Herausforderungen für das Qualitätsmanagement im agilen Umfeld

Projektbericht

Projektnummer	9810011
Projektdauer	01.01.2018 – 31.12.2018
Autor(en)	Gunnar Harde (Automotive Quality Institute GmbH)
Datum	21.12.2018
E-Mail	kontakt@aqigmbh.de

Diese Veröffentlichung basiert auf dem Expertenwissen aus dem AQI und wissenschaftlicher Partner. Sie stellt einen konsolidierten Standpunkt zum entsprechenden Themenkomplex dar.

Inhaltsverzeichnis

Vorwort 3

1. AGILITÄT 4

1.1. Begriffsklärungen 4

1.2. Das Wesen von Agilität 6

1.3. Mitarbeitermotivation 9

1.4. Der Gesamtprozess 10

1.5. Beauftragung agiler Softwareentwicklungen 12

1.6. Die Rolle des Product-Owners 17

1.7. DevOps & QM 18

1.8. Agilität jenseits der Softwareentwicklung 20

1.9. Agilität skaliert 23

1.10. Fazit 25

2. AGILITÄT UND SOFTWAREQUALITÄT 27

2.1. Qualität durch Agilität 27

2.2. Grundsätze des agilen Qualitätsmanagements 29

2.3. Agiles Testen im Entwicklungsteam 34

2.4. Q-Spezialisten 39

2.5. Qualitätsmanagement von außen 40

2.6. Exkurs: Dokumentation 44

2.7. Prozessqualität 47

2.8. Fazit 47

3. SOFTWAREENTWICKLUNG IN DER AUTOMOBILINDUSTRIE 49

3.1. Automotive SPICE 50

3.2. Funktionale Sicherheit (ISO 26262) 57

3.3. AUTOSAR 62

3.4. Reifegradabsicherung für Neuteile gemäß VDA 63

4. QM AGILER SW-ENTWICKLUNG IN DER AUTOMOBILINDUSTRIE 66

4.1. Q-Anforderungen definieren und einbringen 67

4.2. Lieferanten bewerten 75

4.3. Reifegrad absichern 76

4.4. Produkte freigeben 80

4.5. Feld beobachten und Lessons Learned ableiten 81

4.6. Agile Grundsätze leben 82

4.7. Fazit 84

ANHANG 86

A. Qualitätseigenschaften von Softwareprodukten gemäß ISO/IEC 25010 86

B. Basisszenario zur Vereinbarkeit von ISO 26262 und Agilität 90

C. Literaturverzeichnis 94

Vorwort

Die Automobilindustrie erlebt derzeit einen Wandel zu einer weiter zunehmenden Digitalisierung ihrer Produkte und Dienstleistungen. Dieser Wandel betrifft nicht nur die Art der Produkte, sondern auch die Art der Produktentwicklung und -wartung: Dominierten in der Automobilindustrie bislang phasenorientierte Produktentstehungsprozesse, so nehmen agile Ansätze, wie sie in der Softwareentwicklung seit vielen Jahren erfolgreich eingesetzt werden, an Bedeutung zu. Diese zunehmende Agilität, die sich beim Product-Lifecycle von Softwareprodukten in der Automobilindustrie abzeichnet, wird auch das bisherige Verständnis des Qualitätsmanagements in Frage stellen und neue Antworten seitens des Qualitätsmanagements bei der Umstellung auf agile Entwicklungsmethoden in der Automobilindustrie erfordern.

Der hier vorliegende Bericht des Projekts „Herausforderungen an das Qualitätsmanagement im agilen Umfeld“ befasst sich genau mit diesen Aufgaben, vor denen das Qualitätsmanagement steht. Der Bericht besteht aus vier Teilen:

Der erste Teil „Agilität“ gibt einen Überblick über das Thema Agilität. Im Fokus dieses Teils steht die agile Softwareentwicklung, da Agilität dort seinen Ursprung hat und etabliert ist. Viele der dort beschriebenen Grundsätze, Prinzipien und Methoden lassen sich jedoch auch auf Entwicklung jenseits der Softwareentwicklung sinnvoll anwenden. Die Möglichkeiten und Grenzen von Agilität sind ebenso beschrieben wie die grundlegenden Ideen zur Skalierung von Agilität für große Entwicklungsvorhaben.

Der zweite Teil „Agilität und Softwarequalität“ befasst sich mit dem Qualitätsmanagement bei der agilen Softwareentwicklung. Es wird dargestellt, welche Auswirkungen die agilen Werte, Grundsätze und Prinzipien auf die Qualität der Software haben und wie sich das Qualitätsmanagement durch den veränderten Entwicklungsansatz verändert. In der agilen Entwicklung gebräuchliche Techniken zur Test- und Auslieferungsautomatisierung erlauben heute sehr kurze Entwicklungs- und Auslieferungsiterationen und verändern dadurch das Verständnis und die Zusammenarbeit von Entwicklung, Betrieb und Qualitätsmanagement.

Im dritten Teil „Softwareentwicklung in der Automobilindustrie“ werden die in der Automobilindustrie für die Entwicklung von Software wichtigsten Standards betrachtet und hinsichtlich ihrer Vereinbarkeit mit Agilität untersucht. Zu den betrachteten Standards gehören Automotive SPICE, die ISO 26262, AUTOSAR und die Reifegradabsicherung für Neuteile gemäß VDA.

Der vierte Teil „QM agiler SW-Entwicklung in der Automobilindustrie“ gibt Handlungsempfehlungen für ein Qualitätsmanagement von agil entwickelten Produkten in der Automobilindustrie. Dieser Teil berücksichtigt die zuvor dargestellten Themen Agilität, Softwarequalität und branchenspezifischer Standards und beschreibt deren Auswirkungen auf das Qualitätsmanagement in der Automobilindustrie.

1. AGILITÄT

Die Entwicklung von Software nach dem agilen Ansatz hat sich seit der Formulierung des *Manifesto for Agile Software Development* immer weiter etablieren können und ist heute in vielen Unternehmen und Abteilungen, in denen Software entwickelt wird, nicht mehr wegzudenken. Inzwischen entdecken auch immer mehr Organisationen, die keine Software entwickeln, Agilität für sich: beim IT-Betrieb, bei der Entwicklung von Hardware, bei Reorganisationsprojekten bis hin zur agilen Führung ganzer Unternehmen.

Gerade aufgrund der derzeitigen großen Popularität von Agilität hat das Verständnis darüber, was unter Agilität zu verstehen ist, gelitten. Nicht selten sind Schlagworte aus der agilen Szene unreflektiert übernommen worden, ohne die Ideen dahinter wirklich verstanden zu haben. Dies tut meist weder der betroffenen Organisationen gut, noch dem agilen Ansatz, der durchaus seine Berechtigung hat – aber eben auch seine Grenzen. Nüchternheit und Klarheit sind angebracht.

Agilität kann in vielen Bereichen sinnvoll angewendet werden, jedoch nicht in allen. Dieser Teil des Projektberichts erklärt den Begriff *Agilität* und zeigt auf, welche Voraussetzungen für echte Agilität notwendig sind – für die Entwicklung von Software und gegebenenfalls darüber hinaus.

1.1. Begriffsklärungen

Agil – das klingt nach geistiger und körperlicher Beweglichkeit und Vitalität („Oma ist noch recht agil“). Jeder kennt umgangssprachlich den Begriff *agil*[1] und verbindet damit positive Assoziationen. Dies erschwert das Verständnis von *agil* bzw. *Agilität*, wie es im *Manifesto for Agile Software*

[1] Im englischen und teilweise auch im deutschen Sprachgebrauch ist es inzwischen üblich, den Ausdruck *Agile* zu verwenden. Dave Thomas, ein Mitautor des *Manifesto for Agile Software Development*, befürwortet stattdessen die Verwendung des Begriffs *Agility* (Thomas 2015). Dem schließt sich der Autor hier an.

Development (Beck et al. 2001a; im Folgenden vereinfacht als *Agiles Manifest* bezeichnet) gemeint ist: Ein selbstorganisiertes Experten-Team entwickelt Software in enger und kontinuierlicher Zusammenarbeit mit dem Auftraggeber inkrementell-iterativ[2] und ergebnisorientiert.

Das Agile Manifest beschreibt grundlegend die Art und Weise, wie Software aus Sicht der Autoren entwickelt werden sollte. Dass die Autoren den Begriff *agil* verwendet haben, ist plausibel, da sie ausdrücklich eine Abkehr von den bis dato üblichen, stark phasenorientierten und oft schwergewichtigen Softwareentwicklungsansätzen forderten. Änderungen auch noch in einer späten Entwicklungsphase heißen sie ausdrücklich willkommen. Eine solche eher organisatorische Beweglichkeit ist durchaus wesensverwandt mit dem umgangssprachlichen Gebrauch im Sinne von geistiger und körperlicher Beweglichkeit, hat aber im Gegensatz dazu, wie alle Managementansätze, Nebenwirkungen und ist nicht in jedem Kontext sinnvoll.

Zunächst ein paar Begriffsklärungen:

Das Agile Manifest definiert *agile Grundsätze*[3] (Beck et al. 2001a) und zwölf *agile Prinzipien* hinter diesen Grundsätzen (Beck et al. 2001b). Das Agile Manifest definiert damit, was unter Agilität für die Softwareentwicklung zu verstehen ist. Die Grundsätze lauten:

"We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more."

Ken Schwaber und Mike Beedle haben 2002 darüber hinaus *agile Werte* als Voraussetzung für den agilen Wandel und das agile Arbeiten identifiziert: Selbstverpflichtung, Fokus, Offenheit, Respekt und Mut (Schwaber/Beedle 2002). Diese Werte sind wichtige Voraussetzungen für Agilität, sind darüber hinaus aber auch relevant für viele andere betriebliche und außerbetriebliche Bereiche. Sie definieren daher nicht Agilität.

Agile Praktiken sind einzelne Best Practices, die oft bei agilen Entwicklungen genutzt werden. Hierzu zählen u. a. Pair-Programming, User-Stories zur Erfassung von Anforderungen, Planning

[2] Der Begriff „inkrementell-iterativ" soll zum Ausdruck bringen, dass Software sukzessiv um Softwaremodule erweitert (inkrementell) und bestehende Softwaremodule sukzessiv optimiert (iterativ) wird. Ebenso werden die Softwareentwicklungsprozesse und -methoden iterativ hinterfragt und verbessert.

[3] Die agilen Grundsätze werden im Deutschen oft als *agile Werte* bezeichnet. Dies ist irreführend. Grund für den Begriff im Deutschen dürfte die Übersetzung von "Through this work we have come to value" in „Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt" sein. Gemeint ist jedoch weniger ein Wertesystem als vielmehr wertschöpfender Nutzen.

Poker zur Aufwandsschätzung und Test-Driven-Development. Diese Praktiken sind nicht Teil des Agilen Manifests, helfen jedoch, einzelne Prinzipien zu verwirklichen.

Schließlich sind die *agilen Methoden* (auch: *agile Frameworks*) konkrete Anleitungen zur Umsetzung von Agilität. Dabei werden agile Praktiken gebündelt und aufeinander abgestimmt. Es gibt unterschiedliche, teilweise konkurrierende, teilweise sich ergänzende Methoden. Zu den agilen Methoden zählen u. a. Kanban, Scrum und Extreme Programming. Agile Methoden sind ebenso wie agile Praktiken nicht Teil des Agilen Manifests, beziehen sich aber mehr oder weniger explizit darauf. Agile Methoden und Praktiken sind somit konkrete Ausprägungen von Agilität, sind aber für ein agiles Arbeiten nicht zwingend erforderlich.

Der Maßstab zur Bewertung, ob eine Software agil entwickelt wird oder nicht, sind die Grundsätze und Prinzipien des Agilen Manifests. Dabei geht es weniger darum, die einzelnen Prinzipien buchstabengetreu umzusetzen, sondern vielmehr darum, dass alle agilen Grundsätze des Agilen Manifests im Wesentlichen gelebt werden. Dies bedeutet:

- Ein Team, bei dem die Führungskraft bestimmt, wie die Aufgaben zu erledigen sind, und diese an einzelne Mitarbeiter delegiert, arbeitet nicht agil.
- Ein Projekt, bei dem lange nach Projektbeginn Konzepte dem Kunden als Teilergebnisse präsentiert werden, ist nicht agil.
- Eine Beauftragung, die auf einer rechtlich verbindlichen und unveränderlichen Spezifikation basiert, kann nicht agil umgesetzt werden.

1.2. Das Wesen von Agilität

Agilität besteht im Kern aus zwei Teilen: das inkrementell-iterative Entwickeln der Software und die Selbstorganisation des Teams.

1.2.1. Inkrementell-iteratives Vorgehen

Das inkrementell-iterative Entwickeln von Software hat einen entscheidenden Vorteil gegenüber der klassischen phasenorientierten Softwareentwicklung: Der Auftraggeber kann anhand des bislang Erstellten sehr viel besser seine Anforderungen erkennen, konkretisieren und – solange er die finanziellen Mittel bereitstellt – in die Entwicklung einsteuern. Er muss sich nicht vorab ein Endprodukt im Detail vorstellen und vollständig spezifizieren, um am Ende doch feststellen zu müssen, dass es nicht den Erwartungen und dem tatsächlichen Bedarf entspricht. Je unklarer die fachlichen Anforderungen und technischen Bedingungen sind, umso größer der Vorteil durch ein

solches Vorgehen. Gerade bei komplexen Produkten erzielt das Team durch ein inkrementell-iteratives Herantasten oft sehr viel bessere Lösungen als durch eine aufwendige Vorabspezifikation. Daher die beiden Grundsätze des Agilen Manifests:

"**Responding to change** over following a plan"

"**Customer collaboration** over contract negotiation"

Statt Anforderungen vorab vollständig als Basis des Vertrags zu spezifizieren, soll der Kunde eng mit dem Auftragnehmer zusammenarbeiten und seine Anforderungen in den laufenden Entwicklungsprozess einfließen lassen.

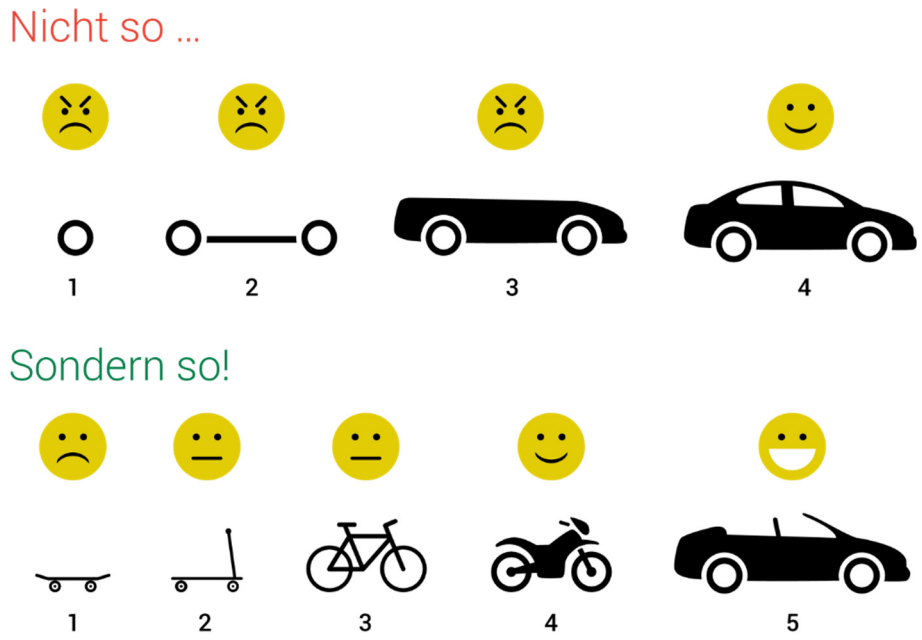

Abbildung 1: Inkrementelle agile Produktentwicklung.

Das inkrementell-iterative Vorgehen beschränkt sich nicht nur auf die Aufnahme von Anforderungen und deren Einsteuerung in die Entwicklung, sondern umfasst auch die Entwicklung selbst: Software wird in kurzen Iterationen getestet und integriert, Entwicklungsabläufe werden regelmäßig im Team bewertet, hinterfragt und verbessert.

Dave Thomas beschreibt Agilität daher wie folgt (Thomas 2015):

- "Find out where you are
- Take a small step towards your goal
- Adjust your understanding based on what you learned
- Repeat"

1.2.2. Selbstorganisation

Die klassische Projektorganisation ist vergleichbar mit dem Spielen von Tischfußball: Es gibt Angreifer, Mittelfeldspieler und Verteidiger, die an Stangen (= Rollenbeschreibung) befestigt sind, und den Spieler außerhalb des Spielfelds (= Projektleiter), der darüber entscheidet, wer sich wie bewegt und schießt. Dieser tayloristische Ansatz der Aufsplitterung der Tätigkeiten nach funktionalen Kriterien und deren Steuerung durch ein übergeordnetes Management soll durch Selbstorganisation des Teams überwunden werden: Es gibt weiterhin Angreifer, Mittelfeldspieler und Verteidiger, die sich aber frei auf dem Platz bewegen und gemeinsam als Team Verantwortung übernehmen. Selbstorganisation bedeutet, Fußball und nicht Tischfußball zu spielen.

Die Entwicklung von Software ist komplex. In einem komplexen Umfeld kann es keine allgemeingültigen Regeln geben. Regeln können nur kontextspezifisch sein. Diesen Kontext kennt das Entwicklungsteam sehr viel besser als das Management, das nur sporadisch mit der Entwicklung direkt zu tun hat.

Von außen vorgegebene Prozesse, einschließlich der dafür zu verwendenden Werkzeuge, grenzen den Gestaltungsspielraum des Teams ein und beschränken damit Selbstorganisation. Dem gegenüber betont das Agile Manifest die Bedeutung der Teammitglieder mit ihren Fähigkeiten und ihrer internen Zusammenarbeit:

> "**Individuals and interactions** over processes and tools"

Zudem soll der Erfolg eines Teams an das zu erzielende Ergebnis bemessen werden; und das Ergebnis ist nun mal die zu entwickelnde Software. Außer der Systemdokumentation für die spätere Pflege der Software und die Benutzerhandbücher (sofern erforderlich) sind alle anderen Dokumente Zwischenergebnisse. Liegt der Fokus auf dieser Dokumentation, stehen diese Zwischenergebnisse im Vordergrund und das eigentliche Ziel gerät zunehmend aus dem Blickfeld. Daher besagt das Agile Manifest:

> "**Working software** over comprehensive documentation"

Es sollte das Team sein, das darüber entscheidet, was es wie in welcher Tiefe dokumentieren will. Dies schließt auch ihre Verantwortung mit ein, die langfristige Wartbarkeit sicherzustellen.

1.2.3. Inkrementell-iteratives Vorgehen und Selbstorganisation

Wie hängen nun inkrementell-iteratives Vorgehen und Selbstorganisation zusammen? Ein Team kann sich auch dann selbst organisieren, wenn es seriell einen Plan abarbeitet, sich dem Ziel also nicht inkrementell-iterativ nähert. Eine inkrementell-iterative Entwicklung ist weder Voraussetzung noch Merkmal für Selbstorganisation.

Umgekehrt ist ein inkrementell-iteratives Vorgehen nur dann effizient, wenn sich das Team selbst organisiert. Das Team schärft die komplexe Aufgabe iterativ und entwickelt inkrementell die Soft-

ware weiter. Dazu sollte das Team sein gesamtes Wissen schnell und in enger Abstimmung einbringen können und nicht auf Entscheidungen von außen bzw. oben warten müssen. Genau dazu dient Selbstorganisation.

1.3. Mitarbeitermotivation

Der tayloristische Imperativ lautet: Zerlege einen Prozess in möglichst kleine Arbeitsschritte; weise diese Arbeitsschritte einzelnen spezialisierten, aber nicht unbedingt hochqualifizierten Mitarbeitern zu; erstelle detaillierte Anleitungen, wie die Arbeitsschritte durchzuführen sind; kontrolliere die Arbeit der Mitarbeiter. Das dazu passende Motivationsprinzip lautet: bestrafe Fehlverhalten und nicht eingehaltene Zusagen und belohne gute Leistungen mit Lob, Geld und Karriereaussichten.

Eine Reihe von psychologischen Studien haben nachgewiesen, dass eine solche Zuckerbrot-und-Peitsche-Methode tatsächlich funktioniert, vorausgesetzt, es handelt sich dabei um ausgesprochen einfache und routinemäßige Tätigkeiten. Die Studien zeigen aber auch, dass sobald eine Aufgabe ein bisschen Kreativität oder analytische Fähigkeit erfordert, dieser Versuch des Motivierens nicht funktioniert (ausführlich beschrieben in Pink 2009). Im Gegenteil: Es konnte gezeigt werden, dass bei anspruchsvollen Aufgaben die Leistungen eher sinkt, wenn Geld als Belohnung in Aussicht gestellt wird.

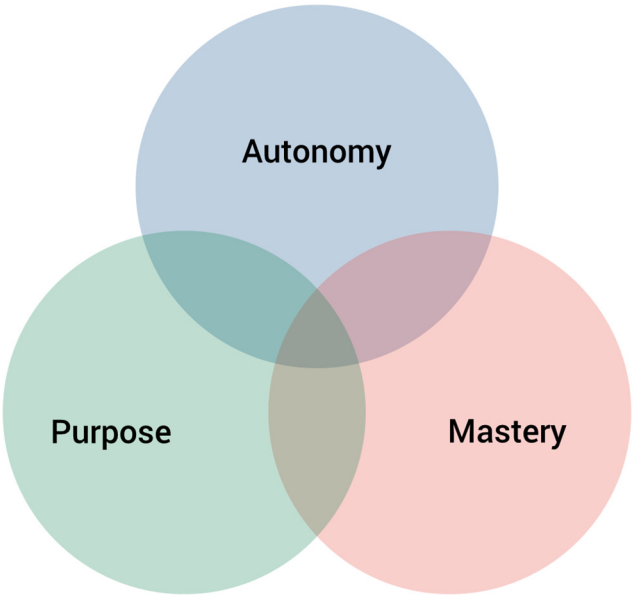

Abbildung 2: Faktoren der Motivation (Pink 2009).

Ausschlaggebend für die Motivation von Mitarbeitern sind andere Faktoren: **Autonomie** bei der Ausübung der Arbeit, **Meisterschaft** in dem ausgeübten Beruf und den **Sinn**, den Mitarbeiter in ihrer Arbeit sehen.

Die Entwicklung von Software ist eine komplexe und anspruchsvolle Aufgabe. Das Motivieren durch Loben, Tadel und monetäre Anreize funktioniert für Softwareentwicklung ebenso wenig wie für die meisten postindustriellen Tätigkeiten. Das Gehalt ist ein Hygienefaktor, keine Motivation.

Die agilen Grundsätze und Prinzipien unterstützen intrinsische Motivation sehr viel besser als traditionelle phasenorientierte Ansätze. Der Sinn, den Mitarbeiter in ihrer Tätigkeit sehen, hängt entscheidend von dem zu erstellenden Produkt ab und ist damit unabhängig vom Entwicklungsparadigma. Autonomie und Meisterschaft hingegen hängen vom Entwicklungsprozess ab.

Die Prinzipien des agilen Manifests fordern Autonomie und Meisterschaft explizit ein:

“Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.”

“Continuous attention to technical excellence and good design enhances agility.”

“The best architectures, requirements, and designs emerge from self-organizing teams.”

So gibt es in Scrum keinen Projektleiter, der Tätigkeiten delegiert und kontrolliert. Stattdessen gibt es einen Product-Owner, der eigenverantwortlich und unabhängig darüber entscheidet, was in welcher Reihenfolge umgesetzt werden soll, der aber nicht darüber entscheidet, wie etwas umzusetzen ist. Über das Wie entscheidet einzig und allein das selbstorganisierte, autonome Entwicklungsteam.

Das beim agilen Framework *Extreme Programming* übliche Pair-Programming, d. h. zwei Entwickler arbeiten gemeinsam vor einem Bildschirm und tauschen sich während des Programmierens kontinuierlich aus, fördert ein permanentes Lernen der Entwickler untereinander. Fachliche Meisterschaft wird so ausdrücklich gefördert.

1.4. Der Gesamtprozess

Das Agile Manifest beschreibt die Grundsätze und Prinzipien agiler Softwareentwicklung. Es bezieht sich damit auf die Entwicklung von Software, nicht auf deren Beauftragung und deren Auslieferung vor dem Kunden. Eine agile Softwareentwicklung eingebettet in eine Organisation, in der der Beauftragungs- bzw. Bewilligungsprozess und die Auslieferung der Software nach schwergewichtigen, langsamen Prozessen erfolgen, kann den Nutzen von Agilität jedoch oft nur sehr eingeschränkt entfalten: Was nützt es, wenn neue Anforderungen innerhalb weniger Tage

vom Product-Owner in den Entwicklungsprozess eingesteuert und diese kurzfristig umgesetzt werden könnten, jedoch die Bewilligung der dafür notwendigen Mittel und die Auslieferung der umgesetzten Anforderungen durch den IT-Betrieb Monate benötigen?

Dies bedeutet jedoch nicht, dass eine solche Agilität, die sich nur auf die Softwareentwicklung beschränkt, nutzlos sein muss. Soll beispielsweise eine betriebliche Software durch eine neu zu entwickelnde Software abgelöst werden, so kann das Team frühzeitig und regelmäßig noch lange vor der Auslieferung Feedback von den Stakeholdern aus den Fachabteilungen einholen und in die weitere Entwicklung einfließen lassen. Erst wenn die neue Software mehr kann als die alte, wird die neue Software ausgeliefert. Anders sieht es jedoch bei Software aus, die für den Massenmarkt vorgesehen ist. Dort ist eine regelmäßige Auswertung des tatsächlichen Nutzungsverhaltens und der mit der Software generierten Umsätze für die weitere Entwicklung der Software von entscheidender Bedeutung. Ein zu langwieriger Prozess zum Einspielen neuer Versionen blockiert eine effiziente Rückkopplung der Marktbeobachtungen in die Entwicklung und damit eine effektive Weiterentwicklung der Software.

Laut einer Forrester-Studie waren im Jahr 2011 die meisten agilen Entwicklungsteams in klassischen Wasserfall-Prozessen eingebunden (West 2011):

“Organizations are adopting Agile through a combination of bottom-up adoption and top-down change. But the reality of Agile adoption has diverged from the original ideas described in the Agile Manifesto, with many adoptions resembling what Forrester labels water-Scrum-fall. This model is not necessarily bad, but if application development professionals do not carefully consider and make the right decisions about where the lines fall between water-Scrum and Scrum-fall, they are unlikely to realize Agile’s benefits.”

Um möglichst das gesamte Potential von Agilität zu heben, sollte sowohl die Beauftragung bzw. die Bewilligung als auch die Auslieferung und damit die Zusammenarbeit mit dem IT-Betrieb agil erfolgen. Hierzu bieten sich folgenden Maßnahmen an:

- Der vertragliche Rahmen zwischen Auftraggeber und Auftragnehmern erlaubt ein einfaches und unkompliziertes Einsteuern neuer und Herausnehmen obsoleter Anforderungen auch noch während der Entwicklung (siehe Abschnitt 1.5).
- Das agile Prinzip „Working software is the primary measure of progress” (Beck et al. 2001b) sollte sich auf die ausgelieferte, funktionierende Software vor Kunden beziehen, nicht auf eine Software noch im Freigabeprozess. Ein Erfolg ist die ausgelieferte Software (*Outcome*), nicht die erreichten Meilensteine oder Dokumente (*Output*) auf dem Weg dorthin. Im Gegensatz zum klassischen Projektleiter richtet der Product-Owner im agilen Umfeld sein Augenmerk stärker nach außen auf die betrieblichen Ziele, die mit der Software erzielt werden sollen (siehe Abschnitt 1.6).

- Die starre organisatorische Grenze zwischen Entwicklung (*Development*) und IT-Betrieb (*Operation*) wird aufgeweicht und überwunden. Entwicklung und IT-Betrieb sind gemeinsam dafür verantwortlich, dass ein entwickeltes Software-Inkrement zügig und qualitätsgesichert bis zum Anwender gelangt (siehe Abschnitt 1.7).

1.5. Beauftragung agiler Softwareentwicklungen

Die Art und Weise, wie Software entwickelt wird, ob agil-inkrementell oder klassisch-phasenorientiert, sollte sich in der Zusammenarbeit zwischen Kunde und Auftragnehmer und dessen vertraglichen Ausgestaltung wiederspiegeln. Werkverträge, deren Grundlage eine Vorabspezifikation aller Anforderungen erfordern, eignen sich gut für phasenorientierte Entwicklungen, sind aber für ein agiles Entwickeln ungeeignet. Dienstleistungsvereinbarungen unterstützen agile Softwareentwicklung sehr viel besser.

1.5.1. Das Scheitern anforderungsbasierter Festpreisprojekte

Bei der klassischen phasenorientierten Softwareentwicklung wird grundsätzlich davon ausgegangen, dass der Kunde zu Beginn seine Anforderungen im Form eines Lastenheftes spezifiziert und der Auftragnehmer ein Pflichtenheft erstellt, in dem auf Basis des Lastenheftes die technischen Anforderungen spezifiziert sind. Diese beiden Dokumente sind insbesondere bei vielen Festpreisprojekten Grundlage der Zusammenarbeit und wichtige Vertragsbestandteile.

Es zeigt sich jedoch, dass viele Kunden gerade bei komplexen Produktentwicklungen nicht in der Lage sind, alle ihre Anforderungen zu erkennen, diese für eine solide Aufwandsschätzung präzise genug zu beschreiben und die Wichtigkeit ihrer Anforderungen im Vorfeld richtig einzuschätzen. Ein Indiz dafür: Laut einer Untersuchung wurden in vier untersuchten Softwareprodukten 45 % der Funktionen gar nicht und weitere 19 % kaum genutzt (Johnson 2002).

Die Vorstellung, dass der Kunde im Vorfeld eine Software spezifiziert und dann zu einem fest vereinbarten Preis die Software entsprechend seinem tatsächlichen Bedarf erhält, scheitert oft aus den folgenden Gründen:

- Selbst anscheinend einfache Anforderungen stellen sich als unzureichend spezifiziert heraus. Z. B. könnte die Anforderung „Ein Nutzer kann seine Telefonnummer in das System eingeben“ bedeuten, dass der Nutzer einfach etwas in ein Feld namens Telefonnummer eingeben kann und dass dieses Etwas im System gespeichert wird. Es kann aber auch bedeuten, dass das System sicherstellen muss, dass es sich tatsächlich um eine Telefonnummer handelt; dass das System die Nummer in ein Standardformat transformieren muss; dass das System prüfen muss, ob die Telefonnummer zum angegebenen Wohnort

passt; dass der Nutzer um erneute Eingabe seiner Telefonnummer gebeten werden muss, wenn die zuerst angegebene Nummer nicht gültig ist; dass der Nutzer ggf. bei wiederholter Fehleingabe vom System gesperrt wird etc. Abhängig von der tatsächlichen Anforderung kann der Aufwand für die Realisierung erheblich schwanken.

- Der Kunde übersieht Anforderungen. Dies passiert oft gerade bei fundamentalen Anforderungen, die vom Kunden so selbstverständlich sind, dass er sie nicht mehr wahrnimmt.
- Anforderungen sind oft unklar formuliert. So könnte die Anforderung „Bringe mir vom Supermarkt einen Liter Milch mit, und falls sie dort Eier haben, bringe mir sechs mit“ dazu führen kann, dass man sechs Liter Milch erhält – weil es dort Eier gab. Sprachliche Missverständnisse sind eine Quelle fehlinterpretierter Anforderungen.
- Während der Entwicklungslaufzeit können sich Rahmenbedingungen wie z. B. rechtliche Anforderungen oder Geschäftsprozesse ändern. Somit ändern sich ggf. auch die Anforderungen an das beauftragte Softwaresystem.
- Oftmals zeigen sich technische Herausforderungen gerade hinsichtlich der Qualitätsanforderungen (Performance, Security, Bedienbarkeit etc. – siehe dazu Abschnitt A im Anhang) erst im Laufe des Projektes. Diese Herausforderungen können beträchtlich sein.

Aufgrund dieser Defizite und Unvollständigkeiten der Anforderungen und technischen Risiken ist die Aufwandsschätzung von Software sehr unsicher. Eine Kostenschätzung basierend auf einem Lastenheft hat typischerweise eine Unsicherheit von -33 % nach unten und 50 % nach oben – und das selbst bei einem hochwertigen Lastenheft und einem erfahrenen Schätzer (McConnell 2006). In der Praxis ist die Unsicherheit oft größer.

