Gesundheits-Apps

Moderne Softwareentwicklung und Gesundheits-Apps: Ein Widerspruch?

Dr. Alexander Bunkenburg, Dr. Joachim Neumann, Barcelona und Dipl.-Ing. Arne Briest, Karlsruhe*

Gesundheits-Apps sind Programme, die auf Smartphones ausgeführt werden um Krankheiten zu erkennen, zu verhüten, oder zu lindern. Da viele Patienten ein Smartphone haben und dieses stets bei sich tragen, können Gesundheits-Apps sowohl dem Patienten helfen, als auch dazu beitragen, Kosten im Gesundheitswesen zu senken. In diesem Artikel gehen wir auf die rechtlichen und technischen Herausforderungen bei der Entwicklung von Gesundheits-Apps ein und zeigen Wege auf, Methoden moderner Softwareentwickung konform mit den rechtlichen Anforderungen einzusetzen. Für Hersteller von Gesundheits-Apps sprechen wir konkrete Empfehlungen aus und zeigen Methoden auf, welche in die Standardvorgehensweisen des Qualitätsmanagementsystems aufgenommen werden sollten.

1. Rechtliche Anforderungen an eine Gesundheits-App

Die Entwicklung und Vermarktung von Gesundheits-Apps wird durch verschiedene Rechtsgebiete abgedeckt[1]. Zunächst ist zu klären, ob es sich bei der App um ein Medizinprodukt handelt, d. h. es gilt die Zweckbestimmung im Sinne des Medizinproduktegesetzes[2] (MPG) oder internationalen geltenden Regularien zu definieren[3]. Dient die App der Therapie oder Diagnose einer Erkrankung (medizinische Bestimmung), dann muss sie den regulatorischen Anforderungen des Medizinprodukterechtes bei seiner Inbetriebnahme entsprechen. Die sich anschließende Frage ist die nach der Produkthaftung für Schäden, die durch die App verursacht werden können. Im Falle einer Verwendung von Cloud-Servern und externer Datenbearbeitung sind das Datenschutz[4]- und Telemediengesetz[5] zu beachten. Sollte die App von einem Pharma- oder Medizintechnikunternehmen vertrieben werden, ist das Heilmittelwerberecht[6] zu berücksichtigen. Daraus leitet sich die Forderung ab, dass die vertragliche Zusammenarbeit zwischen dem Vertreiber, Nutzern, Datenbankanbieter, und Hersteller detailliert frühestmöglich in Entwicklungsprojekten zu klären sind.

Apps können nach der Medizinprodukte-Richtlinie 93/42/EG inklusive der Anhänge 2007/47/EG als eigenständiges Medizinprodukt *(stand alone software)* eingestuft werden. Somit muss eine App die grundlegenden Anforderungen der Medizinprodukte entsprechen und die Sicherheit der App unter Zuhilfenahme von europäischen harmonisierten Standards und von klinischen Daten nachgewiesen werden. Die Klassifizierung der App richtet sich nach der Zweckbestimmung im Sinne des MPG. Die App unterliegt dann auch der Kennzeichnungspflicht nach MPG, welche durch eine elektronische Gebrauchsanweisung in der App erfüllt werden können.

Vor Inverkehrbringen der App muss der Hersteller ein Konformitätsbewertungsverfahren abschließen. Zum Nachweis der Produktlebenszyklen und gleichbleibender Qualität muss der Hersteller ein Qualitätsmanagementsystem installieren. Das Qualitätsmanagementsystem wird von dem Team erstellt, welches für *Quality Assurance* und *Regulatory Affairs* zuständig ist (QA/RA-Team), und muss den gesamten Produktlebenszyklus der App vom Start der Entwicklung bis zur Marktbeobachtung abdecken.

