet:
Dem Requirements Engineering