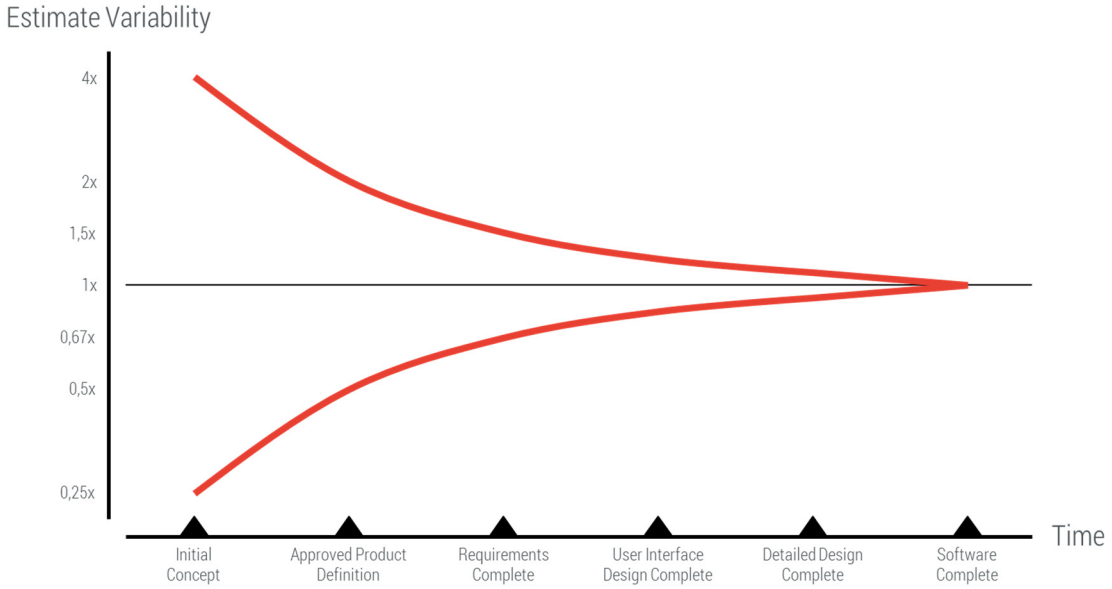

Abbildung 3: Idealisierter Konus der Unsicherheit (McConnell 2006).

Beim klassischen Projektmanagement werden nach der Zieldefinition und der Anforderungsspezifikation die Aufwände geschätzt. Die Spezifikation und die geschätzten Aufwände sind die Basis für den Projektplan. Selbst wenn zu Projektbeginn auf die Unsicherheit der Schätzwerte hingewiesen wird: Die initiale Aufwandsschätzung bleibt Referenz und das Maß zur Bewertung der Entwicklungsarbeit. Wurden die Aufwände zu Beginn unterschätzt, stehen Entwicklungsleiter und Entwicklungsteam unter Druck. Dies hat erhebliche Konsequenzen für die Kosten und die Qualität der Arbeit.

Es ist eine Sache, die Anforderungen mehr oder weniger erfolgreich umzusetzen und die Software zum Laufen zu bringen, aber es ist eine ganz andere Sache, Software von hoher Qualität zu schreiben. Gute Software funktioniert nicht nur irgendwie, sondern lässt sich nachhaltig und effizient weiterentwickeln. Es reicht nicht aus, eine Softwarearchitektur und ein Softwaredesign zu Beginn einmalig zu konzipieren, sondern diese muss auch tatsächlich umgesetzt und während der gesamten Entwicklungsphase hinterfragt und ggf. angepasst werden. Zudem gehören zu einer hochwertigen Software ein konsistenter Programmierstil, eine in sich logische Benennung von Klassen, Methoden und Variablen als auch eine angemessene Kommentierung des Quellcodes. Das Entwicklungsteam muss Zeit und Ressourcen investieren, um die Softwarequalität zu gewährleisten.

Sind die Aufwände bei einer phasenorientierten Entwicklung unterschätzt worden und setzt der Kunde bzw. die Entwicklungsleitung das Team unter Druck, so leidet in der Regel die Softwarequalität. Dieser Druck zeigt zunächst die gewünschte Wirkung: Die Software tut, was sie soll. Was nicht gesehen wird, sind die Qualitätsmängel „unter der Motorhaube“. Und was der Kunden zunächst noch nicht weiß: Jede weitere Entwicklung wird teurer und teurer, und er ist gefangen in einem Projekt, das kaum noch wartbar und an ein anderes Team kaum noch zu übergeben ist.

Hochwertige Software hat ihren Preis. Fehlt die Bereitschaft diesen Preis zu zahlen, gerade wenn zu Beginn aufgrund unterschätzter Aufwände ein zu niedriger Preis zugesagt wurde, so kann die Software am Ende noch sehr viel teurer werden.

1.5.2. Dienstleistungsvereinbarungen

Ein Arzt wird nicht dafür bezahlt, dass er einen Patienten heilt, sondern dafür, dass er es versucht. Dies ist das Charakteristische einer Dienstleistungsvereinbarung: Nicht der Aufwand zur Umsetzung des Anforderungsumfangs, sondern der Aufwand für die Leistungserbringung zählt. Durch diese Offenheit hinsichtlich der zu erbringenden Leistungen entfällt die Notwendigkeit, Anforderungen detailliert vorab zu spezifizieren und den Aufwand für deren Umsetzung zu kalkulieren, und somit entfallen auch die Unsicherheiten bei der Aufwandsschätzung und die Starrheit, die zu den Problemen bei Festpreisprojekten führen.

Doch Dienstleistungsvereinbarungen scheinen aus Sicht des Auftraggebers ein Fass ohne Boden zu sein. Er beauftragt den Auftragnehmer ohne zu wissen, was das Erreichen der Ziele am Ende kosten wird. Er weiß, dass es das Interesse des Auftragnehmers sein wird, möglichst viele fakturierbare Leistungen zu erbringen – ob für seine Ziele nun sinnvoll oder nicht –, und dass er nie

sicher sein kann, dass für die fakturierten Stunden auch Leistungen tatsächlich überhaupt erbracht worden sind.

Der Kontrollverlust des Auftraggebers bei Dienstleistungsvereinbarungen im Vergleich zu anforderungsbasierten Festpreisvereinbarungen existiert jedoch gar nicht: Der Auftraggeber hat die wirkliche Kontrolle bei komplexen Festpreisprojekten auch nur sehr bedingt.

Bei Dienstleistungsverträgen entfällt die Notwendigkeit des Auftraggebers, detaillierte Anforderungen für die Auftragsvergabe zu Projektbeginn zu erstellen. Dies bedeutet nicht, dass ein Anforderungsmanagement bei Dienstleistungsbeauftragungen nicht erforderlich wäre. Es bedeutet aber, dass das Anforderungsmanagement sich an das fachlich und nicht an das zur Vertragsgestaltung Notwendige orientiert. Es erlaubt zudem einen sehr viel flexibleren Umgang mit Änderungswünschen während der Projektlaufzeit, da vertragliche Änderungen, sei es auch nur in Form von Change Requests, nicht erforderlich sind. Das Anforderungsmanagement kann somit sehr viel schlanker und aus fachlicher Sicht effektiver gestaltet werden. Auf eine genaue Beschreibung der Projektziele sollte aber dennoch nicht verzichtet werden, da gerade wegen der rudimentär erfassten Anforderungen zu Projektbeginn diese Zieldefinitionen als Orientierung und Leitplanken während des Projektes dienen.

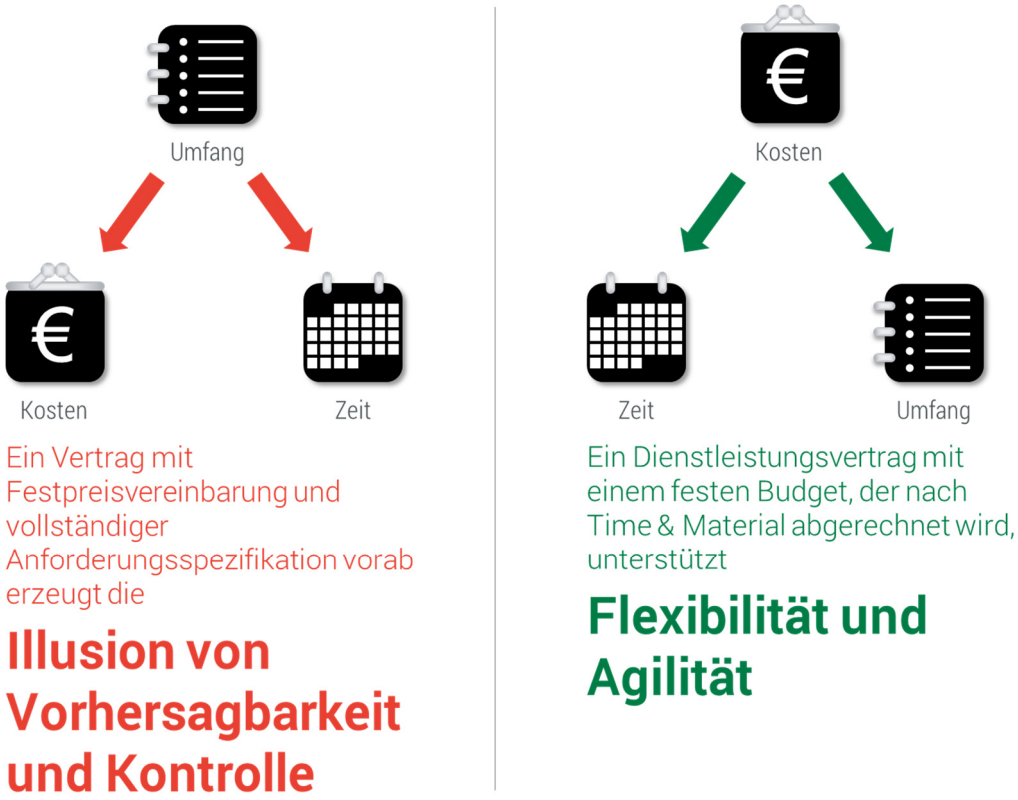

Abbildung 4: Drehung des magischen Dreiecks.

1.5.3. Agiler Festpreis

Bei klassischen, phasenorientierten Softwareentwicklungen werden aus den fest vorgegebenen Anforderungen Zeit und Kosten für deren Umsetzung abgeleitet. Ziel ist die möglichst genaue Ermittlung der Kosten und des Fertigstellungstermins, die sich jedoch aufgrund der großen Schätzunsicherheit in der Praxis meist nicht verlässlich voraussagen lassen.

Bei agilen Dienstleistungsvereinbarungen kann hingegen das Budget und die Zeit vorab fest vorgegeben werden, und der Leistungsumfang wird während der Entwicklungslaufzeit immer weiter konkretisiert. In diesem Fall spricht man von einem agilen Festpreis.

1.5.4. Agiler Werkvertrag

Unternehmensinterne Richtlinien können es erforderlich machen, dass Softwarebeauftragungen, sei es an interne oder externe Auftragnehmer, nur in Form von Werkverträgen vergeben werden dürfen. Das charakteristische Merkmal eines Werkvertrags ist die Abnahme des beauftragten Gewerks durch den Auftraggeber. Ein Auftraggeber kann nur das abnehmen, was zuvor spezifiziert worden ist. Somit scheint jede Form von Werkverträgen eine agile Umsetzung zu verhindern.

Eine Möglichkeit, auch im Rahmen von Werkverträgen eine agile Softwareentwicklung zu ermöglichen, besteht darin, die Anforderungen jeweils für kurze, aufeinanderfolgende Entwicklungsiterationen, sogenannte *Sprints*, festzulegen. Der Werkvertrag selbst enthält dabei weder die Einzelanforderungen noch die Abnahmekriterien, sondern beschreibt lediglich den Abnahmeprozess und legt die Abnahme nach jedem Sprint verbindlich fest. Damit sind die Voraussetzungen eines Werkvertrags erfüllt.

1.5.5. Exkurs: Arbeitnehmerüberlassung

Die Entwicklung von Software erfolgt oftmals durch oder mit Unterstützung externer Partner. Hierbei muss das Thema Arbeitnehmerüberlassung berücksichtigt werden. Sofern formal keine Arbeitnehmerüberlassung vorliegt, muss das auftraggebende Unternehmen darauf achten, dass das Einsteuern von Anforderungen in die Entwicklung ausschließlich über die vertraglich vereinbarten ANÜ-konformen Kommunikationskanäle erfolgt. Dies ist im Einzelfall zu prüfen und hängt von der Zusammensetzung des Teams, dem agilen Framework und dem Vertragsverhältnis ab.

Falls beispielsweise nach Scrum gearbeitet wird und das Entwicklungsteam komplett von einem externen Dienstleister gestellt wird, so könnten Auftraggeber und Auftragnehmer vereinbaren, dass bei der Sprint-Planung I, d. h. bei der Festlegung, was das Entwicklungsteam in dem anstehenden Sprint umsetzen soll, ein Vertreter des Dienstleisters, der gemäß der vertraglichen Vereinbarung Aufträge entgegennehmen darf, anwesend ist und den Umfang bestätigt. Formal verein-

baren somit nicht das Entwicklungsteam und der Product-Owner den Umfang des Sprints, sondern der benannte Vertreter des Dienstleisters und der Product-Owner – mit fachlicher Beratung durch das Entwicklungsteam. Während eines Sprints darf der Product-Owner keine neuen Anforderungen einsteuern, zumindest nicht ohne Einschaltung des Dienstleister-Vertreters. Dies entspricht jedoch ohnehin den Scrum-Regeln. Der Product-Owner muss dem Entwicklungsteam als Ansprechpartner bei fachlichen Fragen zur Verfügung stehen. Sollten unternehmensinterne Anweisungen auch dies nicht erlauben, so ist ein agiles Arbeiten nicht möglich.

Falls ein Entwicklungsteam sowohl aus externen als auch internen Mitarbeitern besteht, dürfte eine agile Entwicklung ausgeschlossen sein, da dadurch Aufgaben nicht mehr von dem Entwicklungsteam gemeinsam entgegengenommen und verantwortet werden könnten.

1.6. Die Rolle des Product-Owners

Der klassische Projektleiter hat die Aufgabe, vorab definierte Anforderungen umzusetzen und dabei den vorgegebenen Zeit- und Kostenrahmen einzuhalten. Sein Blick ist primär auf die internen Kosten und das effiziente und termingerechte Erreichen der Meilensteine gerichtet.

Anders der Product-Owner, wie er von einigen agilen Frameworks, z. B. Scrum, vorgesehen ist. Sein Blick ist weniger nach innen als vielmehr nach außen auf den zu erwarteten Nutzen der auszuliefernden Produktinkremente gerichtet. Er ist nicht dazu verpflichtet, eine Software nach Spezifikation gemäß Kostenrahmen und Meilensteinplan umzusetzen, sondern entscheidet eigenständig und eigenverantwortlich, wann welche Anforderungen in die Entwicklung eingesteuert werden. Hierbei gilt es, die übergeordneten wirtschaftlichen Ziele mit den verfügbaren Mitteln zu erreichen.

Diese ihm zugesprochene eigenständige Entscheidungskompetenz kann gerade in traditionellen Organisationen, in denen die Stakeholder gewohnt sind, ihre Anforderungen zu Entwicklungsbeginn einzufordern, zu Problemen führen. Nicht selten üben Stakeholder auf dem Weg der Eskalation Druck auf den Product-Owner aus und verlangen die Umsetzung der „eigenen“ Anforderungen. Dieser Druck verleitet Product-Owner oftmals dazu, einzelnen Stakeholdern Zusagen über Umfang und Zeitpunkt der Umsetzung ihrer Anforderungen zu machen. Dadurch wird jedoch der agile Ansatz untergraben: Stakeholder erwarten die Einhaltung der gemachten Zusagen, was eine umfassendere Planung erfordert und den Product-Owner in seiner Entscheidungskompetenz erheblich einschränkt.

Besser ist es, wenn der Product-Owner die verschiedenen Anforderungen koordiniert und mit den Stakeholdern abstimmt. Der Product-Owner nimmt so die Rolle eines Moderators ein, der zwischen den verschiedenen Interessen vermittelt. Hierzu stehen dem Product-Owner eine Reihe von

Methoden, wie z. B. das Planning-Poker[4], zur Verfügung. Ziel ist es dabei immer, zusammen mit den Stakeholdern den zu erwartenden Mehrwert durch die Umsetzung der einzelnen Anforderungen zu ermitteln und so eine wirtschaftlich sinnvolle Umsetzungsreihenfolge festzulegen. Nachteilig hierbei ist jedoch, dass dazu eine häufige und ggf. sehr zeitaufwändige Abstimmung der Stakeholder-Interessen notwendig ist, um eine ausreichende Agilität zu gewährleisten. Zudem leidet oftmals die Qualität der Software und die Innovationsfähigkeit: Die so ermittelten und abgestimmten Anforderungen sind meistens funktionaler, selten nicht-funktionaler Natur; die Gruppe tendiert dazu, sich auf den kleinsten gemeinsamen Nenner zu verständigen und Innovationen zu scheuen.

Idealerweise sollte ein Product-Owner die moderierende Rolle nur bei der Anforderungserfassung einnehmen. Die Entscheidung, welche Anforderungen wann in die Entwicklung eingesteuert werden, sollte einzig und allein ihm obliegen. Dies bedeutet jedoch nicht, dass er grundsätzlich niemandem Rechenschaft abzulegen hat: Ein Product-Owner sollte an den übergeordneten wirtschaftlichen Zielen gemessen werden, die er mit den ihm zur Verfügung gestellten Mitteln und innerhalb des Systemkontextes seines verantworteten Produkts zu erzielen hat. Das Festlegen sinnvoller Key-Performance-Indikatoren, die den wirtschaftlichen Erfolg des Produktes am besten widerspiegeln, ist die beste Grundlage für die effektive Arbeit des Product-Owners. Dadurch wird am besten sichergestellt, dass der Product-Owner sich an den wirtschaftlichen *Outcome* orientiert, nicht an den *Output* irgendwelcher Zwischenergebnisse, die für den Nutzer nicht unmittelbar relevant sind.

1.7. DevOps & QM

Auch wenn das Agile Manifest sich explizit auf die Softwareentwicklung bezieht, so kann sich Agilität erst dann richtig entfalten, wenn der Product-Owner die entwickelten Inkremente nicht nur zeitnah entwickeln, sondern auch zeitnah ausliefern lassen kann. Dies setzt voraus, dass sich die Softwarekomponenten zu einem Gesamtsystem effizient integrieren lassen und die Qualität der Software vor der Auslieferung weitgehend automatisiert getestet ist. Die klassische organisatorische Trennung von Entwicklung, Qualitätsmanagement und IT-Betrieb mit ihren definierten Übergabepunkten wird diesen Anforderungen in einem agilen Umfeld nicht mehr gerecht. Auf-

[4] Das Planning-Poker ist eine Methode zur Schätzung im Expertenteam. Jedes Teammitglied hat dazu eine Satz Karten mit jeweils möglichen Schätzwerten. Zunächst schätzt jedes Mitglied für sich alleine und legt die dazugehörige Karte verdeckt ab. Dann legen alle ihre Karten und damit ihre Schätzungen offen. Bei unterschiedlichen Schätzungen diskutieren die beiden Mitglieder, deren Werte am weitesten auseinander liegen. Dann findet eine neue Schätzrunde statt. Dies wird sooft wiederholt, bis es einen Konsens gibt. Planning-Poker wird im agilen Umfeld oftmals zur Aufwandsschätzung verwendet. Es lässt sich auch beim Schätzen des Nutzens von Anforderungen verwenden.

grund der hohen Dynamik bei der agilen Entwicklung müssen Entwicklung, Qualitätsmanagement, IT-Betrieb und alle anderen, an einem Rollout beteiligten Organisationseinheiten enger zusammenwachsen. Aus diesem Gedanken heraus ist *DevOps* entstanden.

Das Wort *DevOps* setzt sich aus den Wörtern *Development* und *Operation* zusammen und ist laut Bass et al. (2015):

"[…] a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality."

DevOps befasst sich mit dem organisatorischen Zusammenwachsen der IT-Bereiche *Entwicklung*, *Qualitätsmanagement* und *Betrieb* und den Werkzeugen und Prozessen, die ein effizientes und qualitätsgesichertes Integrieren (*Continuous Integration*) und Ausliefern (*Continuous Delivery*) der Software unterstützen. Ähnlich wie der Begriff *Agilität* wird der Begriff *DevOps* uneinheitlich verwendet. Insbesondere die Abgrenzung zu *Continuous Delivery* ist unscharf. Tendenziell steht beim *Continuous Delivery* primär der Prozess und die dazu unterstützenden Werkzeuge, die für ein effizientes Ausliefern erforderlich sind, im Fokus, während DevOps darüber hinaus den organisatorischen Aspekt mit einbezieht. Gemeinsam haben sie jedoch das Ziel, schnell dem Endanwender ein qualitätsgeprüftes Produkt zur Verfügung zu stellen – so wie von Bass et al. formuliert.

Dieses Ziel kann nur erreicht werden, wenn es einen durchgängigen, hochgradig automatisierten Prozess gibt, angefangen von automatisierten Tests der Softwareeinheiten und -komponenten, über ein automatisiertes Integrieren des Systems und Regressionstests, bis hin zur automatisierten Softwareauslieferung inkl. abschließender Tests auf der Produktivumgebung mit anschließender Freischaltung. Dieser Prozess kann nicht erst bei der Übernahme der Software durch den IT-Betrieb einsetzen, noch kann der IT-Betrieb oder das Qualitätsmanagement diesen Prozess parallel und unabhängig von der Entwicklung allein verantworten. Vielmehr müssen Entwicklung, Qualitätsmanagement, IT-Betrieb und alle anderen beteiligten Bereiche Hand in Hand arbeiten.

Während es bei der klassischen IT-Organisation einen definierten Übergabepunkt gibt, ab dem der IT-Betrieb die erstellte Software von der Entwicklung übernimmt und dann ausliefert, so stellt der IT-Betrieb nun die gesamte Werkzeugpalette bereit, die ein möglichst automatisiertes Ausliefern ermöglicht. Die Vorstellung eines „fertigen" Produkts, das nach der Prüfung durch eine Qualitätssicherung und der anschließenden Auslieferung nur noch vom IT-Betrieb gewartet werden muss, ergibt in einem agilen Umfeld keinen Sinn. Entwicklung, Wartung und Betrieb erfolgen vielmehr parallel und in enger organisatorischer Abstimmung. Maßstab für die organisatorische Aufgabenteilung sind nicht die funktionalen Einheiten „Entwicklung", „Qualitätssicherung" und „IT-Betrieb", sondern allenfalls Produktmodule, für die interdisziplinäre Teams zuständig sind. Dies stellt viele klassisch aufgestellten Unternehmen von großen Herausforderungen, gerade auch in der Automobilindustrie.

Die neue Rolle des Qualitätsmanagements durch DevOps und das Testen von Software im agilen Umfeld wird in Teil 2 ausgiebig behandelt werden.

1.8. Agilität jenseits der Softwareentwicklung

Der Titel des Agilen Manifests lautet: „Manifesto for Agile Software Development“. Es bezieht sich ausdrücklich auf die Entwicklung von Software. Anders bei Scrum. Im Scrum Guide heißt es: „Scrum has been used to develop software, hardware, embedded software, networks of interacting function, autonomous vehicles, schools, government, marketing, managing the operation of organizations and almost everything we use in our daily lives, as individuals and societies.“ (Schwaber, Sutherland 2017).

Für welche Art von Vorhaben ist ein agiles Vorgehen sinnvoll? Welche Voraussetzungen müssen dafür erfüllt sein?

1.8.1. Inkrementell-iterativ vs. phasenorientiert

Agilität ist dann sinnvoll, wenn das Ziel unscharf und die Aufgabenstellung komplex ist. Eine inkrementell-iterative Entwicklung ist dann der beste Ansatz, da so das zunächst unklare Ziel kontinuierlich geschärft und am effektivsten erreicht wird. Dazu muss jedoch eine Voraussetzung erfüllt sein: Der Aufwand für Änderungen des bereits Erstellten muss angemessen sein. Falls beispielsweise der Auftraggeber in einer späten Entwicklungsphase möchte, dass der Nutzer die Farben der Software konfigurieren kann, so dürfte diese ursprünglich nicht eingeplante Anforderung einige Umbauten erfordern. Die Änderungskosten sind der Preis für ein inkrementelles Vorgehen und müssen im Vergleich zu dem Nutzen wirtschaftlich vertretbar sein. Dies ist für die meisten Anforderungen an eine Software der Fall.

Demgegenüber sind die Änderungskosten bei vielen Hardwareentwicklungen sehr viel höher und rechtfertigen daher nicht immer ein inkrementelles Vorgehen. Beispielsweise sind Produktionswerkzeuge oft teuer und müssten bei Änderungen aufwendig angepasst werden. So sollte die B-Säule bei der Produktentstehung im Fahrzeugbau besser nicht immer wieder verschoben werden. Dennoch gibt es auch Nicht-Software-Produkte, die sich inkrementell entwickeln lassen. So ermöglicht beispielsweise der Einzug von 3D-Druckern ein kostengünstiges Anpassen von Bauteilen und ermöglicht so Agilität. Ebenso gibt es Softwareentwicklungen, die besser phasenorientiert durchgeführt werden. So gibt es Softwareentwicklungen sowohl mit klarer Zielstellung (z. B. die 1:1-Übertragung einer existierenden iPhone-App auf Android-Geräte) als auch mit hohen Änderungskosten (z. B. Änderungen grundlegender Java-Schnittstellen, die von der gesamten Java-Community verwendet werden). Dann kann ein phasenorientiertes Vorgehen angebracht sein.

Generell gelten folgende Kriterien:

Der inkrementell-iterative Ansatz ist dann in Betracht zu ziehen, wenn das Ziel unscharf ist (siehe u. a. Stacey 2000; Abbildung 5). Sind die Ziele hingegen klar und kann eine Abteilung diese durch

klar beschriebene Prozesse erreichen, so ist Agilität nicht sinnvoll. Die Produktion von Massenwaren oder die routinemäßige Sachbearbeitung sind daher für ein agiles Vorgehen eher ungeeignet.

Es muss immer abgewogen werden: Lässt sich das Ziel und die Zielerreichung tatsächlich vorab genau genug und wirtschaftlich ermitteln? Machen die Kosten für nachträgliche Änderungen eine Detailplanung zu Beginn notwendig? Bei vielen Produktentwicklungen kann dies nicht klar mit ja oder nein beantwortet werden. Hier gilt es abzuwägen und ggf. einige Teilentwicklungen phasenorientiert, andere agil umzusetzen.

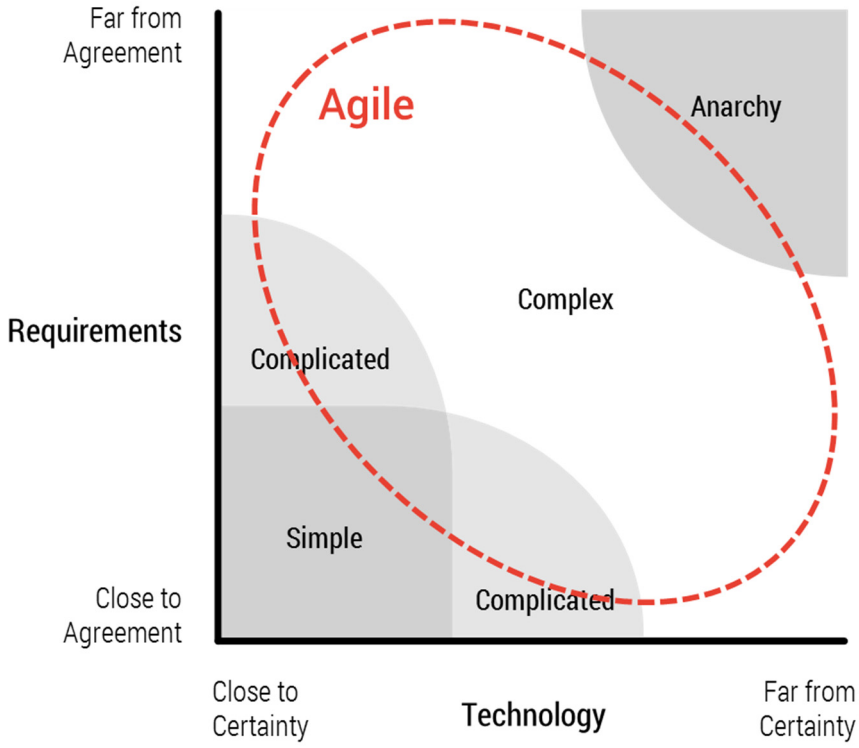

Abbildung 5: Stacey-Matrix.

1.8.2. Selbstorganisiert vs. anweisungsorientiert

Selbstorganisation bedeutet, dass das Entwicklungsteam selbst darüber entscheidet, wie es die Anforderungen umsetzt, und der Vertreter des Auftraggebers (der Product-Owner in der Scrum-Terminologie) als Teil des Teams darüber entscheidet, welche Anforderungen im Einklang mit den übergeordneten Geschäftszielen umgesetzt werden. Selbstorganisation ermöglicht es dem Team, bei einer inkrementell-iterativen Umsetzung einer komplexen Aufgabe schnell und unkompliziert Entscheidungen zu treffen. Selbstorganisation ist Voraussetzung für Agilität. Selbstorganisation setzt wiederum voraus, dass die Teammitglieder Verantwortung übernehmen wollen, dass sie die Qualifikation mitbringen, um Verantwortung übernehmen zu können und dass ihnen der

Freiraum dazu gegeben wird. Demgegenüber wird Agilität behindert oder gar verhindert, wenn eine der folgenden Situationen vorliegt:

- Echtes Know-how-Gefälle: Liegt das für die Entwicklung notwendige Wissen ausschließlich beim Vorgesetzten oder Projektleiter und haben die Teammitglieder nicht die Qualifikation, um die Entwicklung eigenverantwortlich durchzuführen, so kann Selbstorganisation nicht gelingen. Ein agiles Team (ggf. temporär erweitert um unterstützende Experten) muss das fachliche Know-how haben, um selbst Verantwortung für die Qualität des Produktes übernehmen zu können.

- Gefühltes Know-how-Gefälle: Das Wissensgefälle zwischen Führungskraft (Vorgesetzter oder Product-Owner) und den anderen Teammitgliedern wird oftmals von der Führungskraft nur so wahrgenommen und ist nicht real. Auch dies ist jedoch ein echtes Hindernis für Selbstorganisation. Eine Führungskraft, die sich für unentbehrlich hält und Mikromanagement betreibt, wird eigenverantwortliches Arbeiten seiner Mitarbeiter nicht zulassen.

- Vermeidung von Verantwortung: Selbstorganisation bedeutet Übernahme von Verantwortung durch das Team im Rahmen der gestellten Aufgabe. Hierzu muss das Team bereit sein und akzeptieren, dass das bequeme Mittel der Delegation von Verantwortung nach oben („Eskalation“) oft nicht mehr zur Verfügung steht. Ebenso muss jeder akzeptieren, dass das Ergebnis des Teams bewertet wird, nicht der individuelle Beitrag des Einzelnen. Ist das Team dazu grundsätzlich nicht bereit, wird Selbstorganisation nicht funktionieren.

- Restriktive Rahmenbedingungen: Gesetzliche oder organisatorische Restriktionen können Selbstorganisation stark behindern und u. U. verhindern. Beispielsweise gibt es in der Automobilindustrie aufgrund von Rechtsprechungen zunehmend die Tendenz, die zyklomatische Komplexität[5] von Software als ein Qualitätskriterium verbindlich vorzuschreiben. Hier soll nicht bestritten werden, dass die zyklomatische Komplexität ein Indikator für die Wartbarkeit von Software sein kann, aber weder ist eine geringe zyklomatische Komplexität Garantie für gute Wartbarkeit noch muss eine hohe zyklomatische Komplexität im Einzelfall zwingend schlecht sein. Solche Vorschriften bewirken eher, dass das Team Regeln befriedigt statt selbst Qualität zu verantworten.

- Zusammenspiel vieler Gewerke: Gerade bei komplexen und damit schwer zu planenden Entwicklungen ist Agilität ein probater Ansatz. Schwierig wird eine agile Entwicklung jedoch dann, wenn viele unterschiedliche Teams unterschiedlicher Gewerke zum Gesamtprodukt beitragen sollen. Selbstorganisation ist nur bis zu einer gewissen Teamgröße praktikabel (typischerweise bis zu neun Mitgliedern). Eine Aufteilung der Teams ist auch in der agilen Softwareentwicklung gängige Praxis und wird erfolgreich gelebt (siehe Abschnitt 1.9). Können oder wollen einige der Teams jedoch aus den oben genannten Grün-

[5] Die zyklomatische Komplexität ist eine Softwaremetrik zur Messung der Komplexität eines Softwaremoduls. Sie ist definiert als die Anzahl linear unabhängiger Pfade auf dem Kontrollflussgraphen eines Moduls.

den nicht agil arbeiten, muss im Einzelfall geprüft werden, ob Agilität für die Gesamtentwicklung noch funktionieren kann und sinnvoll ist. Die Möglichkeit, in den Einzelteams agil zu arbeiten, sollte hierdurch grundsätzlich nicht beeinträchtigt werden.