Die US-Behörde *Food and Drug Administration* (FDA) hat erfreulicherweise am 25. September 2013 den Leitfaden *„Mobile Medical Applications"* veröffentlicht[7]. Dieser beschreibt, welche App als Gesundheits-App angesehen wird und wie diese in den USA reguliert ist. Auch wenn diese Regulierung zunehmend Sicherheit bei der Projektabwicklung schafft, bleiben viele Detailfragen bezüglich der Entwicklung von Gesundheits-Apps unbeantwortet. Wir hoffen im Folgenden einige dieser Fragen beantworten zu können.

2. Besonderheiten der Smartphone-Plattform

Gesundheits-Apps, die auf einem handelsüblichen Smartphone laufen, unterscheiden sich in vielen Aspekten von *Desktop-software,* die auf einem Rechner ausgeführt wird. Die im Folgenden erwähnten Smartphone-spezifischen Aspekte müssen in der Gebrauchstauglichkeitsorientierten Entwicklung und bei der Anwendung des entsprechenden Standards IEC 62366 berücksichtigt werden:

1. Der kleine Bildschirm eines Smartphones stellt spezielle Anforderungen an das Design der grafischen Benutzeroberfläche

einer App: es ist unmöglich, immer alle wichtigen Schaltflächen und Informationen gleichzeitig darzustellen. Die Spezifikation und Risikoanalyse eines Medizinproduktes muss daher darauf eingehen, welche Funktionalität der App dem Anwender zu welchem Zeitpunkt zur Verfügung steht. Um während der Planung der Benutzeroberfläche Fehler zu vermeiden, kann ein Testen von Papier-Schablonen mit potentiellen Anwendern hilfreich sein. Neben der Navigation der App müssen natürlich auch die Größe und der Kontrast der Schaltflächen den Anforderungen der Anwender entsprechen, z. B. im Falle einer Sehbehinderung. In der klinischen Betrachtung der App sollte auch auf die Besonderheiten des Umgangs von Kindern und älteren Menschen mit einem Smartphone eingegangen werden.