1.9. Agilität skaliert

Agilität kann nur von selbstorganisierten, interdisziplinären Teams gelebt werden, deren Mitglieder sich den Werten Selbstverpflichtung, Mut, Fokus, Offenheit und Respekt verpflichtet fühlen. Interne Hierarchien stehen dem entgegen. Agile Teams lassen sich daher nicht beliebig skalieren. Gemäß Scrum darf das Entwicklungsteam eine Größe von neun Mitgliedern nicht überschreiten.

Große Entwicklungsprojekte müssen jedoch von mehr als einem einzigen Team gestemmt werden. Derzeit werden verschiedene Modelle zur Organisation mehrerer agiler Teams zum Teil sehr kontrovers diskutiert.

Das Fundament der Skalierung von Agilität sind agile Teams. Arbeiten mehrere agile Teams an einem Produkt, so sollten die Teams so zusammengestellt sein, dass ihre Eigenständigkeit soweit wie möglich erhalten bleibt. Der Abstimmungsaufwand zwischen den Teams sollte minimiert werden. Dies gelingt typischerweise am besten, indem Feature-Teams gebildet werden, die für Funktionsbereiche verantwortlich sind. Teams, die für einzelne Technologien spezialisiert sind (z. B. GUI oder Datenbank-Entwicklung), sind im agilen Umfeld hingegen unüblich und meist nicht zielführend.

Im Extremfall kann sich herausstellen, dass sich die Feature-Teams nicht untereinander abstimmen müssen. In diesem Fall gibt es tatsächlich nicht ein Produkt, sondern mehrere Produkte, das jeweils von einem Team autonom entwickelt wird. Eine Koordination der Teams ist damit nicht notwendig und wäre Verschwendung.

Ist eine Entkopplung der agilen Teams jedoch nicht möglich, so bieten sich grundsätzlich zwei Skalierungsmuster an (siehe hierzu auch Foegen und Kaczmarek 2016). Die im Folgenden vorgestellten Muster gehen von Scrum-Teams als agile Teams aus. Eine Übertragung auf andere agile Frameworks ist grundsätzlich möglich.

1.9.1. Eng gekoppelte Teams

Bei diesem Muster gibt es einen Product-Owner und ein Product-Backlog, dessen Einträge von mehreren Entwicklungsteams umgesetzt werden. Die Sprints der Teams sind synchronisiert und beginnen mit einer gemeinsamen Sprint-Planung I aller Entwicklungsteams, bei der die Product-Backlog-Einträge den Entwicklungsteams zugewiesen werden. Anschließend führt jedes Entwicklungsteam seine eigene Sprint-Planung II durch. Vertreter der Entwicklungsteams stimmen während eines Sprints ihre Aktivitäten täglich in sogenannten Scrum-of-Scrum-Meetings untereinander ab. Die Entwicklungsteams stellen beim Sprint-Review ihre Ergebnisse dem Product-Owner gemeinsam vor. Den Abschluss eines Sprints bilden teaminterne Retrospektiven gefolgt von einer gemeinsamen Retrospektive mit allen Entwicklungsteams.

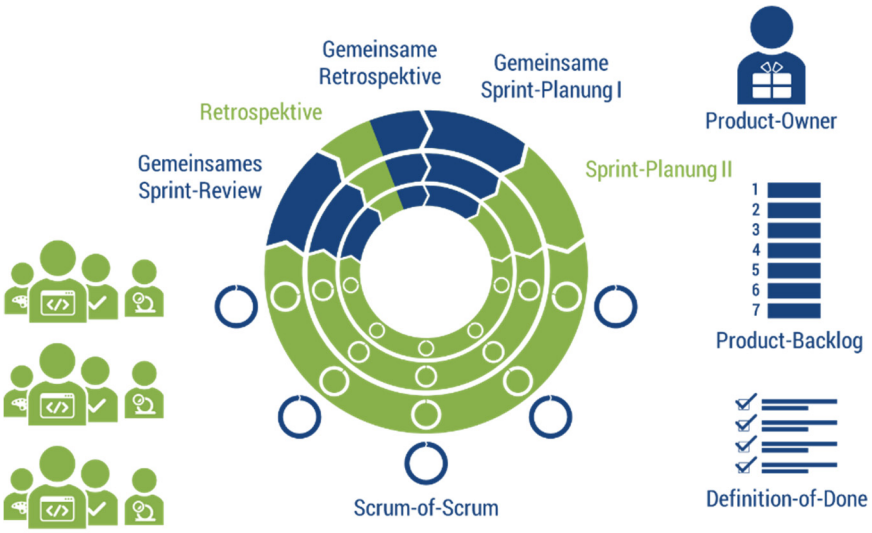

Abbildung 6: Muster für eng gekoppelte agile Teams.

Dieses Muster setzt voraus, dass ein einzelner Product-Owner in der Lage ist, mehrere Entwicklungsteams mit ausreichend Product-Backlog-Einträgen zu versorgen. Da ein Product-Owner Arbeiten ans Entwicklungsteam delegieren kann, dürfte dies bis zu einer Anzahl von vier Entwicklungsteams möglich sein. Das Framework *Large-Scale Scrum* (LeSS), das auf diesem Muster basiert, geht sogar davon aus, dass so bis zu acht Entwicklungsteams auf diese Weise gemeinsam an einem Produkt arbeiten können (Larman und Vodde 2018).

1.9.2. Lose gekoppelte Teams

Im Gegensatz zu eng gekoppelten Teams hat jedes Team einen eigenen Product-Owner und bestimmt selbst, wann welche Ereignisse wie Sprint-Planung oder Retrospektive stattfinden. Die Teams sind somit autonomer; Teamvertreter stimmen sich lediglich während der Sprints in Form von Scrum-of-Scrum ab; die Ergebnisse aller Teams werden bei den gemeinsamen Sprint-Reviews vorgestellt und diskutiert. Dafür gibt es eine zusätzliche Ebene: Alle 1-3 Monate planen und bewerten die Teams ihre Arbeiten zusammen mit einem Chief-Product-Owner und einem Einheit-Agility-Master. Es gibt somit eine Hierarchie sowohl bei den Product-Ownern als auch bei den ScrumMastern. Der Chief-Product-Owner verantwortet auf oberer Ebene das Product-Backlog, das dann bei Etappenplanungen in Etappen-Backlogs für die einzelnen Team-Product-Owner heruntergebrochen wird.

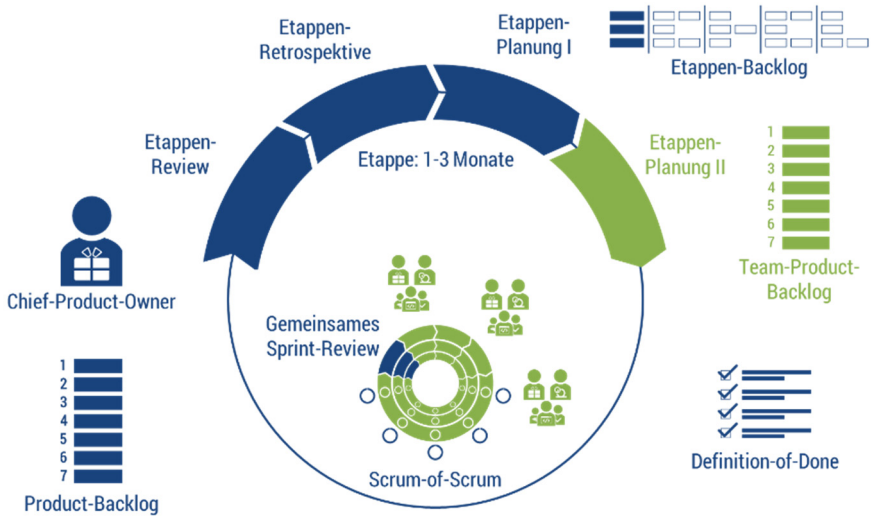

Abbildung 7: Muster für lose gekoppelte agile Teams.

Durch die Hierarchisierung des Anforderungsmanagements entsteht eine planerische Sicherheit und damit eine Einschränkung der Autonomie der einzelnen agilen Teams. Daher muss ernsthaft hinterfragt werden, ob eine so arbeitende Organisation selbst noch agil ist. Gerade das bekannte, auf diesem Muster basierende *Scales Agile Framework* (SAFe) steht in der Kritik, nicht den agilen Grundsätzen und Werten zu entsprechen (Schwaber 2013).

1.10. Fazit

Agilität ist ein sinnvoller Ansatz, damit Entwicklungsteams Software effektiv entwickeln können. Viele Unternehmen haben in den vergangenen Jahren Software erfolgreich agil entwickelt und gegenüber dem klassischen, phasenorientierten Vorgehen sowohl Kosten gespart als auch die

Qualität ihrer Produkte verbessert. Agilität ist in vielen IT-Unternehmen heute nicht mehr wegzudenken.

Der Nutzen agiler Softwareentwicklung entfaltet sich erst dann vollständig, wenn der Gesamtprozess, angefangen von der Beauftragung der Entwicklung bis hin zum Ausliefern der Software, Agilität unterstützt. Dies bedeutet: Ein interdisziplinäres Team konkretisiert eigenverantwortlich und selbstorganisiert Anforderungen iterativ, entwickelt Software inkrementell, liefert Inkremente kontinuierlich aus und optimiert die agilen Praktiken iterativ.

Agilität kann auch jenseits der Softwareentwicklung angebracht sein, gerade bei komplexen Produktentwicklungen und anderen herausfordernden Tätigkeiten. Voraussetzung hierfür ist jedoch, dass der Nutzen von Agilität die Aufwände für den Umbau bestehender Inkremente wirtschaftlich rechtfertigt.

Die Entwicklung umfangreicher Produkte erfordert die Koordination mehrerer agiler Teams. Derzeit existieren unterschiedliche Ansätze zur Skalierung agiler Entwicklungen. Bislang hat sich kein Framework als Standard etablieren können. Voraussichtlich wird es auch in Zukunft aufgrund unterschiedlicher Rahmenbedingungen kein allgemein gültiges Skalierungsmodell als Best Practice durchsetzen können. Aktuell wird diskutiert, inwieweit Skalierungsmodelle die agilen Werte und Grundsätze unterstützen.

2. AGILITÄT UND SOFTWAREQUALITÄT

Ziel der Softwareentwicklung ist die Umsetzung von Anforderungen an eine Software. Dazu zählen neben den funktionalen Anforderungen und Randbedingungen auch Anforderungen an die Qualität. Dies gilt für agil genauso wie für klassisch entwickelte Software. Im Folgenden wird dargelegt, welche Auswirkungen ein Wechsel hin zu einem agilen Ansatz auf die Softwarequalität hat und wie die Qualität im agilen Umfeld sichergestellt werden kann. Es zeigt sich, dass Agilität nicht nur ein Paradigmenwechsel für die eigentliche Entwicklung, sondern auch für das Qualitätsmanagement bedeutet.

Agile Methoden werden inzwischen auch jenseits der Softwareentwicklung angewendet. Dennoch soll dieser Teil explizit das Qualitätsmanagement bei agiler Softwareentwicklung zum Thema haben, da die grundsätzliche Anwendbarkeit des agilen Ansatzes bei der Entwicklung von Software unstrittig und agile Softwareentwicklung am weitesten ausgereift und verbreitet ist.

Softwarequalität umfasst eine Reihe von Qualitätseigenschaften. Die ISO/IEC 25010 beschreibt eine Systematik für Qualitätseigenschaften von Softwareprodukten (ISO/IEC 2010). Die acht Qualitätseigenschaften und insgesamt 31 Untereigenschaften sind im Anhang gelistet. Sie dienen im Folgenden als Referenz, um den Bezug der Qualitätsmaßnahmen zur Softwarequalität zu verdeutlichen.

2.1. Qualität durch Agilität

Im Agilen Manifest heißt es: „Working software over comprehensive documentation.“ Dies suggeriert, dass es einem agilen Entwicklungsteam primär um die Umsetzung der geforderten Funktionen geht; Qualitätseigenschaften, wie z. B. die Wartbarkeit der Software durch eine saubere Systemdokumentation, scheinen eine geringere Bedeutung zu haben. Selbstorganisation und somit das Fehlen verbindlicher Vorgaben durch das Management hinsichtlich einzuhaltender Qualitätsmaßstäbe verstärken zudem den Eindruck, dass mit keiner hohen Qualität der Software zu rechnen ist. Weder das Agile Manifest noch Frameworks wie beispielsweise Scrum kennen explizite QM-Rollen, noch fordern sie ein Vier-Augen-Prinzip zur Sicherung der Qualität. Vielmehr

hängt die Qualität der Software im Wesentlichen von der Qualität des agilen Teams ab: vom Product-Owner, der Anforderungen – auch hinsichtlich der Qualität und rechtlicher Randbedingungen – in die Entwicklung einsteuert, und vom Entwicklungsteam, das deren Umsetzung eigenständig verantwortet.

Für die Stakeholder außerhalb des agilen Teams bedeutet dies einen Kontrollverlust. Zwar werden auch im agilen Umfeld Qualitätsanforderungen vorgegeben und validiert (siehe dazu Abschnitt 2.5), die Qualität der Umsetzung verantwortet jedoch das Entwicklungsteam. Der Verzicht auf ein Qualitätsmanagement, das von außen in die Art und Weise, wie das agile Team Software entwickelt, eingreift, muss keinesfalls eine Verschlechterung der Softwarequalität zur Folge haben. Allein die Einführung von Agilität kann sich, wie im Folgenden erläutert, positiv auf die Qualität auswirken.

2.1.1. Motivation im agilen Team

Agilität fördert Autonomie und Meisterschaft des agilen Entwicklungsteams und somit die Motivation des Teams und jedes einzelnen Teammitglieds (s. Abschnitt 1.3). Ein motiviertes Entwicklungsteam hat den eigenen Anspruch, Produkte von hoher Qualität zu entwickeln. Die Qualität der Umsetzung wird nicht von außen eingefordert, sondern vom Team selbst gewollt. Dies ist eine wichtige Voraussetzung, um Software von hoher Qualität zu entwickeln – egal hinsichtlich welcher Qualitätseigenschaft.

2.1.2. Frühes Nutzerfeedback

Durch eine inkrementelle Entwicklung kann Nutzerfeedback frühzeitig und wiederholt erfasst werden und in die weitere Entwicklung einfließen. Unnötige und damit qualitätsmindernde Anforderungen werden seltener umgesetzt. Die Qualität aus Nutzersicht wird kontinuierlich erhöht. Eine phasenorientierte Entwicklung realisiert die Anforderungen, die zu Beginn als relevant erachtet wurden. Eine agile Entwicklung realisiert eher die Anforderungen, die tatsächlich relevant sind. Die Qualitätseigenschaften *funktionale Vollständigkeit* und *funktionale Angemessenheit* werden bei einem agilen Ansatz daher meist sehr viel besser umgesetzt als beim klassischen phasenorientierten Ansatz.

2.1.3. Angemessene Zeit für Umsetzung der Qualitätsansprüche

Klassisches Projektmanagement geht davon aus, dass basierend auf einer vollständigen Spezifikation Dauer und Kosten der Softwareentwicklung ermittelt werden können. Ist der tatsächliche Aufwand jedoch größer als angenommen, so steht das Entwicklungsteam unter Druck. Dieser Druck geht dann oft zulasten der Qualität.

Betroffen ist besonders die *Wartbarkeit* von Software, da diese für den Auftraggeber kaum zu verifizieren ist. Auch alle anderen Qualitätseigenschaften, die von außen nicht unmittelbar verifizierbar sind, wie z. B. *Sicherheit* oder *Portierbarkeit*, können unter dem Termindruck leiden.

Bei einer agilen Softwareentwicklung gibt es keine vollständige Spezifikation vorab. Die Umsetzungsgeschwindigkeit zeigt sich im Laufe der ersten Iterationen. Das Entwicklungsteam kann die Softwarequalität konstant hochhalten. Vielmehr werden bei Bedarf die Mittel für die Entwicklung oder der Funktionsumfang der Software angepasst.

2.1.4. Continuous Integration, Continuous Delivery

Bei klassischen Softwareentwicklungsprojekten werden die zuvor entwickelten Softwaremodule oft erst spät und kurz vor der geplanten Auslieferung zu einer Gesamtlösung integriert, und die Software wird weitaus seltener ausgeliefert als bei einer agilen Entwicklung. Dies führt oft zu erheblichen Schwierigkeiten bei der Integration: Die zuvor monatelang erstellten Module passen nicht wie erwartet zusammen und müssen unter hohem Zeitdruck überarbeitet werden. Agile Entwicklungsteams hingegen integrieren Software sehr viel häufiger. Kontinuierliches Refactoring, Integrieren und Ausliefern verbessert kontinuierlich die *Wartbarkeit* und *Installierbarkeit* der Software.

2.2. Grundsätze des agilen Qualitätsmanagements

Agile Softwareentwicklung erfordert ebenso wie klassische Softwareentwicklung das Managen der Qualität des Produktes. Dieses agile Qualitätsmanagement folgt denselben agilen Grundsätzen, Prinzipien und Werten wie die agile Softwareentwicklung selbst. Die Art, Softwarequalität zu managen, unterscheidet sich somit im agilen Umfeld grundlegend von der Qualitätssicherung der phasenorientierten Entwicklung, die oft als eigenständige Organisationseinheit erst nach der Entwicklung die Qualität des Produktes prüft.

Für ein agiles Qualitätsmanagement gelten hingegen folgende Grundsätze:

2.2.1. Das agile Team verantwortet Qualität

Ein agiles Prinzip lautet:

“The best architectures, requirements, and designs emerge from self-organizing teams.”

(Beck et al. 2001b)

Bei Scrum heißt es:

"Scrum Teams are self-organizing and cross-functional. Self-organizing teams choose how best to accomplish their work, rather than being directed by others outside the team. Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team."

(Schwaber, Sutherland 2017)

Das agile Team verantwortet das entwickelte Produkt einschließlich der Qualität. Innerhalb des Teams muss jedoch zwischen fachlichen Qualitätsanforderungen und der technischen Umsetzungsqualität unterschieden werden: Es obliegt dem Product-Owner, die fachlich notwendigen und sinnvollen Qualitätsanforderungen in die Entwicklung einzusteuern, und dem Entwicklungsteam, Anforderungen mit der erforderlichen Qualität zu realisieren. Beispielsweise könnte der Product-Owner Anforderungen hinsichtlich des Datenschutzes formulieren. Wie diese fachlichen Anforderungen technisch umgesetzt werden, liegt jedoch allein in der Verantwortung des Entwicklungsteams. Denkbar wäre auch, dass z. B. eine Abteilung für funktionale Sicherheit Sicherheitsanforderungen an das Produkt gegenüber dem Product-Owner einfordert. Auch hier gilt: Vor der Auslieferung prüft der Product-Owner bzw. die Abteilung für funktionale Sicherheit die Umsetzung der fachlichen Anforderungen, nicht die Art der technischen Umsetzung.

Daher muss unterschieden werden zwischen dem Testen innerhalb des Entwicklungsteams – ggf. mit beratender Unterstützung von außen – und einem Qualitätsmanagement außerhalb des Teams, das Qualitätsanforderungen auf rein fachlich-geschäftlicher Ebene stellt und validiert.

Dieser Grundsatz ist von entscheidender Bedeutung für Agilität:

- Würde Verantwortung für die Umsetzungsqualität oder Teilaspekte der Umsetzungsqualität aus dem Team ausgelagert werden, so würde die organisatorische Trennung von Entwicklung und Qualitätsmanagement den Gesamtprozess verlangsamen.
- Ein ausgelagertes Management der Umsetzungsqualität wäre in der Regel für mehrere agile Teams zuständig. Die Qualitätsrichtlinien würden meist für alle Teams gelten und die speziellen Gegebenheiten in den einzelnen Teams wären nur unzureichend berücksichtigt. Nur das Entwicklungsteam, das täglich an der Entwicklung arbeitet, weiß um alle Details, um angemessen die Besonderheiten beurteilen zu können.
- Ein externes Qualitätsmanagement, das nur einzelne Qualitätseigenschaften verantwortet, wird diese Qualitätseigenschaften tendenziell überbetonen. Es besteht die Gefahr, einen Perfektionismus schon zu Beginn der Entwicklung einzufordern, der dem Gedanken eines möglichst kurzem Time-to-Market entgegensteht.

2.2.2. Qualität entsteht während der Entwicklung

Bei einer phasenorientierten Softwareentwicklung werden das Entwickeln und das Testen von Software als zwei voneinander separate Tätigkeiten betrachtet, wie z. B. beim V-Modell, ein in der klassischen Softwarenentwicklung gängiges Vorgehensmodell. Beispielsweise sind in dem V-Modell-basierten Automotive SPICE *SWE.3: Software Detailed Design and Unit Construction* und *SWE.4: Software Unit Verification* als zwei separate Prozesse definiert (A-SPICE 2017).

In der Praxis führt ein phasenorientiertes Vorgehen in der Regel dazu, dass Software auf Komponentenebene zunächst wenig, je näher die Auslieferung rückt, jedoch immer intensiver getestet wird. Die Qualitätssicherung im klassischen Umfeld fokussiert sich auf Verifizierung der fertig entwickelten Software (anhand der Spezifikation) und die Freigabe für den Rollout.

Allerdings forderte z. B. W. Edwards Deming bereits 1986, Qualität nicht erst nachgelagert zu betrachten:

“Inspection does not improve the quality, nor guarantee quality. Inspection is too late. The quality, good or bad, is already in the product. As Harold F. Dodge said, ‘You can not inspect quality into a product.’”

(Deming 1986, S. 29)

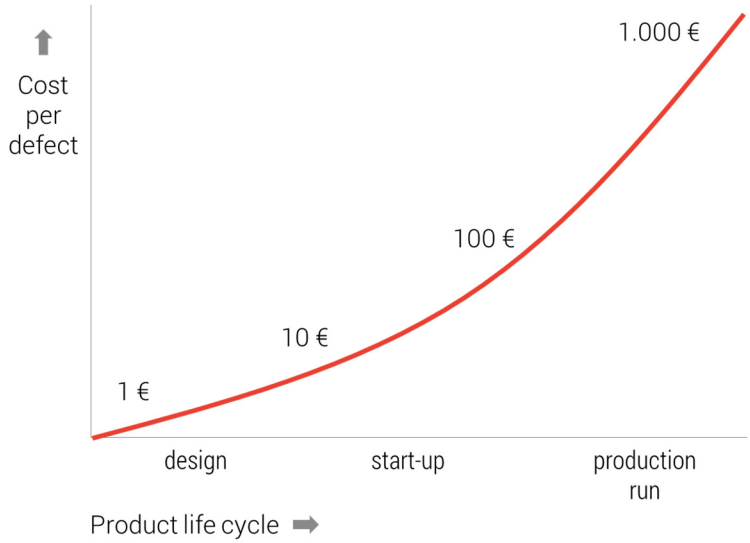

Abbildung 8: Rule of Ten - Bei jedem Schritt in der Wertschöpfungskette lässt sich ein Fehler ungefähr zehnmal schneller und günstiger beheben als beim Folgeschritt.

Bei einer agilen Softwareentwicklung wird die Qualität so früh wie möglich berücksichtigt und – soweit möglich – noch während der Entwicklung überprüft, um den Aufwand für das Beheben von Fehlern möglichst gering zu halten (s. *Rule of Ten*, Abbildung 8). Der Product-Owner legt (ggf. mit Unterstützung des Entwicklungsteams oder Stakeholdern) noch vor einer Iteration Ak-

zeptanzkriterien fest, die als Maßstab für die erfolgreiche Umsetzung dienen. Während einer Iteration programmieren die Entwickler typischerweise noch vor der Komponentenentwicklung Unit-Tests, die dann für das automatisierte Testen verwendet werden. Dieses *Test-Driven-Development* (TDD) trägt dazu bei, dass Anforderungen vor der Umsetzung zwischen Product-Owner und Entwicklungsteam hinreichend geklärt werden und die Entwickler sich auf die Realisierung der in den Unit-Tests festgelegten Anforderungen fokussieren. Da die Unit-Tests in der gleichen Programmiersprache wie die zu entwickelnde Software geschrieben und üblicherweise in die Entwicklungsumgebung eingebunden sind, gehen Testprogrammierung, Softwareprogrammierung und automatisiertes Testen Hand in Hand. Der Entwickler führt diese Schritte in Zyklen von teilweise nur wenigen Minuten eigenständig durch. Softwareentwicklung und Testen auf Unit-Ebene verschmelzen zu einem Arbeitsschritt.

Unit-Tests sind das Herzstück und Fundament agilen Qualitätsmanagements, insbesondere der funktionalen Verifizierung. Weitere Tests, wie z. B. Funktionstests des integrierten Systems oder GUI-Tests, finden auch im agilen Umfeld statt, normalerweise jedoch in einem geringeren Umfang als bei einer klassischen Entwicklung. Unit-Tests sind einfach zu implementieren, robust und performant. Je fachlicher und weniger technisch eine Testmethode ist und somit näher realen Nutzungsbedingungen entspricht, desto instabiler und langsamer sind die Tests und desto aufwendiger ihre Umsetzung.

Grundsätzlich sollten die Testumfänge auf den jeweiligen Integrationsschichten wir eine Pyramide aufgebaut sein (s. Abbildung 9). Auf welche Art von Tests die Schwerpunkte gelegt werden, hängt jedoch auch stark von der Art des Produktes und den Anforderungen ab. Zudem sollten die Tests auf den unterschiedlichen Integrationsstufen untereinander abgestimmt sein. Z. B. sollten Tests auf Unit-Ebene auf UI-Ebene nicht einfach wiederholt werden, sondern auf UI-Ebene gerade explorative Tests unter veränderten Nutzungsbedingungen durchgeführt werden, die so auf Unit-Ebene nicht möglich sind.

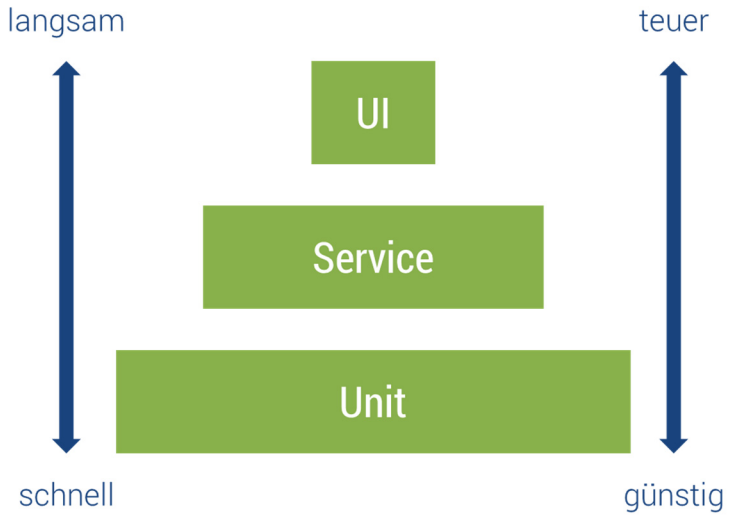

Abbildung 9: Testpyramide mit den schnellen und günstigen Unit-Tests als Fundament.

Verschiedene Testverfahren und ihre Bedeutung im agilen Umfeld sind im Abschnitt 2.3 noch ausführlicher beschrieben.

2.2.3. Kein Micromanagement

Der Product-Owner ist Teil des agilen Teams, aber nicht Teil des Entwicklungsteams. Das Entwicklungsteam ist autonom, und der Product-Owner ist kein Entwicklungsleiter, der dem Entwicklungsteam vorgibt, wie es die Software entwickeln soll. Weder der Product-Owner und erst recht niemand außerhalb des agilen Teams steuern das Entwicklungsteam durch Micromanagement.

Der Product-Owner identifiziert und priorisiert Anforderungen gemäß geschäftlicher Ziele. Er fokussiert sich auf die Validierung der auslieferbaren bzw. ausgelieferten Software. Nicht auslieferbare Artefakte, wie z. B. Lastenhefte, Spezifikationen oder Konzepte, sind weder Maßstab für Qualität noch Objekte des Qualitätsmanagements. Der Product-Owner misst und bewertet hingegen kontinuierlich den geschäftlichen Erfolg des ausgelieferten Produktes und leitet daraus fachliche funktionale und nicht-funktionale Anforderungen ab.

Im Abschnitt 2.5 ist näher beschrieben, wie Qualität durch den Product-Owner und anderen Stakeholdern eingefordert und geprüft werden kann, ohne die Autonomie des Entwicklungsteams zu beschneiden.

2.2.4. Zusammenfassung: Aufgaben des Qualitätsmanagements

Qualitätsmanagement im agilen Umfeld gibt es in zwei verschiedenen Ausprägungen: eines innerhalb des agilen, autonomen Entwicklungsteams, das die Umsetzung der Anforderungen verifiziert; eines außerhalb des Entwicklungsteams, das den Blick nach außen auf die Nutzung des Produktes durch die Anwender hat und die fachliche Qualität validiert.

Das Qualitätsmanagement im Entwicklungsteam wird im folgenden Abschnitt ausführlich behandelt.

Darüber hinaus kann es sinnvoll sein, Spezialisten zu bestimmten Qualitätseigenschaften temporär oder beratend in die Arbeit des Entwicklungsteams einzubinden. Diese Q-Experten-Pools sind im Abschnitt 2.4 näher beschrieben.

Der Abschnitt 2.5 beschreibt, wie Product-Owner und Stakeholder Qualitätsanforderungen stellen und die Qualität der Software prüfen, ohne die Autonomie des Entwicklungsteams zu untergraben.

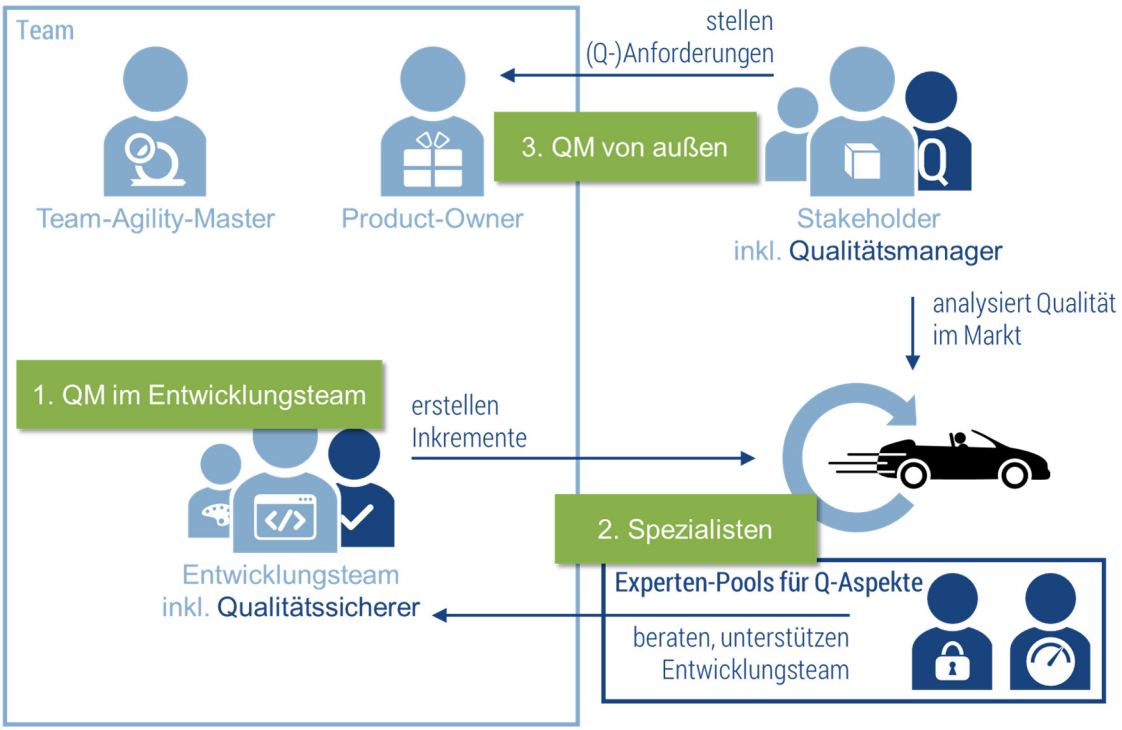

Abbildung 10: Übersicht über Aufgabenfelder des Qualitätsmanagements im agilen Umfeld.

2.3. Agiles Testen im Entwicklungsteam

2.3.1. Anforderungen umsetzen und testen

Die Umsetzung einer Anforderung durch ein agiles Team beginnt damit, dass der Product-Owner die Anforderung – im Scrum-Kontext das sog. *Product-Backlog-Item* (PBI) – beschreibt. Typischerweise wird die Anforderung als *User-Story* in der Form „Als <Rolle> möchte ich <Ziel/Wunsch>, um <Nutzen>“ formuliert.

Bevor ein Product-Owner ein PBI in die Entwicklung einsteuert, sollte er, ggf. mit Unterstützung des Entwicklungsteams oder Stakeholdern, Akzeptanzkriterien definieren. Diese dienen sowohl der Klärung der Anforderung als auch der späteren Abnahme. Akzeptanzkriterien sind in den meisten Fällen funktionale Testfälle, können aber auch Qualitätseigenschaften wie z. B. Zeitverhalten oder Usability-Eigenschaften betreffen. Die Akzeptanzkriterien haben primär den Zweck, die Anforderung genauer zu klären. User-Story, Akzeptanzkriterien und die Diskussion hierüber mit den Stakeholdern und im Team ersetzen die detaillierte Anforderungsspezifikation, wie sie bei der klassischen Softwareentwicklung vorgesehen ist.

Die eigentliche Umsetzung der Anforderung erfolgt durch das Entwicklungsteam. Im Sinne eines Test-Driven-Developments (TDD) kann das Entwicklungsteam bereits vor der Implementierung fachliche funktionale Testfälle, z. B. auf Basis der Akzeptanzkriterien, festlegen und zur Testautomatisierung in die Testwerkzeuge aufnehmen. Zudem können Prototypen, die z. B. einfach mit Stift und Papier erstellt werden, zum Einsatz kommen. Auch Prototypen dienen primär der Auftragsklärung.

Die Implementierung eines Moduls erfolgt durch einen Programmierer. Der Programmierer implementiert die Testfälle für dieses Modul als Unit-Tests, implementiert die ersten Funktionen des Moduls, testet das Modul, implementiert weitere Funktionen, testet erneut usw. Dies geschieht in kurzen Iterationen, oftmals von Minuten. Der Programmierer muss zum Testen nur die in derselben Entwicklungsumgebung wie das Modul implementierten Unit-Tests anstoßen, die dann automatisiert das Modul testen. Unit-Tests sind die kleinste Testeinheit. Sie sind vergleichsweise günstig, robust und schnell zu implementieren.

Vom agilen Framework *Extreme Programming* stammt das sog. *Pair-Programming* (Wells 1999), das inzwischen teilweise auch bei anderen agilen Frameworks verwendet wird. Beim Pair-Programming sitzen zwei Mitglieder des Entwicklungsteams – meist zwei Programmierer – vor einem Rechner und entwickeln gemeinsam ein Modul. Hierdurch soll der Wissensaustausch zwischen den Teammitgliedern gefördert und die Qualität der Software durch gemeinsame Inspektion verbessert werden. Statt des zweiten Programmierers kann auch ein Qualitätssicherer bei dem Finden und Formulieren von Unit-Testfällen unterstützen.

Neben den Unit-Tests stehen dem Entwickler Werkzeuge zur statischen Code-Analyse zur Verfügung. Diese Analysewerkzeuge untersuchen den Quellcode automatisiert nach möglichen Programmierfehlern wie z. B. Speicherlacks oder Performance-Problemen.

Der Programmierer checkt das fertige Modul ins Repository ein, in dem der gesamte Quellcode versioniert und verwaltet wird. Damit kann das Modul in die Gesamtlösung integriert und die Integrationstests gestartet werden. Die Integrationstests bestehen in der Regel aus Funktionstests ohne Einbindung der Benutzeroberfläche, damit sie möglichst performant und robust ablaufen. Im agilen Umfeld wird nicht wie bei der phasenorientierten Entwicklung erst vor einem Releasetermin integriert, sondern sehr viel häufiger, meist mindestens einmal pro Tag.

Die integrierte Software kann dann auf unterschiedliche Arten weiter getestet werden. Die folgende Aufzählung enthält dafür typische Testmethoden, ohne den Anspruch auf Vollständigkeit zu erheben (siehe eine ausführliche Beschreibung in Crispin/Gregory 2009):

Exploratives Testen: Ein Qualitätssicherer versucht ein unerwünschtes Verhalten der Software zu provozieren, indem er die Software auf ungewöhnliche Weise gebraucht/missbraucht. Exploratives Testen erfolgt immer manuell, auch wenn Tools ein solches Testen unterstützen können.

GUI-Tests: GUI-Tests sind funktionale Tests, bei der die grafische Benutzerschnittstelle (*graphical user interface* – GUI) mit getestet wird. Solche Tests können automatisiert sein. Jedoch sind

solche Tests langsam, aufwendig zu erstellen und fehleranfällig. Daher sollten sie Unit-Tests und funktionale Tests ohne GUI-Prüfung lediglich ergänzen.

Performance- und Load-Tests: Performance- und Load-Tests dienen dazu, das Zeitverhalten und das Verhalten des Systems unter hoher Last zu testen. Solche Tests sollten in einer Umgebung stattfinden, die der Produktivumgebung weitgehend entspricht. Die Tests erfolgen in der Regel toolbasiert und automatisiert. Der Aufbau solcher Tests erfordert Spezialwissen.

Regressionstests: Werden Tests regelmäßig wiederholt, so spricht man von Regressionstests. Regressionstests sind in den meisten Fällen automatisiert. Aus Performancegründen sind oftmals nicht alle automatisierten Tests Regressionstests.

Security-Tests: Ähnlich wie bei Performance- und Lasttests ist auch für Security-Tests spezielles Know-how notwendig. Die Tests erfolgen sowohl toolbasiert als auch explorativ.

Usability-Tests/Persona-Tests: Bei Usability- und Persona-Tests werden die Bedienbarkeit und die Funktionalität hinsichtlich Vollständigkeit und Angemessenheit getestet. Die Tests erfolgen manuell.

Das Entwicklungsteam entscheidet, welche dieser Tests sie in welchem Umfang durchführen. Hat die Software diese Tests bestanden, so können selbst auf der Produktivumgebung nach der Installation weitere Tests durchgeführt werden, bevor die Software freigeschaltet wird (sog. *Blue-Green-Deployment*)[6].

Schließlich entscheidet der Product-Owner, ob er die Umsetzung der Anforderungen akzeptiert. Maßstab sind hierbei die von ihm zuvor formulierten Akzeptanzkriterien. Der Product-Owner entscheidet, ob das erstellte Softwareinkrement ausgeliefert wird.

Die ausgelieferte Software kann dazu verwendet werden, um Hypothesen zu testen (sog. *A/B-Tests*). Solche Tests werden insbesondere zum Testen der *funktionalen Eignung* und der *Usability* eingesetzt. Beispielsweise wird ein alternatives Bedienkonzept realisiert und ausgeliefert, jedoch nur ein Teil der Nutzer dafür freigeschaltet. Durch die Analyse des Nutzungsverhaltens kann entschieden werden, ob die unternehmerischen Ziele mit dem neuen Bedienkonzept besser erreicht werden als mit dem bisherigen. A/B-Tests basieren auf das Beobachten des Nutzungsverhaltens und gehören damit zu Testmethoden, die im Abschnitt 2.5 ausführlicher beschrieben sind.

[6] Bei einem Blue-Green-Deployment wird die bestehende Software zunächst weiter – z. B. auf der Blue-Umgebung – betrieben und verwendet, während auf einer zweiten Umgebung – in diesem Beispiel die Green-Umgebung – die neue Software installiert und getestet wird. Sind alle Tests erfolgreich bestanden worden, werden dann die Anfragen an das System auf die Green-Umgebung geleitet. Die Blue-Umgebung ist nach Abschluss aller dort verbliebenen Aktivitäten inaktiv. Wenn ein neues Update eingespielt wird, wird dann die Blue-Umgebung zur Installation und zum Testen verwendet usw.

2.3.2. Testautomatisierung und Staging-Pipeline

Testautomatisierung, Continuous Integration und Continuous Delivery sind typisch im agilen Umfeld, können aber grundsätzlich genauso in der klassischen phasenorientierten Softwareentwicklung genutzt werden. Aufgrund der hohen Integrations- und Auslieferungsfrequenz bei der agilen Entwicklung sind viele der Methoden und Werkzeuge zur qualitätsgesicherten, automatisierten Integration und Auslieferung von Software im agilen Umfeld entstanden und dort zuerst eingesetzt worden.

Der oben beschriebene Prozess zum Testen der Software kann größtenteils automatisiert werden. Dazu zählen insbesondere die Unit-Tests, statische Code-Analysen, Funktionstests (mit oder ohne GUI-Tests), Performance- und Load-Tests, das Blue-Green-Deployment sowie A/B-Tests. Demgegenüber lassen sich explorative Tests, Usability- und Persona-Tests nicht automatisieren. Auch wenn Tools die Qualitätssicherer unterstützen, so bleibt es noch immer Aufgabe der Qualitätssicherer, sie manuell durchzuführen.

Schwierig ist teilweise auch die Automatisierung von Security-Tests. Security-Tests bestehen aus statischen Code-Analysen, die sich automatisieren lassen, und manuellen explorativen Penetrationstests durch Security-Spezialisten. Auch hier stehen Werkzeuge zur Verfügung, die die manuelle Arbeit unterstützen, aber nicht ersetzen können.

Das Entwicklungsteam setzt die unterschiedlichen Testmethoden, ob automatisiert oder manuell, normalerweise zu unterschiedlichen Zeitpunkten während der Umsetzung einer Anforderung ein. Entsprechend werden die Testmethoden in unterschiedlichen Umgebungen verwendet.

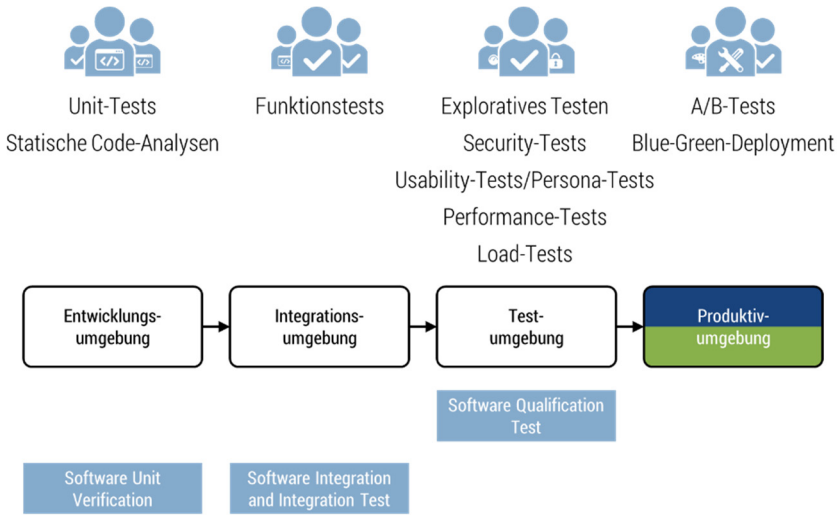

Abbildung 11: Beispiel für eine Staging-Pipeline. Für jede Umgebung sind die typischen Testmethoden aufgelistet.

In Abbildung 11 sind mögliche Systemumgebungen dargestellt. Auf der Entwicklungsumgebung, die typischerweise jeder Entwickler lokal auf seinen Rechner betreibt, findet die eigentliche Softwareentwicklung, die Entwicklung und Durchführung der Unit-Tests und ggf. statische Code-

Analysen statt. Auf der Integrationsumgebung testen primär Qualitätssicherer die funktionale Integration. Auf einer oder mehreren separaten Testumgebungen werden weiterführende Tests durchgeführt. Dabei beteiligt sind in erster Linie Qualitätssicherer, die ggf. von Q-Spezialisten unterstützt werden (siehe Abschnitt 2.4). Auf der Produktivumgebung finden ggf. das Blue-Green-Deployment und A/B-Tests statt.

Die hier beschriebenen Umgebungen, Testmethoden und deren Zuordnung zueinander sind exemplarisch. Je nach Produkt, Organisation und Kritikalität einzelner Qualitätseigenschaften können Staging-Pipelines sehr unterschiedlich aufgebaut sein.

Es existieren Werkzeuge, mit denen das Zusammenspiel der Umgebungen verwaltet und orchestriert werden kann. Beispielsweise kann sichergestellt werden, dass nur Softwaremodule, die zuvor durch Unit-Tests verifiziert worden sind, in die Gesamtlösung integriert werden, oder aufwendige Funktionstests, die aus Performance-Gründen nicht auf den lokalen Umgebungen der Entwickler durchgeführt werden, routinemäßig nachts durchlaufen werden.

2.3.3. Bedeutung von Unit- und A/B-Tests im agilen Umfeld

Alle im Abschnitt 2.3.1 beschriebenen Testmethoden werden meist auch in klassischen Entwicklungsprojekten eingesetzt. Sie werden nicht exklusiv in der agilen Entwicklung genutzt. Neben dem hohen Automatisierungsgrad des Testens ist für die agile Entwicklung jedoch typisch, dass Unit-Tests und A/B-Tests intensiv genutzt werden.

Wie im Abschnitt 2.2.2 dargelegt, bilden Unit-Tests das Rückgrat des agilen Testens. Demgegenüber spielen GUI-Tests, manuell oder automatisiert, eher eine untergeordnete Rolle und dienen primär einem stichprobenartigen Testen der zuvor getesteten Module und Services über alle Architekturschichten hinweg.

Aufgrund der häufigen Auslieferung von Releases bietet es sich im agilen Umfeld an, mit Hilfe von A/B-Tests Hypothesen gerade hinsichtlich der Bedienbarkeit und neuer Funktionen zunächst bei einem kleinen Benutzerkreis zu testen. Statt wie bei klassischen Festpreisprojekten über die Sinnhaftigkeit von Anforderungen zu spekulieren und dann verbindlich ins Lastenheft mit aufzunehmen, können bei der agilen Entwicklung Anforderungen anhand des Nutzungsverhaltens frühzeitig geprüft und ggf. verfeinert oder verworfen werden.

2.3.4. Der Qualitätssicherer im agilen Entwicklungsteam

Der Qualitätssicherer (Qualitätsmanager, Tester oder wie auch immer dieser bezeichnet sein mag) existiert im agilen Entwicklungsteam nicht als Rolle. Beispielsweise kennt Scrum nur die Rollen Product-Owner, ScrumMaster und Entwicklungsteam. Das Entwicklungsteam sollte interdisziplinär zusammengesetzt sein, sodass alle Kompetenzen für eine hochwertige und effiziente Softwareentwicklung im Team vorhanden sind. Nichts spricht dagegen, dass speziell ausgebildete

Qualitätssicherer Teil des Entwicklungsteams sind und disziplinarisch vielleicht sogar Q-Abteilungen angehören. Problematisch ist es jedoch, wenn ein solcher Qualitätssicherer sich in der klassischen Tester-Rolle sieht und sich als Qualitätspolizist versteht, der ausschließlich für das Verifizieren definierter Anforderungen verantwortlich ist. Ein solches Verständnis wäre für ein agiles Team kontraproduktiv. Stattdessen sollten Qualitätssicherer und das gesamte agile Team folgendes Verständnis haben:

- Agile Qualitätssicherer sind Teil des Entwicklungsteams und nehmen an allen Treffen des Entwicklungsteams gleichberechtig teil.
- Agile Qualitätssicherer nehmen keine separate Rolle im Team ein. Sie tragen nicht allein die Verantwortung für die Softwarequalität, sondern übernehmen als vollwertige Teammitglieder gemeinsam mit allen anderen Teammitgliedern die Verantwortung für das gesamte Produkt und dessen Entstehung.
- Agile Qualitätssicherer unterstützen das Team wo sie können. So können sie den Product-Owner bei der Identifikation und Formulierung von Akzeptanzkriterien, Programmierern bei der Entwicklung von Unit-Tests und Systemadministratoren bei dem Aufbau und der Pflege der Testautomatisierung helfen.

2.4. Q-Spezialisten

Agile Entwicklungsteams sind meist kleine Einheiten. Scrum legt die Größe eines Entwicklungsteams mit drei bis neun Mitgliedern explizit fest. Auch wenn mehrere Teams an einer Software arbeiten, so sollte jedes einzelne Team in der Lage sein, Funktionsbereiche eigenständig zu realisieren.

Das agile Team verantwortet die entwickelte Software einschließlich ihrer Qualität. Einige Qualitätseigenschaften, wie z. B. funktionale Korrektheit, Ausgereiftheit oder Analysierbarkeit, sind immanent mit der Softwareentwicklung verknüpft. Diese Eigenschaften sollte jedes Team berücksichtigen, umsetzen und prüfen können. Andere Eigenschaften hingegen erfordern Spezialwissen, welches ggf. nicht ausreichend im Team vorhanden ist. Hierzu zählen beispielsweise Security-, Performance- oder Bedienungseigenschaften. Insbesondere das Testen der Security durch Penetrationstests oder der Aufbau von Performance- oder Last-Testständen erfordert spezielles Know-how.

In diesen Fällen sollten Q-Spezialisten temporär oder beratend in das Entwicklungsteam eingebunden werden. Entscheidend ist auch hierbei, dass das Entwicklungsteam die Verantwortung für das Gesamtprodukt weiterhin übernimmt.

Gerade in großen Organisationen bietet es sich an, dass Expertenpools für spezielle Qualitätsfragen aufgebaut und den Entwicklungsteams zur Seite stehen. Solange es um die technische Umsetzung geht, muss jedoch vermieden werden, dass diese Expertenpools für die Teams verbindliche Standards entwickeln und deren Umsetzung einfordern. Dies würde die Autonomie der Entwicklungsteams schmälern und somit ein agiles Arbeiten gefährden. Anders bei fachlichen Qualitätsanforderungen, wie im folgenden Abschnitt beschrieben.

2.5. Qualitätsmanagement von außen

Im agilen Umfeld ist es das Entwicklungsteam, das die Qualität der Umsetzung von Anforderungen verantwortet. Die Anforderungen selbst kommen, wie bei beim klassischen Vorgehen auch, von Stakeholdern. Bei den Anforderungen kann es sich um funktionale Anforderungen, Qualitätsanforderungen und Randbedingungen, wie z. B. rechtliche Anforderungen, handeln. Der Product-Owner verantwortet das Einsteuern dieser Anforderungen in die Entwicklung. Entscheidend ist, dass die Anforderungen rein fachlicher Natur sind, d. h. keine Vorgaben über die Art der Umsetzung enthalten. Dann ist es auch bei einer agilen Entwicklung gerechtfertigt, dass Stakeholder die Umsetzung zwingend erforderlicher Anforderungen verlangen und andernfalls die Freigabe verweigern können (z. B. bei Anforderungen hinsichtlich des Datenschutzes oder der Funktionssicherheit).

Ein Product-Owner oder Stakeholder kann grundsätzlich prüfen, ob die von ihm geforderten funktionalen Anforderungen umgesetzt worden sind. Einige Softwarequalitätseigenschaften sind von außen relativ gut zu bewerten, z. B. durch Analyse des Nutzungsverhaltens und Testen der Software. Dazu zählen die Qualitätseigenschaften (gemäß ISO 25010):

- Functional suitability: functional completeness, functional correctness, functional appropriateness
- Time behavior
- Compatibility: co-existence, interoperability
- Usability: appropriateness recognizability, learnability, operability, user error protection, user interface aesthetics, accessibility
- Reliability: maturity, availability, fault tolerance, recoverability
- Installability (falls die Nutzer selbst die Software installieren)

Obwohl diese Qualitätseigenschaften generell schwerer zu validieren sind als funktionale Anforderungen, ist deren Umsetzung prinzipiell sichtbar und prüfbar.

Die wirklich schwierig zu validierenden Qualitätseigenschaften sind:

- Performance efficiency: resource utilization, capacity
- Security: confidentiality, integrity, non-reputation, accountability, authenticity
- Maintainability: modularity, reusability, analyzability, modifiability, testability
- Portability: adaptability, installability (falls der Hersteller die Software installiert), replaceability

Diese Qualitätseigenschaften lassen sich nicht ‚sehen' und lassen sich von außen nicht nachweislich verifizieren. Sie liegen versteckt unter der Motorhaube. Dennoch sind die Stakeholder von Qualitätsdefiziten dieser Art betroffen: Unzureichende Wartbarkeit führt zu steigenden Entwicklungskosten, fehlende Portierbarkeit kann in eine technologische Sackgasse führen, Sicherheitslücken können erhebliche wirtschaftliche und rechtliche Folgen haben. Ein gutes Beispiel hierfür ist das unbeabsichtigte Beschleunigen von Fahrzeugen von Toyota: Aufgrund einiger ungetesteter und überkomplexer Funktionen, wurden Softwarefehler nicht gefunden und Personen kamen zu Tode (Safety Research & Strategies, Inc. 2013).

2.5.1. Das Dilemma des klassischen Anforderungsmanagements

Klassischerweise legt ein Requirements Engineer Qualitätseigenschaften in der Softwarespezifikation fest – allgemein und unabhängig von funktionalen Anforderungen oder bezogen auf konkrete funktionale Anforderungen (siehe z. B. Rupp et al. 2009, Seite 247 ff.).

Auch hier zeigt sich die grundsätzliche Schwäche des phasenorientierten Vorgehens. Viele Qualitätsanforderungen sind in der Praxis vage und unpräzise formuliert, und die Kriterien für gute Anforderungsspezifikation wie Testbarkeit und Schätzbarkeit sind meistens nicht erfüllt. Der Aufwand, gute Qualitätsanforderungen zu formulieren, ist hoch. Beispielsweise ist die Anforderung: „Die Software muss so konzipiert sein, dass Änderungen leicht umgesetzt werden können", vollkommen unspezifisch. Es handelt sich dabei eher um eine Absichtserklärung als um eine konkrete Anforderung. Um spezifisch zu sein, müsste geklärt werden: Was bedeutet ‚einfach', d. h. welcher Aufwand wäre akzeptabel? Um welche Art von Modifikationen handelt es sich? Die Beantwortung dieser Fragen würde eine umfassende, zeitaufwendige Analyse und Spezifikation erfordern. Der Aufwand für das Verifizieren der geforderten Qualitätsanforderungen wäre ebenfalls hoch. Zudem stellt sich auch hier die Frage, ob die zuvor festgelegten Anforderungen den tatsächlichen Anforderungen der Nutzer überhaupt entsprechen.

Die Präzisierung der Qualitätsanforderungen, die für den späteren Anwender unsichtbar sind, bedeutet zudem, dem Entwicklungsteam vorzugeben, wie es die Software zu entwickeln hat. Ein solches Micromanagement ist mit agiler Entwicklung unvereinbar. Im agilen Umfeld sollte ein Qualitätsmanagement außerhalb des Entwicklungsteams vielmehr die Validierung des fertigen Inkrements hinsichtlich der geschäftlichen Qualitätskriterien im Fokus haben.

2.5.2. Feldbeobachtung

Product-Owner und Stakeholder können das Nutzungsverhalten im Feld beobachten und so prüfen, ob und in welchem Umfang Qualitätseigenschaften erfüllt sind. Sie beobachten die Nutzung der Software durch echte Nutzer unter realen Bedingungen. Im agilen Umfeld wird die Software typischerweise früh ausgeliefert und Updates kontinuierlich eingespielt. Durch Feldbeobachtung können so erkannte Anforderungen in die laufende Entwicklung eingesteuert werden.

Solange Qualitätsdefizite nur einen geringen Schaden hervorrufen können, lassen sich so Produkte in einem frühen Entwicklungsstadium und Hypothesen testen. Beispielsweise dürfte das Testen alternativer Benutzungskonzepte durch einen Teil der Nutzer in vielen Fällen unkritisch sein. Solche A/B-Tests sind bei der Entwicklung von Webapplikationen gängiges Mittel. Auch für Qualitätseigenschaften, die für Stakeholder und Anwender nicht unmittelbar sichtbar sind, können Methoden der Feldbeobachtung zum Einsatz kommen, wie im Folgenden beschrieben.

Performance Efficiency

Mit Ausnahme des Zeitverhaltens werden Performance-Eigenschaften von den Nutzern der Software selbst nicht bemerkt. Aber sie lassen sich von Systemadministratoren feststellen und schlagen sich in den Kosten für zusätzlichen Speicher oder zusätzliche Rechenleistung nieder. Diese Kosten können erfasst werden und so möglicherweise ein Indiz für ineffiziente Ressourcennutzung sein.

Security

Sicherheitslücken können zu erheblichen Reputationsschäden und rechtlichen Konsequenzen führen. Die Sicherheit sollte daher vorab unter möglichst realen Bedingungen validiert werden (siehe Abschnitt 2.5.3). Dennoch können auch nach der Auslieferung externe Sicherheitsexperten auf Sicherheitslücken hinweisen und Administratoren Auffälligkeiten hinsichtlich möglicher Angriffe (z. B. durch Intrusion-Detection) feststellen und bewerten. Entsprechende Kommunikationswege und Monitoring-Maßnahmen sollten eingerichtet werden.

Maintainability

Die Wartbarkeit von Software besteht aus den Eigenschaften Modularität (*modularity*), Wiederverwendbarkeit (*reusability*), Analysierbarkeit (*analyzability*), Modifizierbarkeit (*modifiability*) und Testbarkeit (*testability*). Alle diese Eigenschaften haben denselben Zweck: Sie ermöglichen eine effiziente und somit wirtschaftliche Weiterentwicklung der Software während des gesamten Produktlebenszyklus‘.

Da die Komplexität von Software mit der Zeit typischerweise zunimmt, führen Defizite hinsichtlich der Wartbarkeit zu steigenden Entwicklungs- und Wartungskosten. Product-Owner und Stakeholder können diese Kosten erfassen und bewerten. Eine sinkende Durchlaufrate in der Entwicklung – bei Scrum die sog. *Velocity* – ist ein starkes Indiz dafür, dass die Software nicht mehr ausreichend wartbar ist.

Portability

Die Anpassungsfähigkeit (*adaptability*) und die Austauschbarkeit (*replaceability*) von Software können im produktiven Betrieb nur sehr begrenzt ermittelt werden, da z. B. das Anpassen für ein anderes Betriebssystem oder der Austausch von Softwarekomponenten eher singuläre Ereignisse sind und nur wenige Daten darüber zur Verfügung stehen. Dennoch kann die Zeit, die für die Installation einer Software benötigt wird, regelmäßig erfasst werden. Eine zunehmende Dauer für die Installation ist ein Indikator für fehlende Automatisierung oder fehlende Dokumentation.

2.5.3. Prelive-Simulationen

Das Simulieren möglichst realer Produktivbedingungen ist eine Alternative bzw. Ergänzung zur Validierung der Softwarequalität im Feld. Das ist besonders dann wichtig, wenn Qualitätsmängel zu inakzeptablen Schäden im Produktivbetrieb führen würden. Mit Hilfe von Simulationen können Defizite vor der Auslieferung entdeckt werden. Sie sind daher ein wichtiges Mittel des Qualitätsmanagements sowohl innerhalb als auch außerhalb des Entwicklungsteams. Jedoch gilt auch hier für Product-Owner und Stakeholder, Micromanagement zu vermeiden und die rein fachlichen Auswirkungen zu betrachten.

Im Folgenden sind mehrere Methoden des Softwarequalitätsmanagements beschrieben, die fachliche Auswirkungen von Qualitätsmängeln noch vor der Auslieferung betrachten.

Performance Efficiency

Load-Tests sind gängige Methoden, um die Kapazität von IT-Systemen zu testen. Typischerweise erfolgen diese Tests auf Prelive-Umgebungen, die technisch so weit wie möglich der Produktivumgebung entsprechen. Die bei den Tests verarbeiteten Daten sollten in Art und Umfang ebenfalls mit den Daten der Produktivumgebung vergleichbar sein.

Security

Die Sicherheit von IT-Systemen kann mittels Penetrationstests geprüft werden. Im Gegensatz zu tatsächlichen Angriffen auf das Produktivsystem, können innerhalb einer Organisation *White Penetrationstests* durchgeführt werden, d. h. Penetrationstests, bei denen der Quellcode für den Qualitätssicherer einsehbar ist. Penetrationstests sind aufwendig und sollten von Security-Experten durchgeführt werden. Diese organisatorische Trennung von Softwareentwicklung und Softwareauslieferung einerseits und Penetrationstests andererseits darf jedoch nicht zur Trennung von Verantwortlichkeiten und Micromanagement führen. Die Verantwortlichkeit für die Softwaresicherheit verbleibt bei dem Entwicklungsteam. Security-Experten beraten das Entwicklungsteam, ohne durch rigide Sicherheitsrichtlinien das Entwicklungsteam in dessen Autonomie einzuschränken.

Analyzability

Während ein Product-Owner oder Stakeholder Wartbarkeitseigenschaften wie Modularität, Wiederverwendbarkeit und Modifizierbarkeit meist nur durch den Verlauf der Entwicklungskosten indirekt abschätzen kann, so kann die Analysierbarkeit durchaus von außen getestet werden. Analysierbarkeit soll sicherstellen, dass auch neu hinzugekommene Entwickler die Softwarearchitektur und den Quellcode in einer angemessenen Zeit verstehen können. Eine saubere Dokumentation – das kann gutgeschriebener Quellcode, Kommentare im Quellcode, dokumentierte Testfälle oder sonstige Dokumente sein – haben den Zweck, die Software leicht verständlich und damit analysierbar zu machen. Ein neu hinzugekommener Entwickler sollte bei guter und angemessener Dokumentation in der Lage sein, z. B. Softwarefehler zu beheben.

Dies kann wie folgt getestet werden: Ein neuer Entwickler soll versuchen, einen Softwarefehler zu beheben. Er erhält den Quellcode und alle anderen verfügbaren Dokumente, aber er darf nicht mit anderen Teammitgliedern sprechen. Der Entwickler soll laut vor dem Team aussprechen, wie er sich in die Dokumentation einarbeitet und versucht, die Aufgabe zu lösen. Anschließend diskutieren der Entwickler, das Team und die Stakeholder die Ergebnisse dieses Tests. Auf diese Weise lassen sich Schwächen erkennen: fehlende Dokumentation, zu umfangreiche oder unnütze Dokumentation, Inkonsistenzen, unklare Benennung von Methoden, Variablen oder Klassen oder ein schwer nachvollziehbares Softwaredesign.

Adaptability

In vielen Fällen ist es nicht zwingend erforderlich, dass eine Software auf allen Betriebssystemen, Browsern oder Endgeräten läuft. Aber es kann eine Sackgasse sein, wenn Software in einer speziellen technischen Umgebung gefangen ist. Software sollte so entwickelt werden, dass eine Anpassung an andere Technologien möglich ist. Dies kann ein Product-Owner prüfen, indem er eine Anpassung eines kleinen Teils der Software an eine andere Technologie frühzeitig in die Entwicklung einsteuert und den Aufwand dazu erfasst. Ist der Aufwand groß, so ist dies ein Indiz für fehlende Adaptierbarkeit. In diesem Fall sollten die Gründe dafür mit dem Entwicklungsteam diskutiert und Maßnahmen zur Verbesserung der Flexibilität des Softwaredesigns festgelegt werden.

2.6. Exkurs: Dokumentation

Im Agilen Manifest heißt es: „Working software over comprehensive documentation.“ Gerade dieser agile Grundsatz scheint gerne missverstanden zu werden. So die Meinung eines Funktionssicherheitsingenieurs auf der Automotive SYS 2017: „Agile Entwicklung kann nicht funktionieren, da nichts dokumentiert wird.“

Der agile Grundsatz besagt, dass eine funktionierende Software im Fokus der Softwareentwicklung stehen soll, nicht die Dokumentation. Dies bedeutet jedoch keinesfalls, dass Dokumentation im agilen Umfeld hinfällig wäre. Auch bei der agilen Entwicklung gibt es Dokumentation. Das Team soll jedoch eine Dokumentation um ihrer selbst willen vermeiden.

Das agile Team verantwortet die entwickelte Software und die Art der Softwareentwicklung. Damit entscheidet das Team grundsätzlich auch über Art und Umfang der Dokumentation. Das Team verantwortet zudem die Qualität und somit die Wartbarkeit der Software, die durch eine ausreichende Dokumentation sichergestellt werden muss. Dennoch gibt es auch Dokumente, die von außen gefordert sind – und deren Erstellung damit vom Product-Owner verantwortet wird.

Im Folgenden wird beschrieben, welche Dokumentationsarten im Kontext der Softwareentwicklung vorkommen können und wie mit ihnen im agilen Umfeld umgegangen werden sollte.

2.6.1. Dokumentation zur Softwareentwicklung

Bei der klassischen phasenorientierten Softwareentwicklung gibt es eine Reihe von Dokumenten, die für die Entwicklung der Software vorgesehen sein können. Dazu zählen z. B. Lastenhefte, Pflichtenhefte, Testpläne oder Analyseberichte. So werden in Automotive SPICE allein für die *Software Engineering Process Group* (SWE) nicht weniger als 19 Arbeitsergebnisse (*output work products*) vorgeschlagen, von denen 14 Dokumenttypen sind, die primär für die Softwareentwicklung vorgesehen sind (A-SPICE 2017).

Scrum hingegen sieht für die Softwareentwicklung neben dem Inkrement selbst nur drei Artefakte vor: das Product-Backlog, das Sprint-Backlog und die Definition-of-Done. Welche Artefakte darüber hinaus erstellt werden, entscheidet das Entwicklungsteam. Entscheidend ist das funktionierende Inkrement am Ende eines Sprints. Genau darauf bezieht sich der agile Grundsatz „Working software over comprehensive documentation“.