2. Im Gegensatz zu Software, die auf einem Rechner ausgeführt wird, werden Apps oftmals unter Umständen benutzt, die der Entwickler nicht vorhergesehen hat, z. B. gleichzeitig mit der Ausübung einer anderen Tätigkeit. Der gezielte Einsatz von Beta-Testern kann helfen, diese Situationen und daraus entstehende Risiken aufzuspüren.
3. Die begrenzte Kapazität der Batterie von Smartphones erfordert spezielle Maßnahmen, um den Stromverbrauch zu begrenzen. Weiterhin müssen Maßnahmen spezifiziert werden, die bei geringem Ladestand der Batterie aktiviert werden (Warnung an den Anwender, Speicherung wichtiger Daten, etc.). Bei der Spezifikation des Laufzeitverhaltens ist darauf einzugehen, wie sich die App im Hintergrundbetrieb verhält, z. B. wenn der Anwender die App mit dem *Home Button* schließt.
4. Smartphones sind in der Regel ständig mit dem Internet verbunden. Dies erlaubt einen Datenaustausch zwischen der Gesundheits-App und einem Server der Herstellers, um die Gesundheits-App mit zusätzlichen Features zu bereichern:
   - Gesundheits-Apps können explizit Daten über ihre Anwender erheben, z. B. in Form eines Fragebogens, der beim ersten Öffnen der App vom Anwender ausgefüllt wird.
   - Ein Arzt kann die App zur Fernüberwachung eines Patienten nutzen, wenn diese regelmäßig Daten protokolliert und der Hersteller diese Daten dem Arzt zur Verfügung stellt.
   - Die Internetverbindung erlaubt den Einsatz spezieller Softwarebiblioteken (z. B. *TestFlight*[8] oder *Hockey*[9]), mit denen der Entwickler ohne explizite Mitwirkung des Anwenders beobachten kann, wie die App vom Anwenden genutzt wird und unter welchen Bedingungen Fehler oder gar Abstürze auftreten. So kann frühzeitig erkannt werden, falls die App von den Anwendern anders als vorhergesehen genutzt wird. Diese Bibliotheken sind insbesondere sinnvoll, wenn Beta-Tests in den Entwicklungsprozess einbezogen werden.
   - Die Risikoanalyse muss berücksichtigen, dass die Internetverbindung eines Smartphones oft instabil ist und auch ganz ausfallen kann. Die Gesundheits-App muss daher spezifizieren, welche Funktionalität auch bei fehlender Internetverbindung zur Verfügung steht. In der Regel empfiehlt es sich, in Falle einer fehlenden Internetverbindung wichtige Daten zwischenzuspeichern, um diese zu einem späteren Zeitpunkt an den Server des Herstellers der App zu übermitteln.
5. Bei der Herstellung einer Gesundheits-App, die auf einem Smartphone ausgeführt wird, ist stets ein Kompromiss zu suchen zwischen dem Datenschutz des Anwenders und dem Vorteil, den ein Datenaustausch zwischen der App und den Servern des Herstellers bietet. Hierbei ist zu beachten, dass eine vollständige Anonymisierung des Anwenders einer Gesundheits-App sich schwierig gestalten kann, da der Anwender auch indirekt identifiziert werden kann, z. B. über zu den Abgleich der IP-Nummern bei Benutzung der Gesundheits-App mit den IP-Nummern bei Benutzung anderer Apps auf dem gleichen Gerät, welche die Identität des Anwenders preisgeben können.
6. Gesundheits-Apps, die Audio- oder Videoinhalte wiedergeben, müssen eine eventuelle Verzögerung dieser Signale aufgrund des Betriebssystems des Smartphones berücksichtigen. Diese können im Einzelfall zu einer Beeinträchtigung der Verständlichkeit des Inhalts führen. Insbesondere bei gleichzeitiger Darbietung von akustischen und visuellen Signalen in der Telemedizin ist auf die relative Verzögerung zwischen dem Video- und Audiokanal zu achten.
7. Falls Sensoren des Smartphones wie der Beschleunigungssensor, Gyroskop, Kompass, eingesetzt werden, müssen die Fehler dieser

Sensoren samt Fehlerfortpflanzung berücksichtigt werden.

8. Bei der Entwicklung einer App werden in der Regel zusätzliche Zulieferer relevant:
   - der Hersteller der Smartphones und eventuell der Hersteller speziellen Zubehörs
   - der Hersteller des Betriebssystems des Smartphones (iOS, Android, Windows Phone, oder BlackBerry OS)
   - der Hersteller der Entwicklungswerkzeuge (z.B *Xcode* von Apple, *Android Studio* von Google[10], *Microsoft Visual Studio* oder RIM's JDE)
   - falls Softwarebibliotheken von Drittanbietern zum Einsatz kommen, ist dies in der Risikoanalyse zu berücksichtigen
9. Für eine App sind Hygiene und Transport nur scheinbar irrelevant. Der Hersteller der App muss sicherstellen, dass die App beim Benutzer unverändert ankommt. Dies wird durch digitales Signieren der kompilierten App ermöglicht, welches bereits Bestandteil der App Stores aller führenden Hersteller ist. Der Transport der ausführbaren Binärdatei von der Produktionsstätte – oftmals ein Rechner in der QA Abteilung – zum Smartphones des Anwender ist so unproblematisch, da vor einer Installation der App auf dem Smartphone die digitale Signatur der Binärdatei geprüft wird und das Betriebssystem des Smartphones die Installation einer veränderten Binärdatei verhindert.
10. Die Implementierung eines eventuellen Rückrufs einer Gesundheits-App erfordert eine gewissenhafte Planung. Ein Kunde, der einer App in einem App Store erwirbt, unterliegt in der Regel nicht der direkten Kontrolle des Herstellers der App. Daher muss der Hersteller für eine direkte Kommunikation zu dem Anwender sorgen. Mechanismen, die von Smartphone Betriebssystemen zur Verfügung gestellt werden – wie z. B. die Mitteilungen der *Mail* App in iOS beim Eintreffen einer Email – sind oftmals nicht dazu geeignet, einen Rückruf zu implementieren. Hier muss der Hersteller prüfen, ob sich diese Mitteilungen in den Einstellungen des Smartphone abstellen lassen. Falls dies der Fall ist, muss ein unabhängiger Kommunikationsmechanismus entwickelt werden, der sich vom Anwender nicht abstellen lässt. Eine solche Möglichkeit bietet ein spezieller, vom Hersteller zu implementierender *Web Service,* der von der App regelmäßig abgefragt wird.
11. Neue Versionen einer Gesundheits-App können über die App Stores bequem vertrieben werden. Ein Problem ist allerdings, dass kein Mechanismus vorgesehen ist, um die Installation einer neuen Version zu erzwingen: ein *upgrade* auf eine neue Version erfolgt immer freiwillig. Der Hersteller einer Gesundheits-App kann allerdings den oben beschriebenen Rückrufmechanismus anwenden, um Anwender zum Installieren einer neuen Version zu bewegen.