2.6.2. Dokumentation zur Freigabe

Idealerweise gibt es im agilen Umfeld keinen formalen Freigabeprozess, der eine zusätzliche Dokumentation erfordern würde. Stattdessen entscheidet der Product-Owner, der Teil des agilen Teams ist und meist täglich in enger Abstimmung mit dem Entwicklungsteam steht, darüber, wann ein fertiges Inkrement ausgeliefert wird.

Nichtsdestotrotz können in der Praxis äußere Randbedingungen Dokumentationen zur Freigabe erforderlich machen. So könnten beispielsweise Nachweise gefordert werden, die gegenüber Behörden den Schutz der Software vor Manipulationen dokumentieren. Solche Dokumentationen können zwar das agile Arbeiten behindern, meist dürfte Agilität dennoch grundsätzlich möglich sein.

In solchen Fällen ist es Aufgabe des Product-Owners, die damit verbundenen Anforderungen in den Entwicklungsprozess einzusteuern und deren Umsetzung vor Auslieferung der Software sicherzustellen. Dazu müssen Prozesse außerhalb der eigentlichen agilen Arbeit und der agilen Teams vorhanden sein, die gewährleisten, dass alle relevanten Anforderungen dem Product-Owner bekannt sind.

2.6.3. Dokumentation zur Softwarewartung

Die Dokumentation zur Softwarewartung muss sicherstellen, dass ein Softwaremodul auch dann noch analysierbar und verständlich ist, wenn längere Zeit nicht mehr an dem Modul gearbeitet wurde und die damaligen Entwickler nicht mehr dem Team zur Verfügung stehen. Während der Softwareentwicklung ist daher zu prüfen, ob die Wartbarkeit ausreichend gewährleistet ist (siehe dazu „Analyzability“ in Abschnitt 2.5.3).

Hierbei sollte berücksichtigt werden, dass zur Dokumentation alle Arten von Artefakten dienen können. Dazu zählen neben dem Quellcode auch die automatisierten Testfälle, die im agilen Umfeld in großer Zahl vorhanden sein sollten.[7]

2.6.4. Dokumentation als Teil des Produkts

Dokumente können für den Anwender der Software bestimmt sein, wie z. B. Benutzerhandbücher. Solche Dokumente sind Teil des Produktes und in der agilen Entwicklung als solche zu behandeln. So könnte beispielsweise der Product-Owner ein PBI dazu formulieren („Als Anwender möchte ich ein Benutzerhandbuch zur Software zur Verfügung haben, um mich mit der Funktionsweise der Software vertraut machen zu können“) und diese in die Entwicklung einsteuern. Um die kontinuierliche Pflege des Benutzerhandbuchs sicherzustellen, wird dann die Definition-of-Done so erweitert, dass eine Anforderung erst dann erfolgreich abgeschlossen ist, wenn das Benutzerhandbuch aktualisiert worden ist.

Jedoch gilt auch hier: Die Dokumentation für den Anwender ist kein Selbstzweck. Das Ziel ist es, dass der Anwender die Nutzung der Software erlernen kann. Dazu gibt es Alternativen zum klassischen Benutzerhandbuch, die in Betracht gezogen werden sollten.[8]

[7] Klassischerweise besteht die Dokumentation der Software in einer eigens dafür erstellten Systemdokumentation. Der Autor hat die Erfahrung gemacht, dass solche Systemdokumentationen aufgrund des hohen Pflegeaufwands oft nicht dem aktuellen Entwicklungsstand der Software entsprechen. Eine nicht aktuelle Systemdokumentation wirft beim Leser oft eher Fragen auf anstatt Fragen zu beantworten. Quellcode und Testfälle, die kontinuierlich als Regressionstests getestet werden, sind hingegen immer aktuell.

[8] So werden die meisten Spiele-Apps von den Nutzern spielerisch erlernt. Das Lesen von Handbüchern dürfte selbst bei komplexen Spielen unüblich sein.

2.7. Prozessqualität

Bislang stand die Qualität der erstellten Software im Vordergrund der Betrachtung. Das Qualitätsmanagement beinhaltet klassischerweise jedoch auch das Definieren von Anforderungen an die Prozessqualität und deren Überprüfung.

Im Sinne des agilen Grundsatzes “individuals and interactions over processes and tools” hat bei einer agilen Entwicklung die Gestaltung des Entwicklungsprozesses aus dem agilen Team heraus eine größere Bedeutung als die Vorgabe von Prozessanweisungen von außen. Agile Teams arbeiten typischerweise nach einem agilen Prozess-Framework wie z. B. Scrum. Das agile Team wählt und gestaltet die konkreten Entwicklungsmethoden innerhalb dieses Rahmens vorwiegend selbst und validiert und optimiert die Methoden kontinuierlich z. B. durch regelmäßige Retrospektiven.

Der Team-Agility-Master (ScrumMaster bei Scrum) spielt hierbei eine wichtige Rolle: Er moderiert Retrospektiven und unterstützt das Team maßgeblich, Hindernisse innerhalb und in der Zusammenarbeit mit anderen Organisationseinheiten zu überwinden. Damit ist der Team-Agility-Master entscheidend mitverantwortlich für die Qualität des Entwicklungsprozesses. Wie in der klassischen Entwicklung kann auch er Anforderungen hinsichtlich der Prozesse einbringen und validieren. Wie im agilen Umfeld üblich geschieht dies nicht durch einmalige formale Anweisungen, sondern im Dialog und durch kontinuierliche Zusammenarbeit mit dem Entwicklungsteam und dem Product-Owner.

2.8. Fazit

Agile Softwareentwicklung unterstützt die Entwicklung hochwertiger Software in vielerlei Hinsicht: Durch ein frühes Kundenfeedback können Verbesserungspotentiale gerade hinsichtlich der funktionalen Eignung und der Usability frühzeitig erkannt und umgesetzt werden; das häufige Integrieren und Ausliefern der Software verbessert die Wartbarkeit und Installierbarkeit; die Qualität der Entwicklung wird vom autonomen Team und damit von den Entwicklern, die die Software am besten kennen, verantwortet. Dies bedeutet jedoch auch, dass die Qualität der Software entscheidend von der Qualität und damit von der Motivation und der Kompetenz des Teams, insbesondere des Entwicklungsteams, abhängt. Qualitätsmanagement im agilen Umfeld ist nicht zuletzt Talentmanagement.

So wie agile Softwareentwicklung das Anforderungsmanagement nicht obsolet gemacht, sondern grundlegend verändert hat, verändert es auch das Qualitätsmanagement. Qualitätsmanagement steht nicht mehr als Kontrollinstanz zwischen der eigentlichen Entwicklung und der Auslieferung, sondern findet im Wesentlichen im Entwicklungsteam selbst sowie durch Beobachtung des Nutzungsverhaltens im Feld statt.

Moderne Werkzeuge zum automatisierten Testen, Integrieren und Ausliefern der Software ermöglichen es heute dem Entwicklungsteam selbst, Software qualitätsgesichert und mit nur geringem manuellen Testaufwand auszuliefern. Explizite Testkompetenz durch erfahrene Qualitätssicherer sind auch im agilen Team weiterhin wertvoll, jedoch verändert sich die Rolle des Qualitätssicherers: weg vom Qualitätspolizisten, hin zu einem vollwertigen Mitglied des Entwicklungsteams, der alle Aktivitäten des Teams mit seinen speziellen Fähigkeiten unterstützt.

Das agile Entwicklungsteam wird ggf. von externen Q-Experten unterstützt. Die Verantwortung der Produktqualität verbleibt jedoch beim agilen Team.

Der Product-Owner und externe Stakeholder stellen Anforderungen an die Software, auch hinsichtlich der Qualität. Sie betreiben jedoch kein Micromanagement und respektieren die Autonomie des Entwicklungsteams hinsichtlich der technischen Umsetzung. Ihr Blick ist vielmehr darauf gerichtet, wie das Produkt sicher und sinnvoll im Markt genutzt wird. Damit schließen sie den so wichtigen Regelkreis, der die Erfahrungen im Feld in die weitere Entwicklung einfließen lässt.

3. SOFTWAREENTWICKLUNG IN DER AUTOMOBILINDUSTRIE

Die Automobilindustrie entwickelt Software nicht grundsätzlich anders als andere Branchen. Die Elektronen, die zur Datenverarbeitung in einem Motorsteuergerät fließen, sind identisch mit den Elektronen, die von Social-Media-Plattformen oder Spiele-Apps verwendet werden. Dennoch müssen je nach Produktart Besonderheiten berücksichtigt werden, die das Entwickeln digitaler Produkte in der Automobilindustrie von dem typischerweise agilen Entwickeln von Webapplikationen unterscheiden:

- Eingebettete Systeme, wie z. B. Steuergeräte in einem Fahrzeug, werden für spezielle, klar definierte Anwendungsfälle entwickelt. Diese Zweckgebundenheit bezieht sich nicht nur auf die Software, sondern auch auf die Hardware. Hard- und Softwareentwicklung erfolgen Hand in Hand und sind Teil der Systementwicklung. Bei der Anwendungsentwicklung hingegen wird auf existierende Hardware (Smartphones, Server usw.) inkl. der dazugehörigen Betriebssysteme und Basissoftware aufgesetzt. Eine solche Softwareentwicklung erfolgt unabhängig von der Hardwareentwicklung.
- Ein Fahrzeug ist ein sicherheitskritisches System, eine Webapplikation oder eine App in der Regel nicht. Die Anforderungen an eine sicherheitskritische Komponente, wie z. B. ein Motorsteuergerät oder einen Parkassistenten, sind hinsichtlich Performance und Ausgereiftheit weitaus höher als bei den meisten Webapplikationen oder Apps.
- Für die Typgenehmigung von Fahrzeugen müssen rechtliche Voraussetzung erfüllt werden, die auch IT-Systeme und deren Entwicklungsprozesse betreffen. Darüber hinaus gibt es branchenspezifische Standards und Empfehlungen, die einzuhalten sind. Dazu zählen u. a. Automotive SPICE und die ISO 26262. Diese haben unmittelbaren Einfluss auf die Entwicklung von Steuergeräten für das Fahrzeug und müssen auch im agilen Umfeld berücksichtigt werden.
- Die IT-Systeme in Fahrzeugen werden fast ausschließlich von Zulieferern entwickelt. Die OEMs, die das Gesamtprodukt Fahrzeug gegenüber den Endkunden und Behörden verantworten, müssen die Qualität gewährleisten, ohne selbst wichtige IT-Komponenten entwickelt und detaillierten Einblick zu haben. Dies stellt eine besondere Herausforderung an das Qualitätsmanagement in der Automobilindustrie dar.

Heute werden fast alle Apps, Web- und Desktopapplikationen agil entwickelt, seltener hingegen eingebettete Systeme in der Automobilindustrie. Aufgrund der weiter zunehmenden Digitalisierung der Produkte und Dienstleistungen auch in der Automobilindustrie hat sich das digitale Angebotsportfolie der OEMs jedoch erheblich erweitert: Das Infotainmentsystem ist schon seit längerer Zeit eines der umfangreichsten und komplexesten IT-Systeme im Fahrzeug; Fahrzeuge sind zunehmend mit der Außenwelt vernetzt (V2X) und greifen auf digitale Dienste außerhalb des Fahrzeugs zu; OEMs entwickeln und betreiben Webplattformen und Apps als Kommunikationsmedien für ihre Kunden. Durch diese Erweiterung der digitalen Produktpalette haben agile Entwicklungsansätze inzwischen Einzug in die Automobilindustrie gehalten. Abhängig von der Kritikalität des IT-Systems muss jedoch auch hier hinterfragt werden, ob und inwieweit Gesetze und Richtlinien hinsichtlich der Produktsicherheit zu berücksichtigen sind und agile Entwicklungsmethoden mit diesen kompatibel sind.

Aufgrund dieser Produktvielfalt in der Automobilindustrie kann die Frage, ob und wie IT-Systeme agil entwickelt werden können oder sollten, nicht pauschal beantwortet werden. Jedoch gibt es Branchenstandards, die bei einer Vielzahl von Entwicklungen zu berücksichtigen sind. Hierzu zählen:

- Automotive SPICE
- ISO 26262
- AUTOSAR
- Reifegradabsicherung für Neuteile gemäß VDA

Aufgrund der großen Bedeutung dieser vier Standards in der Automobilindustrie sollen sie im Folgenden kurz dargestellt und ihre Vereinbarkeit mit Agilität diskutiert werden.

3.1. Automotive SPICE

3.1.1. Kurzbeschreibung

Automotive SPICE ist ein Standard der Automobilbranche zur Bewertung der Leistungsfähigkeit der Entwicklungsprozesse bei Herstellern eingebetteter Systeme. Es ist eine domänenspezifische Variante des Standards ISO/IEC 33020 (SPICE). Automotive SPICE definiert den Referenzprozess (*process reference model* – PRM) und das Prozessbewertungsmodell (*process assessment model* – PAM) für die Entwicklung eingebetteter Systeme. Es übernimmt das Bewertungsframework (Reifegrade, Prozessattribute, Bewertungsschema, Prozessreifegradmodell) von SPICE unverändert (s. A-SPICE 2017). Automotive SPICE liegt derzeit in der Version 3.1 vom November 2017 vor.

Automotive SPICE strukturiert die Prozesse gemäß dem V-Modell (A-SPICE 2017, S. 122). Das V-Modell bezieht sich bei Automotive SPICE sowohl auf den übergeordneten Prozess der Systementwicklung als auch auf den dazu untergeordneten Prozess der Softwareentwicklung. Die Entwicklungsprozesse für Hardware und Mechanik werden in Automotive SPICE nicht betrachtet.

Die im PAM definierten Prozess-Performance-Indikatoren (*process performance indicators*), bestehend aus den *Base Practices* (BP) und den *Work Products* (WP), sind lediglich Indikatoren und somit keine verbindlichen Regeln zur Umsetzung von Automotive SPICE.

3.1.2. Automotive SPICE und Agilität

Ein Prozessreifegradmodell wie Automotive SPICE beschreibt lediglich das Was, d. h. die Ziele des Prozesses. Demgegenüber beschreibt ein Vorgehensmodell wie z. B. Scrum das Wie, d. h. mit welchen Methoden, Mitteln und Organisationsstrukturen die Ziele erreicht werden sollen (s. A-SPICE 2017, S. 23). Im *Automotive SPICE Guideline* ist die Vereinbarkeit von Automotive SPICE mit agilen Frameworks daher ausdrücklich festgestellt worden (VDA 2017, Kapitel 2.2.2). Eine Zusammenfassung über Kompatibilität von Automotive SPICE und Agilität findet sich ebenso in dem Whitepaper “Clarifying Myths with Process Maturity Models vs. Agile” (Besemer et al. 2014), in „Wie sag ich’s dem Assessor?“ (Bondar et al. 2015) und in “How Do Agile Practices Support Automotive SPICE Compliance?” (Diebold et al. 2017).

Bezug des V-Modells

Besemer et al. schreiben, dass SPICE und damit auch Automotive SPICE keine Vorgaben machen, in welcher Reihenfolge die einzelnen Ziele zu erreichen sind: “Since SPICE is at the WHAT level it cannot, and does not, predefine any lifecycle model. Consequently, it does not predefine any logical points in time at which work products shall be available, nor does it predefine any other kind of activity sequences or ordering.” (Besemer et al. 2014, S. 6). Dies bedeutet: Weder das V-Modell noch Automotive SPICE legen explizit fest, dass es sich bei dem V-Modell um ein Phasenmodell bezogen auf die gesamte Entwicklungslaufzeit handelt. Das V-Modell kann sich auch auf ein Release, einen Sprint oder die Umsetzung einer einzelnen Anforderung beziehen. Werden während der gesamten Laufzeit viele kleine V-Prozesse durchlaufen, die sich jeweils auf einzelne Anforderung beziehen, so löst sich der grundlegende Konflikt mit der Agilität auf.[9] Dies ist entscheidend, um eine Kompatibilität zwischen Automotive SPICE und Agilität überhaupt zu ermöglichen. Die typische Interpretation des V-Modells als ein Modell, dass sequentiell zu bearbeitenden Phasen bezogen auf ein Projekt oder ein Release vorsieht, wäre mit einem agilen Vorgehen unvereinbar und ist auch nicht in Automotive SPICE gefordert. Der Bezug des V-Modells hingegen auf die Realisierung einzelner Anforderungen ermöglicht agile Entwicklung.

[9] Man könnte das V-Modell auch auf einen kurzen Entwicklungszyklus – dem Sprint gemäß Scrum – beziehen. Dies entspräche einem Mini-Wasserfall-Modell, das jedoch keine echte Agilität darstellt und vermieden werden sollte.

Chronologische Unabhängigkeit der Prozessergebnisse

Eine weitere Voraussetzung dafür, dass Systeme effizient agil entwickelt werden können, ist die chronologische Unabhängigkeit der einzelnen Prozessergebnisse (*process outcomes*) auch innerhalb eines einzelnen Prozesses. Dies ist notwendig, um sowohl Prozessschritte zur Strategieentwicklung von operationalen Aufgaben zeitlich zu entkoppeln als auch Test-Driven-Development zu ermöglichen.

Dazu als Beispiel SWE.4 *Software Unit Verification*: Das Prozessergebnis 1 fordert „a software unit verification strategy including regression strategy […] developed to verify the software units." Im selben Prozess soll die Durchführung der eigentlichen Software-Unit-Verifikation erfolgen. Diese Vermengung von strategischen und operativen Aufgaben innerhalb eines Prozesses suggeriert, dass sich diese Prozessergebnisse zeitnah innerhalb einer Entwicklungsphase erarbeitet werden. Dies ergibt beim klassischen Wasserfallmodell Sinn: Nach Erstellung aller Software-Units wird eine Strategie für deren Verifikation erstellt und dann die Verifikationen aller Software-Units durchgeführt. Bei einer agilen Entwicklung würde das keinen Sinn ergeben. Demnach müsste vor jeder Verifikation einer einzelnen Anforderung eine Strategie dazu entwickelt werden. Tatsächlich sollte das Prozessergebnis 1 so interpretiert werden, dass es einmalig für alle Software-Units vorab erarbeitet wird und dann während der Entwicklung die einzelnen Software-Units fortlaufend entsprechend getestet werden. Diese Argumentation gilt analog u. a. für die Prozessergebnisse SYS.4: 1 und 2, SYS.5: 1 und 2, SWE.5: 1 und 2, SWE.6: 1, SUP.1: 1 und SUP.2: 1.

Zudem sollte beim Prozess SWE.4 das Prozessergebnis 2 („criteria for software unit verification are developed […]") zeitlich von der eigentlichen Verifikation der Software-Unit unabhängig verstanden werden. Im Sinne eines Test-Driven-Developments werden Testfälle **vor** der Realisierung der Software-Unit beschrieben. Dies gilt analog auch für die Prozessergebnisse SYS.4: 3, SWE.5: 3 und SWE.6: 2.

Die in Automotive SPICE definierten Prozesse sind lediglich thematische Bündel. Die zeitliche Reihenfolge der Prozesse oder deren Prozessergebnisse dürfen bei der Bewertung des Reifegrades keine Rolle spielen.

Einzelne Prozessergebnisse mit Konfliktpotential

Neben diesen grundsätzlichen Interpretationen zu Automotive SPICE gibt es einzelne Prozessergebnisse, die einer Klärung bedürfen, damit sie eine agile Systementwicklung nicht behindern. Dazu zählen u. a.:

- ACQ.3, Prozessergebnis 2: "the contract/agreement clearly and unambiguously specifies the expectations, responsibilities, work product/deliverables and liabilities of both the supplier(s) and the acquirer" – Dieses Prozessergebnis ist sicherlich auch für agile Systementwicklung sinnvoll und notwendig, jedoch darf es nicht so interpretiert werden, dass zum Vertragsabschluss ein vollständiges Lastenheft als Vertragsbestandteil erforderlich

ist. Dies würde den agilen Grundsätzen widersprechen. Stattdessen muss eine Vertragsgestaltung, die mit Agilität vereinbar ist, möglich sein (siehe Abschnitt 1.5).

- SYS.3: “The purpose of the System Architectural Design Process is to establish a system architectural design and identify which system requirements are to be allocated to which elements of the system, and to evaluate the system architectural design against defined criteria.” – Das agile Entwicklungsteam bricht die fachlichen Anforderungen erst bei Bedarf in technische Anforderungen herunter und definiert so iterativ die Architektur. Architekturentscheidungen werden eher ad hoc und weniger formal getroffen. Dies gibt analog auch für die Prozesse SWE.2 und SWE.3.

- SWE.3, Prozessergebnis 5: “results of the unit verification are summarized and communicated to all affected parties” – Das Implementieren der Software und deren Unit-Testfälle, das Verifizieren auf Unit-Ebene und das Einchecken des Quellcodes erfolgt bei einer agilen Entwicklung in kurzen Zyklen und eigenständig durch den Entwickler (siehe Abschnitt 2.3.1). Davon betroffen sind allenfalls Teammitglieder, die dann ggf. weitere Tests durchführen. Eine dokumentierte Zusammenfassung und Kommunikation ist nicht zwingend erforderlich. Vielmehr sind im agilen Umfeld die einzelnen Testfälle meist in Form automatisierter Test erfasst und somit einzeln dokumentiert.

- SUP.1, Prozessergebnis 2: “quality assurance is performed independently and objectively without conflicts of interest” – Die Sicherung der Qualität ist Aufgabe des gesamten agilen Teams. Eine dezidierte Rolle für die Qualitätssicherung ist in Scrum und anderen Frameworks nicht vorgesehen. Es ist jedoch gängige Praxis, dass eine andere Person als der Entwickler, der eine Anforderung implementiert hat, das integrierte Ergebnis testet bevor der Product-Owner das Inkrement selbst nochmals begutachtet und freigibt. Die geforderte Unabhängigkeit der Qualitätssicherung bezieht sich jedoch nicht nur auf das Produkt, sondern auch auf die Prozesse. Prozesse werden im agilen Umfeld aus dem gesamten Team heraus, typischerweise durch Retrospektiven, geprüft und verbessert. Eine Unabhängigkeit wäre so allenfalls durch den unabhängigen Agility-Master gewährleistet.

- SUP.9, Prozessergebnis 2: “problems are recorded, uniquely identified and classified” – Automotive SPICE macht keine klare Aussage darüber, um welche Art von Problemen es sich dabei handelt. Wenn es um Softwarefehler von bereits ausgelieferter Software geht, ergibt dieses Prozessergebnis auch im agilen Umfeld Sinn. Wenn es sich jedoch um Bugs handelt, die während der Entwicklung festgestellt werden, so ist der Prozessschritt zu formal und umständlich. Wenn ein Qualitätssicherer einen Fehler feststellt und gleich darauf mit dem Entwickler spricht, der dann den Fehler behebt, sollte keine Dokumentation gefordert werden. Dies ist gängige Praxis in einem agilen Entwicklungsteam, das an einem Standort arbeitet.

- SUP.10: “The purpose of the Change Request Management Process is to ensure that change requests are managed, tracked and implemented.” – Das Anforderungsmanage-

ment im agilen Umfeld geht bereits von einem kontinuierlichen Verändern der Anforderungen aus. Ein Change Request Management ist implizit bereits vorhanden und ein zusätzlicher expliziter Prozess dazu ist im agilen Umfeld nicht erforderlich.

Prozess-Performance-Indikatoren

Automotive SPICE ergänzt das PRM um das PAM mit den Prozess-Performance-Indikatoren *Base Practices* und *Work Products*. Einige dieser Indikatoren sind bei einer agilen Entwicklung jedoch nicht praktikabel und verletzten insbesondere die agilen Grundsätze, Prozesse nicht zu detailliert vorzugeben und die Dokumentation auf den notwendigen Umfang zu beschränken. Beispielsweise wird das detaillierte Design von Software-Units im Rahmen von Scrum-Entwicklungen während der Sprintplanung bzw. während des Sprints von dem Entwicklungsteam diskutiert und realisiert. Eine umfassende Dokumentation, wie sie in den Work Products von Prozess SWE.3 beschrieben ist – z. B. *review record*, in dem u. a. die Gutachter gelistet und der Status des Reviews und die für das Review benötigte Zeit angegeben werden (s. A-SPICE 2017, S. 109) – wäre im agilen Umfeld völlig unangemessen. Jedoch handelt es sich bei den Prozess-Performance-Indikatoren lediglich um Indikatoren und keine verbindlichen Kriterien zur Bewertung der Prozessreife: "Therefore, work product & characteristics are neither a 'strict must' nor are they normative for organizations. Instead, the actual structure, form and content of work products and documents must be decided by the organization ensuring that it is appropriate for the intended purpose and needs of its projects and product development goals." (Besemer et al. 2014, S. 7).

Dennoch muss kritisch hinterfragt werden, ob diese Indikatoren, auch wenn sie formal unverbindlich sind, **in der Praxis** nicht doch einen erheblichen Einfluss auf die Bewertung der Prozessreife haben. So dürfte ein Entwicklungsverantwortlicher, der Indikatoren ignoriert und die Prozessergebnis auf anderem Wege zu erreichen versucht, im Falle eines Scheiterns des Entwicklungsprojektes sich für diese Entscheidung rechtfertigen müssen; bei penibler Erhaltung aller Base Practices und Erstellung aller Work Products dürfte ein Scheitern hingegen eher als unvermeidlich akzeptiert werden. Ebenso kann ein Unternehmen kaum sicher sein, dass ein Assessor, so wie von SPICE eigentlich gefordert, tatsächlich nur das PRM und nicht die Indikatoren als verbindlichen Maßstab nimmt.

Daher wäre es wünschenswert, wenn Base Practices und Work Products um Erläuterungen ergänzt werden würden, die deren Geltungsbereich und damit die Kompatibilität mit agilen Methoden besser verdeutlichen. Hierzu gibt es bereits erste Ansätze: Diebold et al. haben in ihrem Paper "How Do Agile Practices Support Automotive SPICE Compliance?" (Diebold et al. 2017) ein Mapping von agilen Praktiken auf Base Practices und Work Products vorgenommen und konnten damit 96 % der Base Practices und 87 % der Work Practices abdecken.

Reifegrade

SPICE definiert ein Bewertungsframework (*measurement framework*) mit Reifegradstufen (*capability levels*) von CL 0 bis CL 5. Vorausgesetzt, das Prozessreferenzmodell von Automotive

SPICE ließe sich – wie oben beschrieben – im Sinne einer agilen Entwicklung interpretieren, so stellt sich dennoch die Frage, welche Reifegradstufen agile Teams erreichen können.

Die Stufe CL 0 (*incomplete process*) bedeutet, dass der Prozess nicht existiert bzw. die geforderten Prozessergebnisse nicht erzielt werden.

Für die Stufe CL 1 (*performed process*) müssen die Prozessergebnisse erreicht werden, egal wie. Agile Teams können unter den oben genannten Voraussetzungen CL 1 sicherlich erreichen.

CL 2 (*managed process*) fordert, dass der Prozess geplant, beobachtet und gesteuert wird. Automotive SPICE verlangt als Minimum, dass die Prozessleistungsziele Ressourcen, Aufwand und Zeit gemanagt werden. Dies kann grundsätzlich auch agil erfolgen. Agile Frameworks wie Scrum definieren explizit dazu Rituale, Artefakte und Rollen. Jedoch bedarf es ggf. ergänzender Maßnahmen zur Sicherstellung der Traceability von Work Products (A-SPICE 2017, PA 2.2.). Klassischerweise arbeiten viele agile Teams mit Karteikarten, die sie an Wänden befestigen. Traceability ist so nicht gegeben. Je nach Work Product muss im Einzelfall geklärt werden, ob das Arbeitsergebnis dokumentiert vorliegen bzw. Traceability gewährleistet werden muss. Der Einsatz von Softwaresystemen für das Anforderungs- und Versionsmanagement, die Traceability unterstützen, sind jedoch auch im agilen Umfeld üblich. Agile Entwicklungsteams erstellen typischerweise viele Unit-Testfälle, die auf Anforderungen referenzieren und zusammen mit dem Quellcode in einem Versionsmanagement verwaltet werden. Die geforderte Traceability sollte daher kein grundsätzliches Hindernis für Agilität darstellen.

CL 3 (*established process*) fordert einen Standardprozess für die Entwicklung einer Produktart, der für alle Projekte und Teams gültig ist. Sofern sich dieser geforderte Prozessstandard auch auf die Arbeit innerhalb des agilen Teams bezieht, ist dies ein fundamental anderer Ansatz als bei der agilen Softwareentwicklung und damit inkompatibel mit den agilen Grundsätzen. Zwar gibt es agile Frameworks wie Scrum oder Extreme Programming, die einen prozessualen Rahmen definieren; Ziel ist es jedoch, dass jedes Entwicklungsteam in Abstimmung mit Organisationseinheiten, zu denen es Schnittstellen hat, einen für sich sinnvollen Prozess definiert und kontinuierlich anpasst. CL 3 fordert hingegen einen zentral gemanagten Prozess, der allgemeingültig ist und lediglich durch Tailoring angepasst werden kann. In einem agilen Umfeld kann CL 3 nur dann erreicht werden, wenn der geforderte Standardprozess sehr allgemeingültig ist und primär auf die Schnittstellen zwischen Teams und Organisationseinheiten abzielt. Andernfalls wäre die bei der agilen Entwicklung so wichtige Selbstorganisation nicht umsetzbar.

CL 4 (*predictable process*) fordert den Aufbau eines Kennzahlensystems, um quantitativ bewerten zu können, inwieweit durch den Entwicklungsprozess die Geschäftsziele erreicht werden und welche Optimierungspotentiale vorhanden sind. Ein solcher Fokus auf die geschäftlichen Ziele ist durchaus im Sinne einer agilen Entwicklung (s. dazu auch Abschnitt 1.6). Ebenso unproblematisch sind Kennzahlen, die den Umfang der umgesetzten Anforderungen messen (bei Scrum z. B. die sog. Velocity). Problematisch wäre es jedoch, wenn Zwischenergebnisse und einzelne Kompetenzbereiche innerhalb eines agilen Teams gemessen und bewertet werden. Im agilen Umfeld sollten sich die Kennzahlen auf die erstellte Software, deren Nutzung im Markt und den

Aufwänden und Durchlaufzeiten des gesamten Teams beschränken. Ein detailliertes Controlling ins Team hinein sollte vermieden werden.

Die für die letzte Reifegradstufe CL 5 (*innovating process*) neu hinzugekommenen Prozessanforderungen passen mit den agilen Grundsätzen und Prinzipien sehr gut zusammen. Agilität beinhaltet eine kontinuierliche Bewertung und Anpassung der Entwicklungsprozesse durch das Team. Aufgrund der hohen Selbstorganisation des agilen Teams können Ideen für Produkt- und Prozessverbesserungen ohne großen organisatorischen Aufwand eingeführt und getestet werden.

3.1.3. Fazit

Um Systeme sowohl gemäß Automotive SPICE als auch gemäß den agilen Grundsätzen und Prinzipien zu entwickeln, muss Automotive SPICE anders interpretiert werden als für klassische Entwicklungsprojekte üblich. Dies betrifft insbesondere folgende Punkte:

- Das V-Modell bezieht sich auf Einzelanforderungen, nicht auf das Gesamtprojekt.
- Prozessbereiche und deren Ergebnisse müssen als chronologisch unabhängig betrachtet werden. Strategische Aufgaben sind somit auch chronologisch unabhängig von operativen Aufgaben.
- Als ein unabhängiges Qualitätsmanagement müssen auch qualitätssichernde Maßnahmen innerhalb des agilen Teams, wie z. B. gegenseitige Reviews, Pair Programming oder Retrospektiven, zugelassen werden. Ein externes Qualitätsmanagement darf nicht in die internen Prozesse des agilen Teams eingreifen.
- Prozess-Performance-Indikatoren sind lediglich – ganz im Sinne von SPICE – als Indikatoren und nicht als verbindlicher Maßstab zur Prozessbewertung zu verstehen.

Hinsichtlich der Reifegradstufen ist die Erlangung des CL 3 im agilen Umfeld problematisch, da beim agilen Ansatz gerade auf allgemeingültige Prozessstandards bewusst verzichtet wird und stattdessen das agile Team innerhalb des agilen Frameworks eigene Prozess und Standards entwickelt und kontinuierlich anpasst.

Die Frage ist nicht so sehr: Ist Agilität mit Automotive SPICE vereinbar? (Die Antwort lautet, grundsätzlich ja – zumindest bis Reifegradstufe 2.) Sondern die entscheidende Frage lautet: Wird die Interpretation von Automotive SPICE, wie sie für agile Entwicklung erforderlich ist, von den Assessoren akzeptiert? Mit dem *International Assessor Certification Scheme* (INTACS) gibt es einen wichtigen Befürworter der Vereinbarkeit von Agilität mit Automotive SPICE, sodass von einer grundsätzlichen Bereitschaft, agile Methoden als Ersatz von klassischen Projektmanagementmethoden zu akzeptieren, auszugehen ist.

3.2. Funktionale Sicherheit (ISO 26262)

3.2.1. Kurzbeschreibung

Die ISO 26262 (“Road vehicles – Functional safety”) ist ein internationaler Standard für sicherheitsrelevante elektrische und elektronische Systeme in Fahrzeugen. Die ISO 26262 ist im November 2011 in Kraft getreten. Die folgenden Betrachtungen beziehen sich auf den Entwurfsstand von 2018 (ISO/FDIS 2018).

Ziel der ISO 26262 ist es, Anforderungen und Prozesse zu definieren, um systematische Fehler und zufällige Hardwarefehler zu beherrschen:

“[…]

a) provides a reference for the automotive safety lifecycle and supports the tailoring of the activities to be performed during the lifecycle phases, i.e., development, production, operation, service, and decommissioning;

b) provides an automotive-specific risk-based approach to determine integrity levels [Automotive Safety Integrity Levels (ASIL)];

c) uses ASILs to specify which of the requirements of ISO 26262 are applicable to avoid unreasonable residual risk;

d) provides requirements for functional safety management, verification, validation and confirmation measures; and

provides requirements for relations with suppliers.” (ISO/FDIS 2018:8)

Die ISO 26262 basiert genau wie Automotive SPICE auf dem V-Modell. Im Gegensatz zu Automotive SPICE wird neben der Softwareentwicklung die Hardwareentwicklung als Teil der Systementwicklung explizit mit adressiert; die Entwicklung der Mechanik bleibt wie bei Automotive SPICE unberücksichtigt. Die ISO 26262 beschreibt den Entwicklungsprozess entlang des V-Modells in den Teilen 3 (“Concept phase”), 4 (“Product development at the system level”), 5 (“Product development at the hardware level”) und 6 (“Product development at the software level”). Der Teil 7 (“Production, operation, service and decommissioning”) befasst sich mit dem Produktlebenszyklus nach der Entwicklung. Die Teile 3 bis 7 beschreiben Prozessvorgaben[10] zur Sicherung der Qualität. Die dort beschriebenen einzelnen Prozessvorgaben sind, insbesondere im Teil 6, meistens weder spezifisch für die Automobilbranche noch für sicherheitskritische Systeme.

[10] Hier und im Folgenden sollen unter *Prozessvorgaben* alle Aktivitäten zur Umsetzung der funktionalen Sicherheitsanforderungen verstanden werden. Dazu gehören Maßnahmen zur Gewährleistung der funktionalen Sicherheit, aber auch Aktivitäten mit Bezug zur Sicherheitskultur im Unternehmen.

Vielmehr handelt es sich im Allgemeinen um Vorgaben, die auch in anderen Branchen und für sicherheitsunkritische Systeme sinnvoll angewendet werden können und angewendet werden.

Zu einer ISO 26262 konformen Entwicklung gehören drei wesentliche Merkmale:

1. Die **Erzeugung von Artefakten**[11], die den Nachweis der funktionalen Sicherheit ermöglichen. Diese Artefakte bauen teilweise aufeinander auf.
2. Die **Prüfung dieser Artefakte** zur Sicherstellung, dass die Grundlage für den abschließenden Sicherheitsnachweis korrekt ist.
3. Das **Einhalten von Prozessvorgaben** während der Entwicklung je nach Einstufung des zu entwickelnden Systems.

Herzstück der Artefakte ist die Gefahrenanalyse und Risikobewertung (*hazard analysis and risk assessment – HARA*), die vor der Systementwicklung durchzuführen ist. Das Ergebnis der HARA ist die Klassifizierung der Sicherheitskritikalität des zu entwickelnden Systems in eine der *Automotive Safety Integrity Levels* (*ASIL*). Je nach Klassifizierung in QM bzw. ASIL A bis ASIL D ergeben sich dann unterschiedliche Anforderungen hinsichtlich der in den Teilen 3 bis 7 beschriebenen qualitätssichernden Maßnahmen zur Verifizierung bzw. Validierung des Produkts und anderen Artefakten. Zudem sollen bei der Systementwicklung Sicherheitsanalysen z. B. in Form einer Fehlermöglichkeits- und -einflussanalyse (*Failure Mode and Effects Analysis – FMEA*) durchgeführt werden (ISO/FDIS-9:2018, Abschnitt 8).

Die ISO 26262 legt besonderen Wert auf die Prüfung der Artefakte (z. B. der HARA selbst). Diese Prüfungen werden in der ISO 26262 als Bestätigungsmaßnahmen (*confirmation measures*) bezeichnet. In Teil 2 der ISO 26262 sind in Tabelle 1 die erforderlichen Bestätigungsmaßnahmen aufgelistet und die Anforderungen zu diesen Prüfungen beschrieben. Ein entscheidender Punkt dabei ist das Verhältnis des Prüfers zu demjenigen, der verantwortlich ist für die Erstellung des Artefakts. Dabei unterscheidet die ISO 26262 zwischen vier Unabhängigkeitsgraden:

Unabhängigkeitsgrad	Anforderung an Prüfung
I0	Eine Prüfung **sollte** durchgeführt werden; wenn geprüft wird, dann muss die Prüfung von einer **anderen Person als dem/den Ersteller(n)** durchgeführt werden.
I1	Eine Prüfung **muss** von einer **anderen Person als dem/den Ersteller(n)** durchgeführt werden.
I2	Prüfung **muss** von einer **Person mit einem anderen direkten Vorgesetzten** als dem direkten Vorgesetzten des/der Ersteller/s durchgeführt werden.

[11] Hier und im folgenden gesamten Abschnitt zur funktionalen Sicherheit sind mit *Artefakten* immer die in der ISO 26262 geforderten Dokumente, Produkte oder sonstigen Arbeitsergebnisse gemeint.

I3	Prüfung **muss** von einer **Person aus einem anderen Bereich bzw. einer anderen Organisation** als der Bereich/Organisation, die die Erstellung verantwortet, durchgeführt werden.

Höhere ASIL haben dieselben oder höhere Anforderungen hinsichtlich der Unabhängigkeit als niedrigere ASIL. Es gibt Artefakte, die in allen ASIL-Einstufungen mit dem höchsten Unabhängigkeitsgrad geprüft werden müssen (z. B. die HARA).

Abhängig vom in der HARA definierten ASIL, müssen bei der Entwicklung eines Systems bestimmte Prozessvorgaben berücksichtigt werden. Prozessvorgaben existieren für alle Entwicklungsphasen (Entwicklung des Gesamtsystems, Entwicklung der Hardware, Entwicklung der Software).

Die ISO 26262 definiert u. a. folgende Rollen und deren Verantwortlichkeiten:

- Der *Project-Manager* ist dafür verantwortlich, dass die Sicherheitsziele für das Gesamtprojekt erreicht und die dafür benötigten Ressourcen bereitgestellt werden.
- Der *Safety-Manager* verantwortet die Planung und Durchführung der Sicherheitsaktivitäten sowie die Einhaltung und Anpassung des Sicherheitsplans (ISO/FDIS 26262-2:2018, 6.4.6). Er kann Verantwortlichkeiten delegieren, sofern diese klar zugewiesen und kommuniziert sind (ISO/FDIS 26262-2:2018, 6.4.6). Ein Projekt-Manager kann gleichzeitig Safety-Manager sein (ISO/FDIS 26262-2:2018, 6.4.2.4 Note 1).

3.2.2. ISO 26262 und Agilität

Die ISO 26262 erlaubt explizit der Verwendung agiler Methoden auch für die Entwicklung sicherheitskritische Software (ISO/FID 26262-6:2018, 5.2, Note 1), sofern die geforderte Dokumentationspflicht und die Durchführung der geforderten Sicherheitsprozesse sichergestellt sind. Daraus ergeben sich vier grundsätzliche Konfliktpotentiale zwischen der Norm und Agilität:

Vorgehensmodell

Ähnlich wie Automotive SPICE liegt der ISO 26262 ein V-Modell zugrunde. Im Gegensatz zu Automotive SPICE gibt die ISO 26262 eine zeitlich-logische Reihenfolge der Prozessschritte vor, indem Voraussetzungen für den Beginn eines Prozessschritts gefordert sind, die durch andere Prozessschritte zuvor erarbeitet sein müssen. Jedoch kann auch hier argumentiert werden, dass sich das V-Modell nicht auf die gesamte Entwicklungszeit, sondern auf kleinere Einheiten bis hin zur Umsetzung einzelner Anforderungen bezieht. Somit gilt grundsätzlich dieselbe Argumentation hinsichtlich der Vereinbarkeit von V-Modell und Agilität wie bei Automotive SPICE (s. Abschnitt 3.1.2).

Gleichwohl gibt es sicherheitsrelevante Aktivitäten, die gemäß der ISO 26262 zumindest zu Beginn eines Projektes durchgeführt werden müssen. Hierzu zählt insbesondere die HARA. Je nach Art der Änderungen, die sich im Laufe der Entwicklung ergeben, muss die HARA öfters wiederholt und die ASIL ggf. geändert werden, oder aber die HARA wird zu Beginn nur einmalig durchgeführt, wenn die Änderungen nur geringe Auswirkungen auf die funktionale Sicherheit haben. Unabhängig davon sollte die (erste) HARA in einer initialen Konzeptphase durchgeführt werden.

Dokumentation

Die ISO 26262 fordert die Erstellung einer Reihe von Dokumenten als Nachweis für die konforme Entwicklung gemäß der Norm. Die Erstellung dieser Dokumente durch das Team können als Anforderungen an das zu erstellende Produkt verstanden werden (siehe dazu auch Abschnitt 2.6). Das Team kann diese Randbedingungen in Form einer Definition-of-Ready, von Product-Backlog-Items und einer Definition-of-Done in den agilen Entwicklungsprozess einsteuern. Daher stellt der geforderte Dokumentationsumfang prozessual kein Widerspruch zur Agilität dar, auch wenn damit ein erhöhter Pflege- und Änderungsaufwand verbunden ist.

Dennoch stehen die in der Norm geforderten Dokumente im Widerspruch zum agilen Manifest, demnach eine funktionierende Software mehr zu schätzen ist als eine umfassende Dokumentation: Die Dokumente gemäß der Norm zielen primär auf den Nachweis der Produktqualität ab, sind also nicht Teil des Produktes selbst. Je umfangreicher der Dokumentationsaufwand (in Abhängigkeit von dem ASIL), desto mehr steht die ISO 26262 hinsichtlich der Dokumentationsvorgaben im Widerspruch zur Agilität und behindert so die agile Entwicklung.

Methodenkompetenz- und -einsatz

Die ISO 26262 beschreibt in den Teilen 3 bis 6 Methoden zur Prüfung der funktionalen Sicherheit und empfiehlt diese Methoden je nach ASIL unterschiedlich dringend. Die Methoden unterscheiden sich im Allgemeinen nicht von anderen gängigen Methoden zur Verifizierung und Validierung von Software und sind auch bei einer agilen Entwicklung anwendbar und üblich. Gute Softwareentwickler und Software-Qualitätssicherer sollten die in der ISO 26262 empfohlenen Prüfmethoden kennen und beherrschen, unabhängig davon, ob sie in einem klassischen oder in einem agilen Umfeld arbeiten.

Für ein agiles Arbeiten ist jedoch kritisch zu bewerten, dass diese Methoden von außen durch die Norm vorgeschrieben sind. Entsprechend den agilen Grundsätzen und Prinzipien sollte das Entwicklungsteam vielmehr selbst darüber entscheiden können, welche Methoden es in welchem Umfang anwendet.

Das agile Team muss die von der ISO 26262 geforderten Artefakte erstellen und Indikatoren z. B. für eine notwendige Anpassung des Sicherheitsplans oder für eine erneute Durchführung der HARA erkennen können. Diese Fähigkeiten müssen der Norm entsprechend nachgewiesen werden. Zudem müssen die ISO 26262-Rollen *Project-Manager* und *Safety-Manager* im agilen Team verankert werden. So könnten beispielsweise der Product-Owner die Rolle des Project-Managers

und der Agility-Master die Rolle des Security-Managers wahrnehmen. Diese Anforderungen sind grundsätzlich mit einem agilen Vorgehen vereinbaren, erfordern jedoch explizites und nachweisbares Know-how bezüglich funktionaler Sicherheit im agilen Team. Ggf. müssen Experten der funktionalen Sicherheit das Team, auch den Product-Owner oder den Agility-Master, bei der Wahrnehmung dieser Aufgaben unterstützen.

Unabhängigkeit der Bestätigungsmaßnahmen

Die ISO 26262 fordert, dass Bestätigungsmaßnahmen (*conformation measures*) durchzuführen und hierfür je nach ASIL unterschiedliche Anforderungen hinsichtlich der organisatorischen Unabhängigkeit der Prüfer zu erfüllen sind (siehe ISO/FDIS 26262-2:2018, Tabelle 1). Demgegenüber steht das agile Prinzip, dass das Entwicklungsteam die auslieferbaren Inkremente eigenverantwortlich und selbstorganisiert entwickelt.

Die entscheidende Frage ist, ob der Prüfer, d. h. der für die Bestätigungsmaßnahmen verantwortliche Person, Teil des Entwicklungsteams ist oder nicht. Ist dies der Fall, wäre selbst dann eine explizite Rolle innerhalb des agilen Teams definiert, was eine Verletzung des agilen Prinzips der gemeinschaftlichen Verantwortung des Entwicklungsteams für das Produkt inkl. der funktionalen Sicherheit darstellen würden. Falls der Prüfer jedoch nicht Teil des Teams sein darf, verschärft sich der Wiederspruch zwischen Norm und Agilität sogar noch weiter.

Je sicherheitskritischer das zu entwickelnde Produkt ist (ASIL-Einstufung), umso höher sind im Allgemeinen die Anforderungen an die Unabhängigkeit der Prüfer und umso schwerer dürfte in vielen Organisationen eine kontinuierliche Einbindung der unabhängigen Prüfer in den agilen Entwicklungsprozess sein.

Die ISO 26262 verfolgt mit der unabhängigen Prüfung der sicherheitsrelevanten Artefakte eine grundsätzlich andere Strategie als der Ansatz der Selbstorganisation bei einer agilen Entwicklung: Die Norm setzt auf eine unabhängige Kontrolle, Agilität hingegen auf die Eigenverantwortlichkeit des Entwicklungsteams. Es kann und soll an dieser Stelle nicht entschieden werden, welcher dieser beiden Ansätze zielführender ist und die funktionale Sicherheit besser absichert. Gute Argumente gibt es für beide Ansätze (für den selbstorganisatorischen Ansatz siehe hierzu auch Abschnitt 2.2.2).

Fazit

Eine Aussage über die Vereinbarkeit der ISO 26262 mit einer agilen Entwicklung kann nicht pauschal getroffen werden. Ein mögliches Szenario, wie eine solche Integration der ISO 26262 in ein agiles Team aussehen könnte, ist von Prof. Vogelsang der TU Berlin im Anhang B beschrieben.

Sowohl hinsichtlich des Dokumentationsaufwands und der methodischen Vorgaben an das Entwicklungsteam als auch der organisatorischen Unabhängigkeit von Erstellern und Prüfern sicherheitsrelevanter Artefakte unterscheiden sich die Ansätze der ISO 262626 von denen im agilen Umfeld. Wie gut sich die ISO 26262 mit Agilität dennoch vereinbaren lässt, hängt u. a. auch davon ab,

- wie eng Entwicklungsteam und Prüfer der sicherheitsrelevanten Artefakte in der Organisation zusammenarbeiten können und wollen,
- wie effizient das Entwicklungsteam die erforderliche Dokumentation erstellen kann,
- in welchem Umfang Kompetenzen bezüglich funktionaler Sicherheit im agilen Team vorhanden sind und
- wie sicherheitskritisch das zu entwickelnde Produkt ist.

Unabhängig davon dürfte eine initiale Konzeptphase, in der eine (erste) HARA für das System erarbeitet wird, auch bei einer agilen Entwicklung unumgänglich und sinnvoll sein.

3.3. AUTOSAR

3.3.1. Kurzbeschreibung

Die 2003 von Unternehmen der Automobilindustrie gegründete Entwicklungskooperation AUTOSAR (**AUT**omotive **O**pen **S**ystem **AR**chitecture) hat eine standardisierte Softwarearchitektur für ECUs festgelegt. Diese Architektur sieht eine konsequente Trennung von hardwareunabhängigen Softwarekomponenten (SWCs) und der durch Basissoftware (BSW) realisierten Infrastruktur und Dienste vor. Durch diese Entkopplung können Softwareanwendungen unabhängig von der darunterliegenden Hardware implementiert und getestet werden. Die AUTOSAR-Laufzeitumgebung (RTE) steuert die Verbindung der SWCs zur Basissoftware. Der Datenaustausch zwischen SWCs erfolgt mithilfe des Virtual Functional Bus (VFB).

Das gesamte System wird durch eine AUTOSAR XML-Datei beschrieben, die die SWCs beschreibt und deren Verteilung auf die einzelnen Steuergeräte und die Netzwerkkommunikation festlegt. Aus dieser Gesamtbeschreibung können *ECU Extract of System Descriptions* generiert werden, die dann als Basis für die Entwicklung einzelner ECUs dienen. Der Austausch der Systembeschreibungen zwischen den Entwicklungspartner kann daher im Wesentlichen über XML-Dateien erfolgen.

3.3.2. AUTOSAR und Agilität

Gemäß den agilen Grundsätzen und Prinzipien obliegt es dem Entwicklungsteam zu entscheiden, wie es die fachlichen Anforderungen umsetzt. AUTOSAR als eine technische Plattform und Referenzarchitektur ist grundsätzlich zunächst nur eine Möglichkeit der Umsetzung. Sollte die Entwicklung der gesamten Software autonom durch ein einziges Entwicklungsteam durchgeführt

werden – was in der Automobilindustrie untypisch wäre – so sollte das agile Team eigenständig darüber entscheiden können, ob die Entwicklung auf AUTOSAR basieren soll oder nicht.

Tatsächlich dürften die allermeisten Entwicklungen über verschiedene Teams verteilt erfolgen und AUTOSAR als Referenzarchitektur ggf. vorgegeben sein. In diesem Fall stellen die sich aus AUTOSAR ergebenen technischen Anforderungen Randbedingungen dar, die als Anforderungen in den Entwicklungsprozess einfließen und zu berücksichtigen sind. Nur so können die durch AUTOSAR bereitgestellten Dienste und die Infrastruktur genutzt werden. Auch dies stellt kein grundsätzliches Hindernis für ein agiles Entwickeln der Software dar.

Die Beschreibung von Anforderungen entsprechend der AUTOSAR-XML-Dateien steht ebenfalls nicht im Widerspruch zur agilen Entwicklung. Diese XML-Dateien können Teil der Product-Backlog-Items sein bzw. von den User-Stories referenziert werden.

3.3.3. Fazit

AUTOSAR beschreibt ein technisches Framework für die Entwicklung von Steuergeräten in der Automobilindustrie, kein Entwicklungsmodell und -vorgehen. Dieses Framework kann unabhängig vom Entwicklungsparadigma verwendet werden. Daher stehen AUTOSAR und Agilität in keinem Widerspruch zueinander.

3.4. Reifegradabsicherung für Neuteile gemäß VDA

3.4.1. Kurzbeschreibung

Der VDA-Blaugold-Band *Produktentstehung – Reifegradabsicherung für Neuteile* definiert Messkriterien für insgesamt acht Reifegradbewertungen (VDA 2009). Diese acht Reifegradbewertungen bauen sequentiell aufeinander auf und legen somit ein Phasenmodell für die Entwicklung von Neuteilen fest. Je nach Klassifizierung des zu entwickelnden Neuteils in eine der Reifegradrisiken A, B oder C, werden unterschiedliche Prozesse für die Zusammenarbeit zwischen dem Kunden und dem Lieferanten zur Bewertung und Durchsprache beschrieben: angefangen von der Ergebnisübermittlung der Selbstbewertung durch den Lieferanten an den Kunden (Risikograd C), über die Präsentation der Selbstbewertungsergebnisse bei einem Treffen (Risikograd B), bis hin zur gemeinsamen Reifegradbewertung durch Kunden und Lieferanten an einem runden Tisch (Risikograd A).

Auch wenn der Standard den Anspruch erhebt, auch für die Entwicklung von Software zu gelten (VDA 2009, S. 16), so ist sein Ursprung in der Entwicklung mechanischer Bauteile unverkennbar,

z. B. anhand der Forderungen nach rechtzeitiger Verfügbarkeit der Produktionsstätten, Konzepte für die Verpackung etc.

3.4.2. Reifegradabsicherung und Agilität

Die vom VDA vorgeschlagene Reifegradabsicherung für Neuteile beschreibt ein Phasenmodell für die Produktentwicklung und fordert damit, übertragen auf die Softwareentwicklung, ein Wasserfallmodell. Dieser Ansatz ist daher mit Agilität grundsätzlich nicht vereinbar.

Die erste Reifegradbewertung RG0 *Innovationsfreigabe für Serienentwicklung* dient der Mittelfreigabe beim Kunden und ist noch unabhängig von der Entwicklungsmethode. Bereits RG1 *Anforderungsmanagement für Vergabeumfänge* beinhaltet die Freigabe eines Lastenheftes (VDA 2009, S. 44). Dies könnte für agile Entwicklung ggf. noch uminterpretiert und als initiales Product-Backlog verstanden werden. Beim RG2 *Festlegung der Lieferkette und Vergabe der Umfänge* wird jedoch die Vollständigkeit des Lastenheftes (VDA 2009, S. 50) und beim RG3 *Freigabe technische Spezifikationen* ein vollständiges Pflichtenheft (VDA 2009, S. 62) gefordert. Spätestens beim RG3 ist der Reifegradprozess selbst durch großzügige Uminterpretationen der geforderten Artefakte nicht mit agilen Grundsätzen zu vereinbaren. Die Bewertungen RG4 *Produktionsplanung abgeschlossen* und RG5 *Serienwerkzeugfallende Teile und Serienanlagen verfügbar* stellen keine grundsätzlichen Widersprüche zur agilen Entwicklung dar. Die letzten beiden Bewertungen RG6 *Produkt- und Prozessfreigabe* und RG7 *Projektabschluss, Verantwortungsübergabe an Serie, Start Requalifikation* spiegeln den Projektcharakter mit klarem Übergabezeitpunkt des Endergebnisses wider. Dies ist im agilen Umfeld eher untypisch, behindert eine agile Entwicklung jedoch nicht grundlegend und ist im Kontext der Gesamtfahrzeugentwicklung erforderlich.

Die durch den VDA beschriebenen Reifegradbewertungen sind in vielen Unternehmen der Automobilindustrie wichtiges Instrument zur Bewertung des gesamten Entwicklungsstandes bis zum *Start of Production* (SOP). Um eine agile Entwicklung von Neuteilen in eine klassische Betrachtung der Gesamtentwicklung nach dem oben beschriebenen Phasenmodell zu ermöglichen, müsste das Phasenmodell an einigen Stellen verändert werden. Eine Lösung könnte beispielsweise sein, dass für agile Entwicklungen

- statt eines Lastenheftes ein initiales Product-Backlog mit allen zum Zeitpunkt der Erstellung bekannten, zwingend erforderlichen Anforderungen erstellt sein muss (betrifft die Reifegradbewertungen RG1 und RG2),
- ein Pflichtenheft nicht gefordert wird, d. h. agil entwickelte Neuteile bei der Reifegradbewertung RG3 nicht berücksichtigt werden,
- die Reifegradbewertung RG4 ein Konzept für DevOps und deren Umsetzung fordert, nicht jedoch die begonnene oder gar abgeschlossene Entwicklung des Neuteils, und

- für die Reifegradbewertung RG5 lediglich zwingend erforderliche Anforderungen umgesetzt und auslieferbar sein müssen und die erfolgreiche Integration in das Gesamtsystem nachgewiesen werden muss.

Durch eine solche Veränderung würde ein Rahmen geschaffen werden, in dem agile entwickelte Neuteile bei der Gesamtbewertung weitestgehend mitberücksichtigt werden, ohne die agilen Grundsätze und Prinzipien wesentlich zu verletzen.

3.4.3. Fazit

Die Reifegradabsicherung für Neuteile gemäß VDA eignet sich gut für die Vorbereitung der Serienproduktion mechanischer Teile, nur bedingt für die Softwareentwicklung, nicht für agile Entwicklung und erst recht nicht für agile Softwareentwicklung. Insbesondere die Forderungen für den Reifegrad 3 *Freigabe technische Spezifikationen* ist mit Agilität unvereinbar. Agil entwickelte Umfänge sollte bei der Reifegradbewertung RG3 unberücksichtigt bleiben.

4. QM AGILER SW-ENTWICKLUNG IN DER AUTOMOBILINDUSTRIE

Im Folgenden werden die Herausforderungen und Lösungsansätze für das Qualitätsmanagement in der Automobilindustrie bei der Einführung agiler Entwicklungsmethoden für digitale Produkte dargestellt. Dabei wird sowohl das gesamte Spektrum möglicher digitaler Produkte in der Branche im Allgemeinen als auch branchentypische Entwicklungen wie z. B. sicherheitskritische Steuergeräte im Besonderen betrachtet. Ausgangspunkt dabei sind die bekannten Aufgabenfelder des Qualitätsmanagements:

1. **Q-Anforderungen definieren**: Qualitätsanforderungen hinsichtlich des Produkts und der Methoden erfassen, formulieren und in die Entwicklung einsteuern
2. **Lieferanten bewerten**: Lieferanten auditieren und deren Fähigkeit zur Umsetzung der Anforderungen prüfen
3. **Reifegrad absichern**: Produkt- und Prozessqualität verifizieren und validieren
4. **Produkte freigeben**: Produkt final prüfen und zur Auslieferung freigeben
5. **Feld beobachten**: Produktnutzung durch den Kunden und Beanstandungen erfassen
6. **Lessons Learned ableiten**: Erkenntnisse aus der Feldbeobachtung und dem Entwicklungs- und Produktionsprozess erfassen und in die Entwicklung einsteuern

Für jedes dieser Aufgabenfelder wird gezeigt, wie das Qualitätsmanagement bei einer agilen Entwicklung digitaler Produkte sinnvoll gestaltet werden kann.

4.1. Q-Anforderungen definieren und einbringen

4.1.1. Produktklassifizierung und Q-Anforderungen

Je nach Art des Produktes müssen sowohl unterschiedliche fachliche Qualitätsanforderungen umgesetzt als auch organisatorische und rechtliche Randbedingungen und Branchenstandards berücksichtigt werden. Als erstes sollte daher vor Beginn der Entwicklung Klarheit darüber geschaffen werden, zu welcher Klasse das zu entwickelnde Produkt gehört und welche Anforderungen mit der Produktklasse verbunden sind.

Beispiel Parkassistentsystem[12]

Es soll ein neues Parkassistentsystem entwickelt werden, welches auch in komplexeren Parksituationen (Garagen, Parkhäuser etc.) ein automatisiertes Einparken ermöglicht. Hieraus ergeben sich eine Reihe fachlicher Qualitätsanforderungen, z. B. hinsichtlich der Wartbarkeit eines solchen Systems durch Werkstätten und der Ausfallsicherheit. Viele dieser Anforderungen sind bereits aus vorhergehenden Entwicklungen von Parkassistentsystemen bekannt und können für die Neuentwicklung übernommen werden. Das Parkassistentsystem ist ein eingebettetes System, das über den CAN-Bus mit anderen Steuergeräten im Fahrzeug kommunizieren und auf der AUTOSAR-Plattform und dessen Architektur basieren soll. Das System muss entsprechend den Vorgaben von Automotive SPICE realisiert werden. Es handelt sich dabei zudem um ein sicherheitskritisches System, das daher nach ISO 26262 zu entwickeln ist. Es wird hier angenommen, dass die Gefahren- und Risikoanalyse ASIL D, also die höchste sicherheitskritische Stufe, gilt[13].

Beispiel Paketzustellung

Paketzustellern soll es zukünftig möglich sein, Pakete in dem Kofferraum von Fahrzeugen zuzustellen. Hierzu müssen mehrere Systeme entwickelt bzw. angepasst werden:

1. In Onlineshops können Kunden das amtliche Kennzeichen ihres Fahrzeugs als Lieferadresse, das zweistündige Zeitfenster für die gewünschte Zustellung und die Position des Fahrzeugs während dieses Zeitfensters eingeben.

[12] Die in diesem Kapitel beschriebenen Fallbeispiele *Parkassistentsystem*, *Paketzustellung* und *Ladesäulensuche* sind rein exemplarisch und erheben kein Anspruch auf Allgemeingültigkeit für Systeme dieser Art. Je nach Produktart, fachlicher Anforderungen an das Produkt und Organisation kann die Entwicklung anders erfolgen.

[13] Die Einstufung der Sicherheitskritikalität dient hier lediglich zur Darstellung des Fallbeispiels. Es sollen keine Rückschlüsse daraus gezogen werden, welche Einstufung ein tatsächlich zu entwickelndes System dieser Art hätte. Dies kann nur im Rahmen einer von Experten für funktionale Sicherheit durchgeführten Gefahren- und Risikoanalyse bezogen auf ein konkretes Entwicklungsvorhaben erfolgen.

2. Das System des Paketzustellers muss um das amtliche Kennzeichen, Zeitfenster und Position erweitert werden und diese von Onlineshops empfangen können.
3. Das Fahrzeug muss dem Paketzusteller ein GPS-Signal mit der genauen Position des Fahrzeugs übermitteln.
4. Das System des Paketzustellers muss das GPS-Signal empfangen und verarbeiten.
5. Das Fahrzeug muss dem Paketzusteller einen Code übermitteln, den dieser zum einmaligen Öffnen des Kofferraums des Fahrzeugs nutzen kann.
6. Das System des Paketzustellers muss den Code empfangen und verarbeiten.
7. Das Fahrzeug muss den Code vor dem einmaligen Öffnen prüfen und bei erfolgreicher Verifizierung den Kofferraum öffnen.

Die Umsetzung der Punkte 3., 5. und 7. liegt in der Verantwortung des OEMs und soll hier näher betrachtet werden. Es wird angenommen, dass eine Gefahren- und Risikoanalyse keine besondere Sicherheitskritikalität festgestellt hat und daher die ISO 26262 nicht zwingend einzuhalten ist[14]. Demgegenüber ergeben sich eine Vielzahl an Anforderungen hinsichtlich der Cybersicherheit. So muss beispielsweise sichergestellt werden, dass die Kommunikation zwischen Fahrzeug und Paketzusteller von Unbefugten nicht abgefangen und ausgelesen werden kann und dass die Eingabe des Zugangscodes ausreichend vor Hackangriffen geschützt ist. Zur Kommunikation mit dem Paketzusteller und zum Öffnen des Kofferraums müssen bestehende Steuergeräte des Fahrzeugs erweitert werden. Hierbei sind die Anforderungen von Automotive SPICE einzuhalten. Neben diesen Randbedingungen definiert das Qualitätsmanagement fachliche Anforderungen z. B. hinsichtlich der Wartbarkeit des Systems durch Werkstätten oder der Deaktivierung des Systems durch den OEM.

Beispiel Ladesäulensuche

Es soll eine App entwickelt werden, mit der Fahrer Ladesäulen finden und weitere Informationen zu den jeweiligen Ladesäulen (Anbieter, Kosten, Verfügbarkeit etc.) erhalten können. Da dazu weder Steuergeräte des Fahrzeugs entwickelt, verändert oder angesteuert noch sicherheitskritische Funktionen realisiert oder angepasst werden müssen, müssen Automotive SPICE und die ISO 26262 nicht berücksichtigt werden. Es gibt Anforderungen hinsichtlich der Cybersicherheit (so sollten GPS-Daten des App-Nutzers verschlüsselt übertragen werden); diese gehen jedoch nicht über die Sicherheitsanforderungen vergleichbarer Apps hinaus. Das Qualitätsmanagement selbst hat keine fachlichen Anforderungen an die App, muss aber die Qualität der umgesetzten funktionalen Anforderungen und die Wartbarkeit der App sicherstellen.

[14] Die Einstufung der Sicherheitskritikalität dient hier lediglich zur Darstellung des Fallbeispiels. Es sollen keine Rückschlüsse daraus gezogen werden, welche Einstufung ein tatsächlich zu entwickelndes System dieser Art hätte. Dies kann nur im Rahmen einer von Experten für funktionale Sicherheit durchgeführten Gefahren- und Risikoanalyse bezogen auf ein konkretes Entwicklungsvorhaben erfolgen.

Die Bestimmung der Produktart und die Ermittlung der Q-Anforderungen unterscheidet sich im agilen Umfeld grundsätzlich nicht vom klassischen Vorgehen: Fachliche Anforderungen müssen ebenso erhoben und Randbedingungen ebenso erfüllt werden wie bei einer wasserfallartigen Entwicklung auch. Jedoch unterscheidet sich typischerweise die Art der Dokumentation und Übermittlung der Anforderungen an die Entwicklung. Statt der klassischen Anforderungsbeschreibung in Form eines Lastenheftes werden die Anforderungen bei einer agilen Entwicklung in das Product-Backlog aufgenommen und meist in Form von User-Stories verfasst. Die Anforderungen sollten möglichst den INVEST-Kriterien (*independent, negotiable, valuable, estimable, small, testable*) entsprechen (Wake 2003).