3. Problemfeld Werkzeuge und Mentalität moderner Softwareentwicklung

Eine Besonderheit von Apps wird leider oftmals vergessen, insbesondere wenn die Entwicklung der App von einem externen Softwarehaus durchgeführt wird: Die Mentalität der Softwareentwickler von Apps. Die oben angeführten rechtlichen Anforderungen, welche seitens des Risikomanagements (ISO 14971), des Qualitätsmanagementsystems (ISO 13485) und dem Standard zur Softwareentwicklung (IEC 62304) an eine Gesundheits-App gestellt werden, scheinen die strenge Anwendung des Wasserfallmodells[11] oder des V-Modells[12] nahezulegen, in denen jede Entwicklungsphase

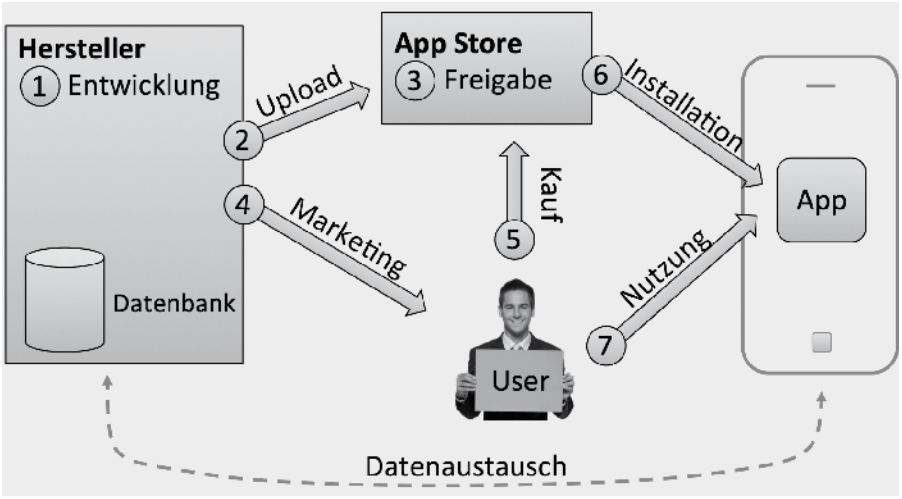

Abb. 1: Eine Gesundheits-App wird wie jede andere Smartphone-App über einen App Store vertrieben. Dies erhöht die zum Vertrieb notwenigen Schritte und erlaubt es dem Hersteller nur indirekt mit dem Anwender zu kommunizieren.

abgeschlossen sein muss, bevor die nächste Phase begonnen wird. So verlangt diese Vorgehensweise, dass die Spezifikation der App vollständig vorliegen muss, bevor die erste Zeile Quellcode geschrieben wird. Obwohl diese Modelle großen Wert auf Softwareverifizierung und -validierung legen, wird diese starre Vorgehensweise heutzutage von vielen Softwareentwicklern als schlecht angesehen, da sie wenig Spielraum für Änderung lässt. Ein solcher Spielraum für Änderungen ist wichtig, weil im Laufe der Zeit bei fast jeder Softwareentwicklung mehr Informationen und Erfahrungen zur Verfügung stehen. Dies erfordert fast immer eine Anpassung der Produktspezifikation.

Bei Smartphone-Apps, die nicht als Gesundheits-Apps reguliert werden, stehen Entwickler in der Regel unter dem Druck, ihre App frühzeitig zu vermarkten. Ein Entwickler, der zum Zeitpunkt der Vermarktung mit seiner Software zufrieden ist, wird manchmal kritisiert zu lange gezögert zu haben. Hinter dieser Strategie steht der Erkenntnis, dass sich bereits mit einer fehlerhaften oder unvollständigen frühen Version der App wichtige Rückmeldungen von Anwendern sammeln lassen, welche dann, helfen folgende Softwareversion zu verbessern. Erstaunlicherweise wird diese Herangehensweise in der Smartphone-Plattform oftmals toleriert, sofern die Anwender mit regelmäßigen *Updates* belohnt werden. Diese Einstellung wird von Softwareentwickler unter dem Motto *release soon, release often*[13] zusammengefasst. Entwickler von Smartphone-Apps werden daher oftmals zögern, wenn der Hersteller einer Gesundheits-App fordert, die App zunächst vollständig zu spezifizieren, bevor mit der Umsetzung angefangen wird.

Ein weiteres Problemfeld ist die Wahl der eingesetzten Entwicklungswerkzeuge: aufwendige Projektmanagementsoftware, das nicht-iterative Vorgehen des Wasserfallmodells, auf dem V-Models basierte Entwicklungszyklen oder Produktspezifikationen in der *Unified Modeling Language*[14] (UML) werden in der Welt der Smartphone-Apps ebenfalls oftmals als unflexibel und bürokratisch abgelehnt.

Für den Hersteller einer Gesundheits-App stellt sich somit die Herausforderung, die Mentalität moderner Softwareentwicklung mit den Anforderungen zu vereinbaren, welche Seitens des Risikomanagements (ISO 14971), der Gebrauchstauglichkeit (IEC 62366), des Qualitätsmanagementsystems (ISO 13485) und des Standards zur Softwareentwicklung (IEC 62304) gestellt werden. Wie wir im Folgenden aufzeigen, haben sich jedoch gerade durch die Anforderungen der schnellen Produktzyklen eine Reihe von modernen Methoden verbreitet, die für die Entwicklung einer Gesundheits-App äußerst dienlich sind, da sowohl die rechtlichen Anforderungen als auch der moderne Softwareentwickler ein gemeinsames Ziel haben: eine hohe Produktqualität.

4. Wie können moderne Methoden helfen, die regulatorischen Anforderungen von IEC 62304 zu erfüllen?

Die Entwicklung medizinischer Software nach IEC 62304 beschreibt vier Phasen bei der Softwareentwicklung und jeder Übergang in die nächste Phase erfordert die Erfüllung gewisser Kriterien und muss verantwortlich abgezeichnet werden:

- *Analyse.* Während der Analyse wird noch nicht programmiert.
- *Entwicklung.* Nach Abschluss der Analyse kann das Projekt in die Entwicklungsphase treten. Hier findet ein Großteil der Softwareentwicklung und der Tests statt.
- *Marktfreigabe.* Zu diesem Zeitpunkt muss das Produkt muss vollständig entwickelt, beschrieben, validiert, und verifiziert sein.
- *Marktbeobachtung.* Das Produktumfeld (ähnliche Produkte, neue Erkenntnisse der Forschung, usw.) müssen beobachtet werden, um eventuelle Risiken, die von dem Produkt ausgehen könnten, frühzeitig erkennen zu können. Ferner müssen alle Rückmeldungen von Anwendern in das *Corrective And Preventive Actions* (CAPA) System des Herstellers eingepflegt werden.