Beispiel Paketzustellung

Auszug aus dem Product-Backlog:

Anforderung X: „Als OEM kann ich den Betrieb des Paketzustelldienstes generell deaktivieren, um im Falle von entdeckten, nicht zeitnah lösbaren Sicherheitsproblemen, Häufung von Beschwerden oder sonstigen Gründen den Dienst (vorläufig) einstellen zu können."

Anforderung Y: „Als Werkstatt-Mitarbeiter kann ich ein Update der Software für das Steuergerät zur Kommunikation mit dem Paketzusteller einspielen, um das Steuergerät auf dem aktuellen Stand halten zu können."

etc.

Die User-Stories sollten um Akzeptanzkriterien ergänzt und im späteren Dialog mit dem Entwicklungsteam erläutert und ggf. konkretisiert werden. Es ist die Aufgabe des Product-Owners, die fachlichen Anforderungen in Absprache mit dem Anforderer soweit in feingranulare fachliche Einzelanforderungen zu zerlegen, dass die Anforderungen innerhalb einer Iteration umgesetzt werden können (*ready for sprint*).

Gerade fachliche Qualitätsanforderungen und Randbedingungen können in vorhergehenden Projekten bereits formuliert und in die damalige Entwicklung eingesteuert worden sein. Meist liegen diese Anforderungen bereits detailliert spezifiziert vor. Auch wenn im agilen Umfeld eine solche dokumentierte Detailspezifikation einzelner Anforderungen unüblich ist, so ergibt es dennoch Sinn, solche existierende Dokumente für neue agile Entwicklungen zu nutzen.

Beispiel Parkassistentsystem

Eine Reihe von Q-Anforderungen sind bei vorhergehenden Entwicklungen von Parkassistentsystemen im Anforderungsmanagementsystem DOORS bereits erfasst worden. Bei der neuen Entwicklung soll JIRA als Tool zur Verwaltung des Product-Backlogs verwendet werden. Dort werden die neuen funktionalen Anforderungen gesammelt. Eine Übertragung der aus Vorprojekten bekannten, in DOORS vorliegenden

Anforderungen in JIRA hat sich als unpraktikabel erwiesen, sodass zu jeder fachlichen Einzelanforderung in DOORS ein Backlog-Eintrag in JIRA mit einer Kurzbeschreibung und einem Link auf den entsprechenden DOORS-Eintrag erstellt wird. Um die in Automotive SPICE geforderte bidirektionale Traceability sicherzustellen, werden auch in DOORS die IDs der Backlog-Einträge gepflegt.

Bei einer agilen Entwicklung können Anforderungen auch nach Beginn der Implementierung in die Entwicklung eingesteuert werden. Dies gilt auch für Q-Anforderungen. Entscheidend ist, dass das Qualitätsmanagement über neue geplante funktionale Anforderungen informiert ist und die sich daraus ergebenen Q-Anforderungen über den Product-Owner in das Backlog aufnehmen lassen kann.

Beispiel Ladesäulensuche

Die App soll um Funktionen zur Personalisierung erweitert werden (z. B. Ladesäulensuche in der Nähe der hinterlegten Wohnungs- und Arbeitsplatzadressen). Daraus ergeben sich rechtliche Randbedingungen bezüglich des Datenschutzes, z. B. Einhaltung der DSGVO in Europa. Diese Randbedingungen müssen ebenfalls in die Entwicklung einfließen.

4.1.2. Entscheidung für oder gegen Agilität

Einfaches oder komplexes Vorhaben?

Agilität ist kein Selbstzweck, sondern dient der effektiven Umsetzung komplexer Entwicklungsvorhaben. Generell sollte vor jeder Entwicklung die Frage gestellt werden: Ist es sinnvoll ein System agil zu entwickeln? Das entscheidende Argument für Agilität ist die Unsicherheit bzgl. der fachlichen Anforderungen und der technischen Umsetzung bei komplexen Produkten. Ob diese Komplexität bei der Entwicklung eines Systems vorliegt, kann nicht pauschal beantwortet werden. Beispielsweise dürfte die Komplexität von Steuergeräten im Fahrzeug tendenziell weiter zunehmen und somit agile Methoden vermehrt sinnvoll sein. Dies kann letztendlich nur für jedes zu entwickelnde System einzeln entschieden werden. Die initial erfassten Anforderungen, auch die bezüglich der Qualität des Produkts, sind hierfür wichtige Entscheidungsgrundlage. Ist davon auszugehen, dass diese Anforderungen vollständig, korrekt und ausreichend präzise beschreiben sind, sollte eher nach einem klassischen Vorgehen entwickelt werden.

Geringe oder hohe Änderungskosten?

Darüber hinaus können hohe Änderungskosten Agilität behindern oder gar verhindern. Es sollte geklärt werden, ob es möglich ist, z. B. eingebettete Systeme, bestehend aus Hard- und Software, agil zu entwickeln – zumindest aus rein fachlicher Sicht, d. h. ohne Berücksichtigung von Randbedingungen durch Gesetze und Standards. Voraussetzung für ein wirtschaftliches, agiles Vorgehen sind geringe Kosten für notwendige Überarbeitungen der bisher realisierten Inkremente zur

Umsetzung weiterer Anforderungen. Diese Kosten scheinen für Hardware jedoch sehr hoch zu sein: Neue Funktionen können ein Überarbeiten des Hardwaredesigns und somit die Erstellung neuer Produktionswerkzeuge erforderlich machen. Spätestens bei der Vorstellung, neue Versionen eingebetteter Systeme kontinuierlich auszurollen, verbunden mit Rückrufaktionen nach jeder Entwicklungsiteration, zeigt sich, dass eine Agilität inkl. Continuous Delivery wie bei einer reinen Softwareentwicklung wirtschaftlich unsinnig wäre. Betrachtet man jedoch nur die reine Entwicklung und klammert die Produktion und Auslieferung aus, zeigt sich, dass Hardware in vielen Fällen durchaus wirtschaftlich agil entwickelt werden kann. Dazu muss lediglich der Begriff „auslieferbar" weiter gefasst werden: Meint „auslieferbar" in der reinen Softwareentwicklung die Auslieferung an den Kunden, so sollte „auslieferbar" hier als „fertig für die serienmäßige Produktion" interpretiert werden. Die Produktion selbst erfolgt dann klassisch phasenorientiert und nach wohldefinierten Prozessschritten.

Beispiel Parkassistentsystem

Angenommen, für das neue Parkassistentsystem muss die Hardware von Sensoren überarbeitet werden. Eine häufige Anpassung der Produktionsanlagen für die Erstellung der Sensoren wäre in diesem Fall mit unvertretbar hohen Kosten verbunden. Ein Lösungsansatz könnte darin bestehen, dass der Product-Owner beschließt, Product-Backlog-Items, die die Sensorik betreffen bzw. auf den erweiterten Funktionsumfang der Sensoren zurückgreifen, vorrangig in die Entwicklung einzusteuern. Die gesamte Entwicklung einschließlich der Hardwareentwicklung der Sensoren würde dann agil in einem interdisziplinären Team erfolgen. Erst wenn die Hardware der Sensoren ausreichend durch Prototypentwicklungen ausgereift ist, werden die Sensoren wie gewohnt produziert. Die weitere Softwareentwicklung unter Verwendung der neuen Sensoren erfolgt weiter agil.

Randbedingungen mit Agilität vereinbar?

Gesetzliche Vorschriften, branchenspezifische Richtlinien und hausinterne Prozesse können sich auf die Entwicklung von Produkten beziehen und müssen beachtet werden. Falls ein Produkt agil entwickelt werden soll, muss geprüft werden, ob das agile Vorgehen mit diesen Vorschriften, Richtlinien und internen Anweisungen – im Folgenden unter dem Begriff *Standards* zusammengefasst – grundsätzlich vereinbar ist und inwieweit das agile Vorgehen ggf. modifiziert werden muss. Dazu wird hier folgender Prozess vorgeschlagen, der in einem Unternehmen bei der Einführung von agilen Methoden einmalig und ggf. bei Änderungen relevanter Standards erneut durchgeführt werden sollte.

1. **Konflikte zwischen Standard und Agilität identifizieren**: Je Standard ist zu prüfen, ob agile Entwicklung mit dem Standard in Einklang zu bringen ist. Eine pauschale Beurteilung des gesamten Standards dürfte in vielen Fällen nicht möglich sein. Vielmehr müssen Prozessgruppen, Prozessschritte, Prozessergebnisse, Artefakte oder sonstige Regeln – im Folgenden unter dem Begriff *Regel* zusammengefasst – hinsichtlich der Vereinbarkeit mit agilem Vorgehen untersucht werden.

2. **Regeln fachlich bewerten**: Sofern eine Regel agilen Grundsätzen, Praktiken oder dem vorgesehenen agilen Framework widerspricht, sollte geprüft werden, ob die Regel fachlich sinnvoll ist. Dabei muss der teilweise oder vollständige Verlust von Agilität durch die Einhaltung der Regel berücksichtigt werden. Dies ist der schwerste Teil der Bewertung: Für jede Regel dürfte es Gründe geben, die für die Beibehaltung der Regel sprechen. Die große Herausforderung besteht darin, abzuwägen, ob diese Gründe einen Verzicht oder eine Einschränkung von Agilität rechtfertigen. Sollte eine Regel als so sinnvoll bewertet werden, dass der Verlust an Agilität in Kauf genommen werden sollte, so sollte diese Regel unverändert beibehalten werden.
3. **Regeln ggf. regelkonform ändern**: Falls die Beibehaltung einer Regel den Verlust der Agilität nicht aufwiegt, so sollte geprüft werden, ob die Regel durch Tailoring so angepasst werden kann, dass die Regel konform mit einer agilen Entwicklung ist.
4. **Regeln ggf. verhandeln**: Falls eine Uminterpretation bzw. ein Tailoring nicht sinnvoll möglich ist, kann ggf. die Aussetzung der Regel oder deren Veränderung verhandelt werden. Hierzu kommen insbesondere Regeln von Standards in Frage, die hausintern festgelegt worden sind.

Im Abschnitt 3 sind die Standards Automotive SPICE, ISO 26262, AUTOSAR und die Reifegradabsicherung für Neuteile gemäß VDA hinsichtlich der Vereinbarkeit mit agiler Entwicklung beschrieben.

Beispiel Parkassistentsystem

Wie oben bereits beschrieben müssen bei der hier angenommenen Parkassistentsystementwicklung AUTOSAR, Automotive SPICE und die Anforderungen an ASIL D gemäß der ISO 26262 berücksichtigt werden. Die Kompatibilität dieser Standards mit agiler Entwicklung ist im Abschnitt 3 erläutert. Es wird hier angenommen, dass ein agiles Vorgehen grundsätzlich möglich ist. Der von der ISO 26262 geforderte Unabhängigkeitsgrad I3 bei Bestätigungsmaßnahmen für ASIL D wird dadurch sichergestellt, dass die Prüfer im Auftrag des Product-Owners zwar eng und kontinuierlich mit dem Entwicklungsteam zusammenarbeiten, jedoch zu anderen Abteilungen gehören und somit anderen Vorgesetzten berichten als die Mitglieder des agilen Teams. Zudem wird im Unternehmen grundsätzlich gefordert, dass der Reifegrad aller Neuteile prinzipiell gemäß dem VDA-Standard geprüft wird. Aufgrund der schlechten Vereinbarkeit dieses Standards mit agiler Entwicklung (s. Abschnitt 3.4) werden für die Entwicklung des Parkassistentsystems folgenden Änderungen vereinbart:

- Für die Reifegradbewertungen RG 1 und RG 2 muss statt eines vollständigen Lastenheftes ein initiales Product-Backlog erstellt werden, in dem alle bekannten Randbedingungen und sonstige Anforderungen, die bereits als Lastenheftpunkte umfassend dokumentiert vorliegen, in das Product-Backlog referenziert werden.
- Die Reifegradbewertung RG 3 wird nicht gefordert.

- Für die Reifegradbewertung RG 4 müssen lediglich die Anforderungen hinsichtlich Continuous Integration einschließlich automatisierter Regressionstests erfüllt sein.
- Die Reifegradbewertung RG 5 umfasst lediglich die Umsetzung der Mindestanforderungen.

Beispiel Paketzustellung

Für die Paketzustellung müssen insbesondere Maßnahmen zur Sicherung der Cybersecurity umgesetzt werden. Voraussichtlich wird dazu zukünftig der ISO-Standard 21434 zu berücksichtigen sein (ISO/SAE 2017). Da sich dieser noch im Aufbau befindet und die Common Criteria zu starre Prozessvorgaben machen (CC 2009), beschließt das Unternehmen, dass mindestens ein Cybersecurity-Experte Teil des Entwicklungsteams sein muss. Der Umgang mit dem VDA-Standard zur Reifegradabsicherung für Neuteile erfolgt wie beim dem Parkassistentsystem.

Beispiel Ladesäulensuche

Für die Ladesäulensuche sind weder Branchenstandards zu berücksichtigen noch muss die Entwicklung bei der gesamten Fahrzeugentwicklung mitberücksichtigt werden, sodass deren Reifegradabsicherung nicht gemäß VDA zu prüfen ist.

4.1.3. Organisation

Wie bei klassischen Entwicklungsprojekten muss die grundlegende Organisation des Entwicklungsvorhabens im Vorfeld geklärt werden. Im agilen Umfeld müssen dazu u. a. folgende Fragen beantwortet werden:

- Nach welchem agilen Framework soll die Entwicklung erfolgen?
- Wie sieht bei Bedarf das Skalierungsmodell aus?
- Wer übernimmt die Rolle des Product-Owners, des Agile Team-Masters bzw. ScrumMasters? Zu welcher Organisation bzw. Organisationseinheit gehören diese Rollen?
- Wie soll das Entwicklungsteam zusammengesetzt sein? Zu welchen Organisationen/Organisationseinheiten sollen deren Mitglieder gehören?
- Wie wird die Transparenz des Product-Backlogs, des Entwicklungsstands und der anderen Artefakte sichergestellt? Wer soll in den Informationsfluss eingebunden werden?
- Wer kann/soll an den agilen Ritualen teilnehmen?

Je nach Entwicklungsvorhaben können die Frage sehr unterschiedlich beantwortet werden.

Beispiel Parkassistentsystem

Die Gesamtverantwortung für die Entwicklung des Parkassistentsystems liegt bei der Technischen Entwicklung eines OEMs. Der Vertrieb stellt den Product-Owner. Ein Lieferant soll die Software für das System entwickeln. Darüber hinaus werden einzelne Lieferanten von Systemkomponenten (z. B. für die Sensorik) teils über den OEM, teils über den Tier-1-Lieferanten eingebunden. Die Entwicklung bei dem Lieferanten erfolgt nach dem Extreme-Programming-Framework. Der Lieferant hat drei agile Teams aufgebaut, die lose gekoppelt sind und im Wesentlichen gemäß dem SAFe-Ansatz ihre Aktivitäten abstimmen (s. dazu Abschnitt 1.9.2). Jedes dieser Teams hat einen eigenen agilen Team-Master und einen eigenen Team-Product-Owner. Seitens des OEMs gibt es einen Bereich *Qualitätsmanagement*, der in Abstimmung mit dem Product-Owner des Vertriebs Anforderungen einsteuern und das entwickelte Gesamtsystem vor jeder Auslieferung formal abnehmen wird (s. dazu Abschnitt 4.4).

Beispiel Paketzustellung

Die Entwicklung des Systems zur Paketzustellung im Auto erfolgt durch ein Konsortium bestehend aus einem OEM, einem Paketzusteller und einem Onlinehändler. Jedes dieser Mitglieder stellt ein Team. Die Teams sind eng miteinander gekoppelt und stimmen sich gemäß LeSS ab (s. dazu Abschnitt 1.9.1). Die Technische Entwicklung des OEMs stellt den Product-Owner. Das Team des OEMs entwickelt nach Scrum. In dem Team des OEMs nimmt ein Cybersecurity-Experte aktiv an der Entwicklung teil. Auch hier begleitet der Fachbereich *Qualitätsmanagement* des OEMs die Entwicklung und kann sich über den aktuellen Entwicklungsstand jederzeit informieren.

Beispiel Ladesäulensuche

Ein einzelnes Team soll die App zur Ladesäulensuche im Auftrag eines OEMs entwickeln. Die Entwicklung soll zunächst nach Scrum, später ggf. nach Kanban erfolgen. Der OEM stellt den Product-Owner; das Entwicklungsteam und der ScrumMaster werden vom Auftragnehmer gestellt. Der Fachbereich *Qualitätsmanagement* des OEMs steht während der gesamten Entwicklung im Austausch mit dem Product-Owner, stellt über ihn Anforderungen ein und nimmt bei Bedarf an Backlog-Refinements und Sprint-Reviews teil.

4.2. Lieferanten bewerten

4.2.1. Lieferantenaudit

Viele Unternehmen der Automobilindustrie fordern die Durchführung eines Lieferantenaudits vor der Beauftragung. Je nach Art des zu entwickelnden Produktes sind hierzu unterschiedliche Standards einzuhalten. Bei einer agilen Entwicklung muss geprüft werden, ob und inwieweit diese Standards mit einem agilen Vorgehen kompatibel sind (s. dazu auch Abschnitte 3 und 4.1.2).

Im Gegensatz zu Lieferantenaudits bei klassischen Entwicklungsprojekten sollten folgende Kriterien bei einer agilen Entwicklung berücksichtigt werden:

- Der Lieferant sollte Expertise für die Entwicklung nach agilen Grundsätzen und Prinzipien und je nach agilem Framework entsprechende Qualifikationen bzw. Zertifikate für die von ihm übernommenen agilen Rollen vorweisen.
- Der Lieferant sollte bei Bedarf Know-how, Werkzeuge und Methoden zur qualitätsgesicherten Continuous Integration und zum Continuous Delivery nachweisen.

4.2.2. Vertragsgestaltung und Probezeit für Lieferanten

Das Gestalten der Verträge mit Lieferanten ist nicht Aufgabe des Qualitätsmanagements. Dennoch spielt die Art des Vertrages eine große Rolle für das Qualitätsmanagement, da im Vertrag u. a. die Anforderungen an die Bewertung der Lieferanten festgeschrieben sein sollten. Im Gegensatz zum klassischen Wasserfallmodell gibt es bei einer agilen Entwicklung kein fertiges Lastenheft, das als Grundlage eines Werkvertrages dienen könnte. Stattdessen müssen Alternativen zur klassischen Festpreisvereinbarung mit verbindlicher Anforderungsspezifikation gefunden werden (s. Abschnitt 1.5). So könnte dem Auftraggeber die Möglichkeit eingeräumt werden, den Vertrag zu kündigen und einen anderen Lieferanten zu beauftragen. Die Bewertung des Lieferanten erfolgt dann vielmehr während der Zusammenarbeit, weniger vorab durch ein Lieferantenaudit.

Beispiel Ladesäulensuche

Zwei Unternehmen stehen in der engeren Wahl, die App zur Ladesäulensuche zu realisieren. Der Auftraggeber vereinbart mit beiden Unternehmen eine Dienstleistungsvereinbarung, in der sich die Unternehmen verpflichten, alle zwei Wochen auslieferbare Inkremente zu entwickeln. Vier Monate nach Beginn der Entwicklung bewertet der Auftraggeber die Leistungen der beiden Unternehmen, entscheidet sich für eins der Unternehmen zur weiteren Zusammenarbeit und kündigt den Vertrag mit dem anderen.

Um die Konformität mit den hausinternen Richtlinien zum Thema Arbeitnehmerüberlassung sicherzustellen, ist in den Verträgen festgelegt, dass bei der Sprint-Planung immer der offizielle Ansprechpartner des Auftragnehmers anwesend ist, der Aufträge entgegennehmen kann.

Beispiel Paketzustellung

Hausinterne Richtlinien sehen vor, dass Aufträge dieser Art nur als Werkverträge vergeben werden dürfen. Um Agilität durch den Vertrag nicht zu verhindern, kann jedoch nur die Zieldefinition der Entwicklung und generelle fachliche Anforderungen Vertragsgrundlage sein. Um den Werkscharakter des Vertrages zu gewährleisten, wird das Vorgehen nach Scrum mit einer Sprintlänge von drei Wochen vertraglich vereinbart und eine Abnahme jedes Inkrements durch den Product-Owner im Rahmen des Sprint-Reviews vorgeschrieben; bei mangelhafter Umsetzung des Sprintziels haftet der Auftragnehmer. Beiden Seiten wird das Recht auf Kündigung zum Ende eines Sprints mit einer definierten Kündigungsfrist eingeräumt. Die Definition-of-Done sieht vor, dass zu einem Sprintende dem Auftraggeber der Quellcode zur Verfügung gestellt und das dann aktuelle System hinreichend dokumentiert sein muss.

4.3. Reifegrad absichern

Die Reifegradabsicherung für Software im Allgemeinen ist im Abschnitt 2 ausführlich beschrieben. Demnach muss grundsätzlich unterschieden werden zwischen einem Qualitätsmanagement auf Seiten des Auftraggebers und einem Qualitätsmanagement auf Seiten des Auftragnehmers (siehe Abschnitt 2.2). Während das Qualitätsmanagement des Auftraggebers die Qualität von außen validiert, unterstützt das Qualitätsmanagement des Auftragnehmers die Verifizierung der Qualität als Teil des agilen Entwicklungsteams.

4.3.1. Verifizierung der Softwarequalität durch das Entwicklungsteam

In Abschnitt 2.3 ist ausführlich dargelegt, wie ein Entwicklungsteam die Qualität von Software testen kann. In einem agilen Entwicklungsteam ist es Aufgabe des gesamten Teams, die Qualität sicherzustellen. Qualitätssicherer können und sollten in vielen Fällen Teil des Entwicklungsteams sein, unabhängig davon, ob sie zur selben Organisationseinheit wie die Programmierer gehören oder nicht. Diese Qualitätssicherer übernehmen einen Großteil der qualitätssichernden Aufgaben innerhalb des Teams, verantworten aber nicht allein die Qualität, sondern eben nur als Teil des Entwicklungsteams.

Beispiel Parkassistentsystem

Aufgrund der hohen Qualitätsanforderungen an das Parkassistentsystem insbesondere hinsichtlich der funktionalen Sicherheit, sind in den agilen Teams jeweils mehrere Qualitätssicherer mit den Schwerpunkten Testautomatisierung, Performance und funktionale Sicherheit Teil des Entwicklungsteams. Diese Qualitätssicherer identifizieren insbesondere Safety-relevante Testfälle, unterstützen die Entwickler bei der Implementierung der dazugehörigen Unit-Tests, verifizieren die integrierte Software und konfigurieren die DevOps-Werkzeuge so, dass alle kritischen Tests als Regressionstests vor der Auslieferung einer neuen Version automatisiert getestet werden. Die Qualitätssicherer gehören zu einer von der Entwicklung unabhängigen Organisationseinheit „Qualitätssicherung".

Beispiel Ladesäulensuche

Da die App zur Ladesäulensuche vorwiegend hohe Anforderung an die Usability der App hat, jedoch unkritisch bezüglich Cybersecurity und funktionale Sicherheit ist, besteht das Entwicklungsteam ausschließlich aus Softwareentwicklern, von denen einige Expertise in den Bereichen DevOps und User-Experience haben.

4.3.2. Validierung der fachlichen Anforderungen durch den Auftraggeber

Die Absicherung der Softwarequalität von außen, d. h. ohne direkte Beteiligung an der Entwicklung als Teil des Teams, ist ausführlich in Abschnitt 2.5 beschrieben. Im Wesentlichen ist es hierbei Aufgabe des Auftraggebers, die Umsetzung der funktionalen Anforderungen, Randbedingungen und Qualitätsanaforderungen rein fachlich zu validieren, dabei jedoch dem Entwicklungsteam die Entscheidungen zu überlassen, wie es diese Anforderungen technisch realisiert.

Wie ein solches Qualitätsmanagement konkret gestaltet werden kann, hängt stark von der Art des Produktes und den damit verbundenen Anforderungen ab und kann nicht allgemeingültig festgelegt werden. Im Folgenden sind mögliche Validierungsansätze exemplarisch und auszugsweise skizziert.

Beispiel Parkassistentsystem

Bei der Reifegradabsicherung für das Parkassistentsystem stehen insbesondere die Qualitätskriterien Funktionstauglichkeit – dabei wiederum insbesondere die funktionale Sicherheit –, Zeitverhalten und Zuverlässigkeit im Vordergrund. Diese Qualitätskriterien können auch von außen noch relativ einfach validiert werden. Hierzu wird das integrierte System, sei es in einem virtuellen Teststand oder die physische Komponente bzw. das physische Fahrzeug, hinsichtlich dieser Kriterien getestet. Das Testen selbst unterscheidet sich damit grundsätzlich nicht vom Testen eines klassisch entwickelten Systems, jedoch werden diese Tests häufiger und nicht nur zum Ende der Produktentwicklung durchgeführt. Zudem fokussiert sich das Qualitätsmanagement auf dieser Ebene auf explorative Tests; das massenhafte Testen

verschiedener Testfälle erfolgt innerhalb des Teams und möglichst schon auf Unit-Ebene (siehe hierzu auch Abschnitt 2.3.3).

Beispiel Paketzustellung

Im Fokus der Reifegradabsicherung steht neben der Benutzbarkeit durch den Anwender und der Funktionstauglichkeit der gesamten Dienstleistung gerade das Thema Cybersecurity. Das Qualitätsmanagement beauftragt dazu Security-Spezialisten, die anhand von Penetrationstest die Security-Maßnahmen prüfen. Die Ergebnisse dieser Tests fließen in die weitere Entwicklung ein.

Beispiel Ladesäulensuche

Die App zur Ladesäulensuche ist unkritisch hinsichtlich der funktionalen Sicherheit und Angreifbarkeit durch Hacker. Hingegen sind die Anforderungen an die Wartbarkeit, die Benutzbarkeit und die Portierbarkeit hoch. Gerade weil die App schnell auf dem Markt gebracht und dann sukzessiv um weitere Funktionen erweitert werden soll, muss ein hohes Maß an Modularität, Wiederverwendbarkeit, Analysierbarkeit, Modifizierbarkeit und Testbarkeit gewährleistet werden. Hierzu erfasst das Qualitätsmanagement regelmäßig die Geschwindigkeit der Umsetzung funktionaler Anforderungen. Ein Verlangsamen der Geschwindigkeit ist Indiz für mangelnde Wartbarkeit der Software.

Das Qualitätsmanagement fokussiert sich zudem auf die Usability der App und stimmt sich dazu mit dem Vertrieb und dem Product-Owner fortlaufend ab. Das Qualitätsmanagement führt Usability-Tests mit potentiellen Nutzern der App durch und diskutiert die Ergebnisse mit dem agilen Team. Der Product-Owner nimmt die sich daraus ergebenen Anforderungen in das Backlog auf und priorisiert diese. Zudem veranlasst das Qualitätsmanagement Performance- und Lasttests durch den Lieferanten und bewertet die Ergebnisse. Auch diese Tests erfolgen nicht einmal zum Projektende, sondern werden während der Entwicklungszeit mehrfach durchgeführt.

4.3.3. Automatisierung der Qualitätssicherung

Aufgrund der kurzen Iterationen ist bei einer agilen Entwicklung die Testautomatisierung von großer Bedeutung und eine Voraussetzung, um mit einem vertretbaren Aufwand Continuous Integration und Continuous Delivery, sofern gefordert, zu ermöglichen (s. Abschnitt 2.3.2). Je nach Produktart sollte das Qualitätsmanagement dies vom Lieferanten einfordern. Für das Qualitätsmanagement bedeutet dies zudem, dass während oder nach der Entwicklung entdeckte Fehler oder sonstige Defizite – sofern sinnvoll möglich – zukünftig durch einen automatisierten Regressionstest abgedeckt sein sollten und nicht manuell wiederholt werden müssen.

Beispiel Parkassistentsystem

Die einzelnen Entwicklungsteams der Lieferanten betreiben eigene Continuous Integration-Systeme für ihre zu entwickelnden Komponenten, mit deren Hilfe die Unit-getesteten Softwareeinheiten in die jeweiligen Integrationsumgebungen eingespielt und dort auf Komponentenebene automatisiert getestet werden.

Der OEM als Auftraggeber stellt den Entwicklungsteams darüber hinaus eine Plattform zur Verfügung, auf der die entwickelte Hardware eingebaut und die Software online eingespielt werden kann. Auf diese Abnahmeinstanz werden Änderungen erst eingespielt, wenn die Komponenten alle Tests beim jeweiligen Lieferanten erfolgreich durchlaufen haben. Die Abnahmeinstanz sollte dennoch den aktuellen Entwicklungsstand soweit wie möglich widerspiegeln. Die Abnahmeinstanz wird sowohl von Qualitätssicherern der agilen Entwicklungsteams, als auch vom Product-Owner, von Qualitätsmanagern und von Q-Experten des Kunden genutzt. Der OEM meldet dort die meist explorativ erkannten Fehler. Wenn möglich werden daraus abgeleiteten Testfunktionen beim Lieferanten bereits auf Unit-Ebene implementiert und zukünftig als Regressionstests beim Lieferanten automatisiert ausgeführt.

4.3.4. Kontinuität und Kooperation

Das klassische Wasserfallmodell sieht Phasen für die Qualitätssicherung vor. Bei einer agilen Entwicklung gibt es solche Phasen nicht. Qualitätssicherung begleitet sowohl den gesamten Entwicklungsprozess innerhalb des Entwicklungsteams als auch die Reifegradprüfung durch den Kunden. Für das kundenseitige Qualitätsmanagement bedeutet dies, dass es während einer Iteration über umgesetzte Product-Backlog-Items (PBIs) informiert sein sollte, bei Bedarf und in Absprache mit dem Product-Owner an Reviews teilnehmen kann und den Releaseplan des Product-Owners kennt, um so vor einem geplanten Release das Produkt ausgiebiger testen zu können. Ziel ist es, dem agilen Team ein möglichst frühes Feedback zu den umgesetzten PBIs und Inkrementen geben zu können.

Eine solche Kontinuität in der Zusammenarbeit zwischen Kunden und Lieferant setzt eine andere Art der Zusammenarbeit als bei einem klassischen Vorgehen voraus. Das Qualitätsmanagement – ob nun allein vertreten durch den Product-Owner oder durch einem oder mehrere Stakeholder aus einem Bereich *Qualitätsmanagement* – nimmt möglichst persönlich an agilen Events wie Backlog-Refinement oder Review teil. Es kennt die Entwickler und steht dem Entwicklungsteam als Ansprechpartner zur Verfügung.

Beispiel Parkassistentsystem

Das Qualitätsmanagement des OEM prüft insbesondere die Umsetzung ihrer eigenen fachlichen Q-Anforderungen und Anforderungen zur funktionalen Sicherheit. Anforderungen zum Beheben von Fehlern und Defizite werden nur in Abstimmung mit dem Product-Owner in das Product-Backlog aufgenommen und priorisiert. Qualitätsmanager, Sicherheitsingenieure und sonstige Q-Experten des OEM nehmen je

nach Bedarf an Gesprächen zur Anforderungserhebung und -klärung und an der Vorstellung der Inkremente teil und stehen somit im Dialog mit den agilen Teams.

Beispiel Paketzustellung

Der Fachbereich Qualitätsmanagement kennt den jeweils aktuellen Releaseplan für das Paketzustellsystem und übernimmt hier hauptsächlich koordinierende Aufgaben vor der Auslieferung, indem es sich die Durchführung der erforderlichen Security-Prüfungen durch den Product-Owner bzw. den Security-Experten bestätigen lässt. Bei der Umsetzung eigener Q-Anforderungen z. B. hinsichtlich der Wartbarkeit durch Werkstätten ist der Bereich ebenso über den Product-Owner in die Entwicklung eingebunden wie andere Stakeholder auch, d. h. es nimmt bei Bedarf an Sprint-Reviews teil und kann neue Anforderungen auch noch während der Entwicklung stellen.

4.4. Produkte freigeben

Je nach Produktart muss das Qualitätsmanagement ggf. eine Auslieferung des Produktes vor Kunde bzw. eine Übernahme in den Produktionsprozess formal freigeben. Dieser Freigabeprozess ist weder Teil des agilen Manifests noch eines der üblichen agilen Frameworks. Stattdessen kann beispielsweise der Product-Owner gemäß Scrum eigenständig entscheiden, ob ein Inkrement ausgeliefert werden soll oder nicht.