Bei der Umsetzung dieser Phasen räumt IEC 62304 dem Hersteller einen gewissen Freiraum ein, sofern die Umsetzung schlüssig begründet werden kann. So kann sich der Hersteller entscheiden, ein Qualitätsmanagementsystem aufzubauen, in dem die Analysephase recht kurz gehalten wird, und nur eine erste Version der Spezifikation der App erstellt wird. Die in der Standardvorgehensweise beschriebene verkürzte Analysephase muss aber dennoch mindestens folgende Bereiche abdecken: erste Spezifikation der App, medizinischer Nutzen für den Anwender, eventuelle Anbindung an das Internet, eventuelle Rolle von Ärzten, Risikoanalyse, das rechtliche Umfeld, und die wirtschaftlichen Chancen des Produktes.

Nach dem definierten und verantwortlich abgezeichneten Übergang zur Entwicklungsphase kann die Standardvorgehensweise nun eine Softwareentwicklung in kurzen Iterationen vorsehen. In jeder Iteration lernen wir aus den in den vorhergehenden

Iterationen gewonnenen Einsichten und verbessern das Produkt. Diese Einsichten können technischer oder medizinischer Art sein. Während der Entwicklungsphase ist es allerdings nicht mehr möglich, die generelle Richtung der App zu ändern. Eine solche grundlegende Änderung des Konzepts einer App ist bei Online-Spielen und sozialen Apps auch zu einem späten Zeitpunkt nicht ungewöhnlich, kann aber bei Gesundheits-Apps nicht akzeptiert werden, insbesondere in Hinblick auf die Risikoanalyse.
Offensichtlich ist hierbei zwingend erforderlich, dass Softwareentwickler sich den Anforderungen der Qualitätssicherung bewusst sind und eng mit der Qualitätssicherung zusammenarbeiten, z. B. durch die Gewährleistung der Rückverfolgung.
Viele moderne Entwicklungsmethoden und Werkzeuge versuchen durch eine ständige Rückverfolgbarkeit die Qualität der Software zu erhöhen. Diese Werkzeuge sind daher für die Entwicklung von Gesundheits-Apps äußerst vorteilhaft. Neben der Identifizierbarkeit jeder Aktion erhöht die Rückverfolgbarkeit darüberhinaus auch indirekt die Qualität des Entwicklungsprozesses, da sich jeder Entwickler seiner erhöhten persönlichen Verantwortung bewusst ist. Im Folgenden erwähnen wir *Ticketing*-Systeme, digitale Unterschriften, Wikis, und Quellcode-Versionierung als Beispiele für Werkzeuge, die eine bessere Rückverfolgbarkeit ermöglichen.
In den letzten Jahren ist es üblich geworden, die Entwicklungsarbeit mit *Tickets* zu organisieren. Das Produkt *JIRA*[15] von der Firma Atlassian ist hier marktführend. Ein Ticket ist ein virtuelles Dokument, welches einen Entwicklungsschritt beschreibt und leicht auf einer Webseite suchbar und editierbar ist. Dieser Schritt kann eine Fehlerkorrektur *(bug fix)* sein oder die Erstellung einer neuen Funktion *(feature request)*. In einem Ticket werden alle Informationen zu diesem Entwicklungsschritt beschrieben. Ein Ticket hat zu jedem Zeitpunkt ein definierten Status und muss einen Ablauf durchlaufen, der in den Standardvorgehensweisen des Qualitätsmanagementsystem definiert ist. Der Status-Ablauf für einen *bug fix* könnte zum Beispiel sein: Entdeckung des Fehlers, Diagnose, Erzeugen einer Verzweigung des Quellcodes, Korrektur des Quellcodes, Überprüfung der Code-Änderung, automatische und manuelle Test, und schließlich die Einführung der Code-Änderung in das Produkt. Jedes Ticket ist einer Person