Dennoch stellt ein formaler Freigabeprozess durch das Qualitätsmanagement keinen grundsätzlichen Widerspruch zu einem agilen Vorgehen dar, solange das Qualitätsmanagement das Produkt während der vorangegangenen Entwicklungszeit geprüft und Mängel zeitnah dem Team gemeldet hat, sodass die Endabnahme keine umfangreichen Änderungsanforderungen ergeben sollte und die Freigabe zeitnah erfolgen kann. Es sollte unbedingt vermieden werden, dass vor der geplanten Freigabe Qualitätsmanager oder andere Stakeholder Defizite, die schon lange in dem Produkt enthalten sind und schon früher hätten erkannt werden können, melden oder gar neue Anforderungen stellen und so eine rechtzeitige Freigabe verhindern. Dies wird dadurch vermieden, indem alle Stakeholder während der gesamten Entwicklungszeit eng und kontinuierlich mit dem Product-Owner und dem agilen Team zusammenarbeiten und sich dabei auf das fachlich und rechtlich Notwendige fokussieren.

Je nach Produktklasse und Kontext können die Release-Zyklen im agilen Umfeld sehr viel kürzer sein als bei einer klassischen Entwicklung. Gerade bei kurzen Zyklen muss die Freigabe schnell erfolgen können. Hierbei hilft ein hoher Grad an Testautomatisierung, die aufwendige manuelle Prüfungen zumindest teilweise überflüssig macht.

Beispiel Parkassistentsystem

Die Entwicklung des Parkassistentsystems ist in die Gesamtfahrzeugentwicklung mit einem vorgegebenen Termin für den Start-of-Production (SOP) eingebettet. Obwohl das Parkassistentsystem inkrementell in kurzen Iterationen von wenigen Wochen entwickelt wird, gibt es nur einen Releasetermin. Die erforderliche formale Abnahme und Freigabe durch das Qualitätsmanagement erfolgt wie gewohnt, kann jedoch aufgrund der vorgehenden Zusammenarbeit zwischen den agilen Teams und dem Qualitätsmanagement und aufgrund der inzwischen hohen Testautomatisierung schneller erfolgen als bei einem klassischen Vorgehen üblich. Der Aufwand für das Qualitätsmanagement ballt sich nicht kurz vor dem SOP, sondern verteilt sich über die gesamte Entwicklungszeit.

Beispiel Ladesäulensuche

Die App zur Ladesäulensuche soll so schnell wie möglich ausgerollt werden, um das Nutzungsverhalten und die Akzeptanz im Markt zeitnah ermitteln zu können. Das erste Release stellt lediglich ein Minimum Viable Product (MVP) dar, dass dann sukzessiv funktional erweitert wird. Der Product-Owner entscheidet, wann das Produkt ein MVP ist und als erstes Release ausgerollt werden soll. Dann erfolgen Updates ca. alle zwei bis drei Sprints; der Product-Owner entscheidet auch hier über den Rollout-Termin. Das Qualitätsmanagement, das während der gesamten Entwicklungszeit im Austausch mit dem Product-Owner steht und über den Entwicklungsstand informiert ist, hat sich vergewissert, dass es eine umfassende Suite für Regressionstests gibt, die für das Qualitätsmanagement jederzeit einsehbar ist, und dass die Prozesse für den Continuous Integration und Continuous Delivery implementiert sind. Daher gibt das Qualitätsmanagement die jeweiligen Releases ohne umfangreiche Prüfungen zeitnah frei.

4.5. Feld beobachten und Lessons Learned ableiten

Die Feldbeobachtung erfolgt bei einer agilen Entwicklung der ausgelieferten Produkte grundsätzlich genauso wie bei klassisch entwickelten Produkten. Die in der Automobilindustrie etablierten Fehlerabstellprozesse können auch für agil entwickelte Produkte genutzt werden.

Falls ein Produkt durch Updates verändert wird, können Erkenntnisse aus dem Feld ganz im agilen Sinne in die aktuelle Entwicklung einfließen. Insbesondere kann das agile Team Annahmen mit Hilfe von A/B-Tests prüfen. Häufige Releases erlauben eine zeitnahe Ableitung von Lessons Learned. Diese können sowohl der Product-Owner und die Stakeholder verwenden, um neue Anforderungen zu erkennen und zu stellen, als auch das agile Team, um das Produkt technisch zu optimieren.

Beispiel Parkassistentsystem

Das neue Parkassistentsystem wird nach dem SOP für das Fahrzeugprojekt, für das das System entwickelt worden ist, im Feld genutzt werden. Beanstandungen am Parkassistentsystem werden vom OEM wie gewohnt über den Fehlerabstellprozess bearbeitet und ggf. Lessons Learned für die Entwicklung zukünftiger vergleichbarer Systeme abgeleitet.

Beispiel Ladesäulensuche

Der Product-Owner analysiert das Nutzungsverhalten, indem er die Datenbanken und Logdateien mit Hilfe von Analysewerkzeugen auswertet und die für ihn relevanten KPIs bewertet. Daraus kann er schließen, welchen Wert die einzelnen bislang umgesetzten PBIs haben und welche Werte die noch nicht umgesetzten PBIs voraussichtlich besitzen. Entsprechend priorisiert er das Product-Backlog und ergänzt es um neue Einträge.

4.6. Agile Grundsätze leben

In den vorhergehenden Abschnitten wurden die einzelnen Aufgabenfelder des Qualitätsmanagements unter dem Blickwinkel einer agilen Produktentwicklung beleuchtet. Entscheidend für eine erfolgreiche agile Entwicklung sind die agilen Grundsätze, wie sie im agilen Manifest formuliert sind. Diese Grundsätze gelten für die gesamte agile Entwicklung und sollten bei allen Aktivitäten, auch hinsichtlich des Qualitätsmanagements, berücksichtigt und von allen Beteiligten gelebt werden. Die vier agilen Grundsätze und deren Bedeutung für das Qualitätsmanagement werden daher im Folgenden zusammenfassend und abschließend betrachtet.

4.6.1. Individuals and interactions over processes and tools

Die Qualität des Produktes hängt entscheidend von der Kompetenz aller beteiligten Mitarbeiter und deren Zusammenarbeit ab. Dies gilt gerade auch für agile Entwicklung und betrifft auch die Qualitätssicherer innerhalb des Teams und die Qualitätsmanager, die kundenseitig Q-Anforderungen einbringen und validieren. Der persönliche Austausch der Fachexperten und das gemeinsame Finden von Lösungen ersetzen im agilen Umfeld oftmals umfassende Prozessrichtlinien. Dies stellt große Anforderungen an die Mitarbeiter hinsichtlich ihrer Fach-, Kommunikations- und Teamkompetenz. Es verändert die Rolle und die Art der Arbeit des Qualitätsmanagements: Es ist weniger Qualitätspolizei, die die formale Einhaltung von Standard verifiziert, sondern vielmehr Partner des agilen Teams, der die Entwicklung unterstützt, indem es die Kundensicht einnimmt

und in die Entwicklung einbringt und sicherstellt. Qualitätsmanager sollten dabei ergebnisorientiert und weniger prozessorientiert denken und handeln. Diese Kompetenzen – Teamfähigkeit, Kommunikationskompetenz und Ergebnisorientiertheit – sollten bei den Mitarbeitern des Qualitätsmanagements gefördert werden und sich in einem entsprechenden Talentmanagement widersiegeln. Qualitätsmanager, die die Einhaltung formaler Standards einfordern, ohne deren Sinnhaftigkeit in dem gegebenen Kontext zu hinterfragen, sind für ein agiles Arbeiten ungeeignet.

4.6.2. Working software over comprehensive documentation

Entscheidend für den Kunden ist, dass das Produkt funktioniert und die erforderliche Qualität aufweist. Dokumentationen zum Produkt und zur Produktentwicklung sind kein Selbstzweck, sondern sollten diesem Zweck dienen (s. dazu auch Abschnitt 2.6). Klassisches Qualitätsmanagement bewertet den Entwicklungsstand primär anhand von Dokumenten und fordert diese entsprechend ein. Das Qualitätsmanagement sollte jedoch verstehen und akzeptieren, dass ein agiles Team ggf. anders als im klassischen Umfeld üblich untereinander und mit seinem Umfeld kommuniziert und Informationen archiviert. Den Zweck von Dokumenten erzielt das Team vielleicht auf einem anderen Weg. Entscheidend ist, dass die fachlichen Anforderungen umgesetzt werden. Diese sollte das Qualitätsmanagement im Fokus haben und die Sinnhaftigkeit der Dokumentation diesbezüglich prüfen.

Das intensive Testen des auszuliefernden Produktes ist Voraussetzung, um die geforderte Funktionalität und Qualität des Produktes sicherzustellen. Dies gilt für agile ebenso wie für klassische Entwicklung. Durch die kurzen Iterationen und den häufigen Überarbeitungen des Produktes ist der Testumfang im agilen Umfeld besonders hoch. Testautomatisierung spielt hierbei eine Schlüsselrolle. Nur ein hoher Grad an Testautomatisierung kann sicherstellen, dass eine agile Entwicklung mit häufigen Auslieferungen und begrenzten Mitteln für das manuelle Testen keine fragile Entwicklung wird. Aufgabe des Qualitätsmanagements ist es, angemessene Anforderungen hinsichtlich einer qualitätsgesicherten Continuous Integration und eines Continuous Delivery zu stellen und deren Umsetzung zu unterstützen und sicherzustellen.

4.6.3. Customer collaboration over contract negotiation

Das agile Manifest betont die Bedeutung einer engen Zusammenarbeit zwischen dem Kunden und dem Lieferanten. Eine vertrauensvolle Zusammenarbeit ist wichtiger als die vertragliche Regelung zwischen den beiden Parteien. Für das kundenseitige Qualitätsmanagement bedeutet dies, dass es den Entwicklungsprozess kontinuierlich begleitet statt ausschließlich auf Basis der Aktenlage die Einhaltung vertraglicher Zusagen in einer Qualitätssicherungsphase zu prüfen. Der persönliche Kontakt und Austausch mit dem agilen Team – sei es der Product-Owner zur Klärung von Anforderungen, der Team Agility-Master zu prozessualen Fragen oder das Entwicklungsteam bei Fragen während einer Entwicklungsiteration – ist wichtige Voraussetzung für ein agil entwickeltes, qualitätsgesichertes und hochwertiges Produkt. Das Qualitätsmanagement sollte sich auf

eine kontinuierliche Begleitung und Teilnahme an agilen Events, insbesondere der Backlog-Refinements und der Sprint-Reviews, einstellen.

4.6.4. Responding to change over following a plan

Agile Entwicklung bietet dem Kunden die Möglichkeit, Anforderungen auch noch während der Produktentwicklung zu formulieren und in die Entwicklung einzusteuern. Dies gilt auch für Qualitätsanforderungen. Qualitätsdefizite, die während der Entwicklung und möglicherweise erst nach der Auslieferung festgestellt werden, können durch neue Q-Anforderungen in der laufenden Entwicklung behoben werden. Kritische Anforderungen, gerade hinsichtlich der funktionalen Sicherheit oder der Cybersecurity, sollten sicherlich vor der Auslieferung bekannt und umgesetzt sein. Unkritische Qualitätsanforderungen, beispielsweise bezüglich der Bedienbarkeit, kann das agile Team, sofern das Einspielen von Updates mit keinem großen Aufwand verbunden ist, hingegen auch nachträglich umsetzen. Bei frühen Auslieferungen und dem Einsatz von Analysewerkzeugen bietet sich dem Qualitätsmanagement die Möglichkeit, Erfahrungen aus dem Feld zu sammeln und neue Anforderungen abzuleiten. Das Qualitätsmanagement übernimmt damit eine wichtige Aufgabe zum Schließen des agilen Regelkreises.

4.7. Fazit

Das Qualitätsmanagement in der Automobilindustrie umfasst im agilen Umfeld dieselben Tätigkeitsfelder wie im klassischen Umfeld. Auch bei einer agilen Entwicklung müssen Qualitätsanforderungen eingesteuert, Lieferanten bewertet, Reifegrade geprüft, Auslieferungen freigegeben, das Feld beobachtet und Lessons Learned für die weitere Entwicklung berücksichtigt werden. Dabei gilt es ebenso, verbindliche Qualitätsstandard zu berücksichtigen und einzuhalten.

Die Art und Weise, wie diese Tätigkeiten organisiert und durchgeführt werden, kann sich jedoch erheblich vom Qualitätsmanagement bei wasserfallartigen Entwicklungen unterscheiden. Entsprechend müssen agile Teams und Qualitätsmanager Standards anders als gewohnt interpretieren und gleichzeitig die mit den Standards verbundenen Ziele verstehen und erreichen. Dies erfordert sowohl ein tiefes Verständnis der einzelnen Standards und deren sinnvolle Anwendung für das jeweilige Entwicklungsvorhaben als auch bedarfsweise eine Anpassung der agilen Praktiken.

Ist ein agiles Vorgehen für ein Entwicklungsvorhaben sinnvoll und möglich, so sollte die Qualität entsprechend den agilen Grundsätzen gemanagt und gesichert werden:

- Der persönliche Austausch von Qualitätsmanagern und Qualitätssicherern mit dem Team und deren Mitgliedern ist wertvoller als eine Beurteilung nach Aktenlage.

- Eine kontinuierliche Begleitung über den gesamten Produktelebenszyklus ist zielführender als isolierte Phasen der Qualitätssicherung.
- Eine Validierung der Qualitätsanforderungen und deren Anpassung und Schärfung auch noch während der Entwicklung ist effektiver als eine Vorabspezifikation aller eventuell benötigen Qualitätseigenschaften.
- Eine iterative Anpassung der Methoden und Werkzeuge des Qualitätsmanagements an das jeweilige Vorhaben ist besser als die Verwendung eines allgemeines Methodenkanons für alle Entwicklungen.

ANHANG

A. Qualitätseigenschaften von Softwareprodukten gemäß ISO/IEC 25010

“Functional suitability

This characteristic represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions. This characteristic is composed of the following subcharacteristics:

- **Functional completeness.** Degree to which the set of functions covers all the specified tasks and user objectives.
- **Functional correctness.** Degree to which a product or system provides the correct results with the needed degree of precision.
- **Functional appropriateness.** Degree to which the functions facilitate the accomplishment of specified tasks and objectives.

Performance efficiency

This characteristic represents the performance relative to the amount of resources used under stated conditions. This characteristic is composed of the following subcharacteristics:

- **Time behaviour.** Degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements.
- **Resource utilization.** Degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements.
- **Capacity.** Degree to which the maximum limits of a product or system parameter meet requirements.

Compatibility

Degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions, while sharing the same hardware or software environment. This characteristic is composed of the following subcharacteristics:

- **Co-existence.** Degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product.
- **Interoperability.** Degree to which two or more systems, products or components can exchange information and use the information that has been exchanged.

Usability

Degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. This characteristic is composed of the following subcharacteristics:

- **Appropriateness recognizability.** Degree to which users can recognize whether a product or system is appropriate for their needs.
- **Learnability.** degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use.
- **Operability.** Degree to which a product or system has attributes that make it easy to operate and control.
- **User error protection.** Degree to which a system protects users against making errors.
- **User interface aesthetics.** Degree to which a user interface enables pleasing and satisfying interaction for the user.
- **Accessibility.** Degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use.

Reliability

Degree to which a system, product or component performs specified functions under specified conditions for a specified period of time. This characteristic is composed of the following subcharacteristics:

- **Maturity.** Degree to which a system, product or component meets needs for reliability under normal operation.
- **Availability**. Degree to which a system, product or component is operational and accessible when required for use.
- **Fault tolerance.** Degree to which a system, product or component operates as intended despite the presence of hardware or software faults.
- **Recoverability.** Degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system.

Security

Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization. This characteristic is composed of the following subcharacteristics:

- **Confidentiality.** Degree to which a product or system ensures that data are accessible only to those authorized to have access.
- **Integrity.** Degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data.
- **Non-repudiation.** Degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later.
- **Accountability.** Degree to which the actions of an entity can be traced uniquely to the entity.
- **Authenticity.** Degree to which the identity of a subject or resource can be proved to be the one claimed.

Maintainability

This characteristic represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements. This characteristic is composed of the following subcharacteristics:

- **Modularity.** Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

- **Reusability.** Degree to which an asset can be used in more than one system, or in building other assets.
- **Analysability.** Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.
- **Modifiability.** Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.
- **Testability.** Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met.

Portability

Degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another. This characteristic is composed of the following subcharacteristics:

- **Adaptability.** Degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments.
- **Installability.** Degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment.
- **Replaceability.** Degree to which a product can replace another specified software product for the same purpose in the same environment."

(ISO/IEC 2010)

B. Basisszenario zur Vereinbarkeit von ISO 26262 und Agilität

Beschreibung

Grundsätzlich gibt es verschiedene Möglichkeiten, agile Entwicklung und die Forderungen der ISO 26262 zu vereinbaren. Im Folgenden wird ein mögliches Szenario vorgestellt, das die in Abschnitt 3.2.2 präsentierten Herausforderungen adressiert und in Abbildung 12 dargestellt ist. Das Szenario ist darauf ausgelegt, den gesamten Sicherheitslebenszyklus der ISO 26262 in einem agilen Setting umzusetzen. Für eine solches agiles Setting sollte dennoch zu Beginn eine initiale Hazard and Risk Analysis als Grundlage für die weitere Entwicklung durchgeführt werden. Dies kann in einer initialen Konzeptphase geschehen. Jeder Iteration der agilen Entwicklung deckt anschließend dennoch den gesamten Sicherheitslebenszyklus ab. Die dort vorgeschriebenen Phasen müssen allerdings nicht in der angedachten festen Abfolge durchgeführt werden. Es muss lediglich sichergestellt sein, dass die erforderlichen Maßnahmen Anwendung finden und sich in den entsprechenden Artefakten wiederfinden. Daher müssen in jeder Iteration auch die Ergebnisse der Konzeptphase hinterfragt und eventuell überarbeitet werden, sodass eine neue Einschätzung der Kritikalität eines Inkrements bezüglich der funktionalen Sicherheit definitiv möglich ist. Um diese Umsetzung zu ermöglichen, wird das nachfolgende organisatorische Szenario vorgestellt.

Die Grundidee des Basisszenarios ist es, die volle Verantwortung für das Produkt im agilen Team zu belassen, und dennoch die nötigen Rollen der ISO 26262 abzudecken und insbesondere auch alle Unabhängigkeitsgrade bei den Bestätigungsmaßnahmen zu ermöglichen, so dass der komplette Sicherheitslebenszyklus vom Szenario abgedeckt ist.

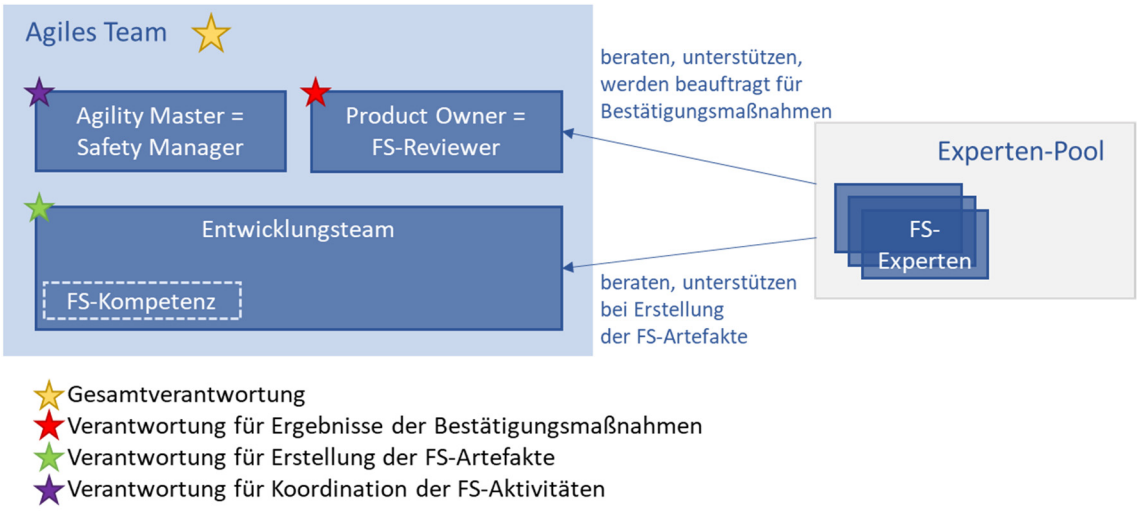

Abbildung 12: Strukturelle Darstellung des Basisszenarios

Das Entwicklungsteam hat die Verantwortung für die Erstellung der sicherheitsspezifischen Artefakte, die von der ISO 26262 gefordert werden. Dazu muss das Entwicklungsteam mit der entsprechenden Kompetenz ausgestattet sein. Es ist durchaus denkbar, dass das Team Unterstützung durch Experten auf dem Gebiet der Funktionssicherheit hinzuzieht. Die grundlegende Kompetenz muss jedoch im Entwicklungsteam vorhanden sein, so dass es die volle Verantwortung für die Erstellung der Artefakte übernimmt.

Die Rolle des Safety-Managers, der in diesem Szenario vom Agility-Master übernommen wird, überwacht, dass die geforderten sicherheitsspezifischen Artefakte erstellt werden und sich das Entwicklungsteam an den Sicherheitsplan hält. Durch die Integration der Safety-Manager-Rolle in das agile Team wird das agile Prinzip der Selbstverantwortung erfüllt. Die organisatorische Verantwortlichkeit des Safety-Managers lässt sich gut mit der Rolle des Agility-Masters vereinen. Der Agility-Master ist grundsätzlich für die Organisation des Entwicklungsteams verantwortlich und kann somit – vorausgesetzt, ein grundsätzliches Verständnis der ISO26262 und vorhandener Kompetenz bezüglich der funktionalen Sicherheit ist vorhanden – den zusätzlichen organisatorischen Aufwand durch die Rolle als Safety-Manager übernehmen. Eine weitere wichtige Rolle, die in der ISO26262 benannt wird, ist der Project-Manager. Dessen Verantwortung lässt sich gut durch die bereits aufgeführte Rolle des Safety-Managers ergänzen. Daher werden in dem dargestellten Szenario beide Rollen in der Person des Agility-Masters vereint. Diese Integration der Rollen ist laut ISO26262 explizit zulässig (ISO/FDIS 26262-2:2018, 6.4.2.4 – NOTE 1).

Während somit der Agility-Master die generelle Durchführung der Sicherheitsaktivitäten verantwortet, wird dem Product-Owner die Verantwortung über die Bestätigungsmaßnahmen zugewiesen. In der ISO26262 gibt es keine explizite Rollenbeschreibung, welche zu den in diesem Ansatz geforderten Verantwortungen passt. Es gibt allerdings einige Rollen, die sich zu dieser neuen Rolle des *FS-Reviewers* zusammenfassen lassen, beispielsweise der *Functional Safety Assessor*. Durch die Zuweisung dieser und verwandter Rollen erhält der FS-Reviewer direkte Verantwortung nicht nur für die Durchführung der Bestätigungsmaßnahmen, sondern auch für deren Ergebnisse. Allerdings muss der Product-Owner diese neuen Aufgaben nicht zwangsläufig selber durchführen, sondern kann Unterstützung aus dem Experten-Pool für funktionale Sicherheit anfordern. Diese Aufteilung der Durchführung, bei gleichzeitig bleibender voller Verantwortung, ist nach der ISO26262 explizit zulässig und beispielsweise in der Beschreibung der bereits genannten Rolle des Functional-Safety-Assessors zu finden. Der FS-Reviewer kann einer anderen Organisationseinheit angehören als alle Mitglieder des Entwicklungsteams, was die Anforderung an dessen Unabhängigkeit erfüllt.

Bewertung des Szenarios in Bezug auf die Herausforderungen

Das vorgestellte Szenario bietet eine agile Lösung für die zuvor beschriebenen Herausforderungen, die hier kurz diskutiert werden.

Kompetenz: Im Basisszenario sind alle Kompetenzen bezüglich der Funktionssicherheit vertreten. Die Entwicklung sicherheitsspezifischer Artefakte wird vom Entwicklungsteam übernommen, die Bestätigungsmaßnahmen vom Product-Owner. Beratung kann zusätzlich von externen Experten in Anspruch genommen werden.

Verantwortung/Unabhängigkeit: Im Szenario sind die Rollen, die gemäß ISO 26262 Verantwortung für die Funktionssicherheit übernehmen, mit angemessener Unabhängigkeit integriert, so dass die Gesamtverantwortung für das entwickelte Produkt vom agilen Team übernommen werden kann. An die Rolle des Safety-Managers werden gemäß ISO 26262 keine Anforderungen bezüglich der Unabhängigkeit vom Entwicklungsteam gestellt. Daher kann der Safety-Manager, alternativ zum Basisszenario, auch innerhalb des Entwicklungsteams verortet sein. Dies kann eine sinnvolle Entscheidung sein, um die Rolle des Agility-Masters nicht mit Aufgaben zu belasten, welche eine zusätzliche, erhebliche fachliche Verantwortung bzw. Entscheidungsgewalt und damit notwendiger FS-Kompetenz erfordern, wie sie durch die Rolle des Safety-Managers gefordert wären.

Phasenorientierter Sicherheitslebenszyklus / Änderungen: Das Basisszenario ist so konzipiert, dass es grundsätzlich die Durchführung aller Aktivitäten und die Entwicklung aller Artefakte des Sicherheitslebenszyklus in jeder Iteration erlaubt. Dadurch ist die Phasenorientierung überwunden. Das bedeutet auch, dass Änderungen von Anforderungen während der Entwicklung kein Problem darstellen. Das agile Team ist selbst für die durchgehende Anpassung aller betroffenen sicherheitsspezifischen Artefakte verantwortlich und besitzt die erforderliche Kompetenz.

Iterationslänge/Teamgröße: Es verbleibt die Herausforderung, dass der Aufwand für die Durchführung oder Anpassung aller erforderlicher Aktivitäten des Sicherheitslebenszyklus längere Iterationen benötigt. Zudem verteilen sich viele erforderliche Kompetenzen auf kleine agile Teams, was in der Praxis selbst bei großzügigem Aufbau von Kompetenzen im Bereich Funktionssicherheit im Unternehmen schwierig sein kann.

Fazit

Das vorgestellte Basisszenario zeigt eine Möglichkeit, wie die Herausforderungen durch Integration von Kompetenz und Verantwortlichkeiten aus dem Bereich der Funktionssicherheit in das agile Team grundsätzlich gelöst werden können. Dadurch kann die geforderte Selbstorganisation des Entwicklungsteams aufrechterhalten und prinzipiell der gesamte Sicherheitslebenszyklus in agilen Iterationen eines agilen Teams durchgeführt werden. Aufgrund der beschränkten Teamgröße und Iterationslänge eignet sich das Szenario vorrangig für kleinere Projekte. In der Automobilindustrie haben Projekte beziehungsweise Produkte aber häufig eine Größe, die von einem einzigen agilen Team nicht gehandhabt werden kann. Daher ist aus praktischer Sicht für die Anwendung in großen Projekten eine Skalierung der Agilität erforderlich, beispielsweise durch die Aufteilung der Entwicklung auf mehrere agilen Teams, die lose oder eng gekoppelt sind (s. dazu Abschnitt 1.9). Außerdem ist es notwendig, eine geeignete Granularität der Entwicklungsumfänge

sicherzustellen, die von den einzelnen agilen Teams verantwortet werden. Bereits die Entwicklung eines Systems (im Sinne der gebräuchlichen Definition eines Automotive OEMs) kann für ein einzelnes agiles Team eine zu komplexe Aufgabe darstellen. Dies bedeutet, dass die Entwicklung von Systemen entsprechend in Funktionen oder sogar Anforderungen heruntergebrochen werden muss. Mit einer solchen hierarchischen Strukturierung der Entwicklungsumfänge wäre eine einhergehende Strukturierung von agilen Entwicklungsteams denkbar. Eine solche Umsetzung der Agilität öffnet allerdings neue Herausforderungen bezüglich der Integration der Iterationsergebnisse auf Systemebene. Diese können jedoch unabhängig von den Herausforderungen der ISO26262 betrachtet werden und werden auch schon im Rahmen von Ansätzen wie „Scaled Agile“ behandelt. Abschließend lässt sich also die Vereinbarkeit von agilen Prinzipien und den speziellen Anforderungen der ISO26262 grundsätzlich bestätigen. Allerdings muss selbstverständlich in jedem Projektkontext zuvor die Praktikabilität der exakten Ausprägung und Implementierung der agilen Ansätze evaluiert werden.

C. Literaturverzeichnis

A-SPICE (2017): Automotive SPICE Process Assessment / Reference Model. Version 3.1. *VDA QMC Working Group 13 / Automotive SIG*. 2017.

Bass, Len / Weber, Ingo / Zhu, Liming (2015): *DevOps: A Software Architect's Perspective*.

Beck, Kent / Beedle, Mike / Bennekum, Arie van / Cockburn, Alistair / Cunningham, Ward / Fowler, Martin / Grenning, James / Highsmith, Jim / Hunt, Andrew / Jeffries, Ron / Kern, Jon / Marick, Brian / Martin, Robert C. / Mellor, Steve / Schwaber, Ken / Sutherland, Jeff / Thomas, Dave (2001a): *Manifesto for Agile Software Development*. http://agilemanifesto.org

Beck, Kent / Beedle, Mike / Bennekum, Arie van / Cockburn, Alistair / Cunningham, Ward / Fowler, Martin / Grenning, James / Highsmith, Jim / Hunt, Andrew / Jeffries, Ron / Kern, Jon / Marick, Brian / Martin, Robert C. / Mellor, Steve / Schwaber, Ken / Sutherland, Jeff / Thomas, Dave (2001b): *Principles behind the Agile Manifesto.* http://agilemanifesto.org/principles.html

Besemer, Frank / Karasch, Timo / Metz, Pierre / Pfeiffer, Joachim (2014): *Clarifying Myths with Process Maturity Models vs. Agile*. White Paper.

Bondar, Arina / Steckinger, Otmar / Dussa-Zieger, Klaudia (2015): "Wie sag ich's meinem Assessor?" *HANSER automotive 11-12 / 2015*, Carl Hanser Verlag, München

CC (2009): *Common Criteria for Information Technology Security Evaluation*. ISO 15408.

Crispin, Lisa / Gregory, Janet (2009): *Agile Testing – A Practical Guide for Testers and Agile Teams*. Addison-Wesley

Deming, W. Edwards (1986): *Out of the Crisis*. The Mit Press

Diebold, Philipp / Zehler, Thomas / Richter, Dominik (2017): "How Do Agile Practices Support Automotive SPICE Compliance?" *ICSSP'17*, July 2017, Paris, France

Foegen, Malte / Kaczmarek, Christian (2016): *Organisation in einer digitalen Zeit*. Zweite Auflage

ISO/FDIS (2018): *ISO/DIS 26262 Road vehicles – Functional safety*. ISO/DIS Draft International Standard, 2018-07

ISO/IEC (2010): *ISO/IEC 25010 System and software quality models*. ISO/IEC

ISO/SAE (2017): *ISO/SAE AWI 21434 Road Vehicles – Cybersecurity engineering*, (Standard in Entwicklung), angekündigt unter https://www.iso.org/standard/70918.html, 2017.

Johnson, Jim (2002): *ROI, It's Your Job*. Third International Conference on Extreme Programming, Alghero, Italien

Larman, Craig / Vodde, Bas: Introduction to LeSS (2018). https://less.works/less/framework/introduction.html

McConnell, Steve (2006): *Software Estimation. Demystifying the Black Art*. Redmond, Washington

Pink, Daniel H. (2009): *Drive – The surprising truth about what motivates us*. Riverhead Books, New York

Rupp, Chris / die SOPHISTen (2009): *Requirements-Engineering und -Management*. 5. Auflage. Carl Hanser Verlag München Wien

Safety Research & Strategies, Inc. (2013): *Toyota Unintended Acceleration and the Big Bowl of "Spaghetti" Code*. http://www.safetyresearch.net/blog/articles/toyota-unintended-acceleration-and-big-bowl-%E2%80%9Cspaghetti%E2%80%9D-code

Schwaber, Ken (2013): *UnSAFe at any speed*. https://kenschwaber.wordpress.com/2013/08/06/unsafe-at-any-speed/

Schwaber, Ken / Beedle, Mike (2002): *Agile Software Development with Scrum*. Prentice Hall.

Schwaber, Ken / Sutherland, Jeff (2017): *The Scrum Guide – The Definitive Guide to Scrum: The Rules of the Game.*

Stacey, Ralph D. (2000): *Strategic Management and Organisational Dynamics: the challenge of complexity*.

Thomas, Dave (2015): *Agile is Dead*. GOTO Conference 2015

VDA (2009): *Produktentstehung – Reifegradabsicherung für Neuteile*. VDA-Blaugold-Band, 2. überarbeitete Auflage, Oktober 2009

VDA (2017): *Automotive SPICE® - Guidelines*. VDA-Blaugold-Band, 1. Auflage, September 2017

Wake, Bill (2003): *INVEST in Good Stories, and SMART Tasks*. https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Wells, Don (1999): *Pair Programming*. http://www.extremeprogramming.org/rules/pair.html

West, Dave (2011): *Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today*.