zugeordnet, die dafür verantwortlich ist, das Ticket in den nächsten Status weiterzubringen. Das Ticket-System behält die gesamte Geschichte des Tickets. Jeder Schritt und jede hinzugefügte Information kann mit Datum und Name auf die Person zurückverfolgt werden, die diese Änderung vorgenommen hat. Wir können so immer sehen: Wer hat dieses Ticket geschrieben? Wer hat entschieden, dass es entwickelt werden soll? Wer hat die Spezifikation für diese Funktionalität geschrieben? Wer hat die Funktionalität entwickelt? Die Tickets zeigen somit im Detail die Geschichte der Entwicklung und entledigen den Entwickler von der Aufgabe, die Entwicklung in einem getrennten System zu dokumentieren.
Im Falle von *Ticketing*-Systemen erfolgt die Identifizierung der Person, die eine Änderung vorgenommen hat, in der Regel durch ein Passwortgeschütztes Login. Sinnvollerweise werden Rollen vergeben, die mit bestimmten Befugnissen verknüpft sind. So kann z. B. sichergestellt werden, dass nur bestimmt Personen die Spezifikation ändern können.
Es ist durchaus nützlich, die Spezifikation der App in das Ticketing-System einzupflegen. So können z. B. entsprechende Tests samt ihrer Akzeptanzkriterien bequem aktualisiert werden und sie sind den Entwicklern leicht zugänglich. Da Änderungen an der Spezifikation einer Gesundheits-App weitreichende Folgen haben können sollte der Personenkreis, der Änderungen an der Spezifikation vornehmen kann, klein gehalten werden. Dies muss in den Standardvorgehensweisen des Qualitätsmanagementsystem definiert sein.
Im Falle von Dokumenten, die mit herkömmlicher Textverarbeitung geschrieben werden, kann eine *digitale Unterschrift* Arbeitsabläufe vereinfachen, insbesondere wenn eine Änderung an einem Dokument von einem Mitarbeiter genehmigt werden soll, der nicht vor Ort ist. Digitale Unterschriften können mit Hilfe eines *root certificate* erstellt werden, welches erlaubt mit dem frei erhältlichen *Acrobat Reader* Dokumente digital zu unterzeichnen. Acrobat Reader kann auch genutzt werden, um die Gültigkeit aller digitaler Unterschriften eines Dokuments zu überprüfen. Ein root certificate kann ohne Kosten durch ein *self-signed certificate* oder gegen Bezahlung mittels einer *Root Certificate Authority* erstellt werden. Zur Vorbereitung eines Dokumentes zur Unterschrift gibt es viele Möglichkeiten, die Software Adobe Acrobat Pro ist eine Möglichkeit.
Oft interessiert uns allerdings nicht, wie die geschichtliche Genesis einer Funktion war, sondern wir wollen wissen, wie genau ist die Funktion zum gegenwärtigen Zeitpunkt aussieht. Oder, wir wollen wissen, wie genau sie vor drei Wochen war, als bei einem Anwender ein Fehler aufgetreten ist. Wir brauchen also zusätzlich zu den Tickets eine genaue vollstän-

dige Spezifikation der App, wie in einem Buch. Dieses Buch soll ebenfalls leicht zugänglich und suchbar sein, aber auch leicht editierbar. Zusätzlich sollte es auch versioniert sein, damit wir leicht sehen können, wie das Buch zu einem beliebigen Zeitpunkt in der Vergangenheit aussah. So eine Funktionalität bieten moderne Wikis. Wikis sind Webseiten, die leicht editiert werden können, und die untereinander verlinkt sind. In solchen Wikis kann die Information sehr leicht editiert werden, auch von mehreren Personen gleichzeitig. Es können relativ leicht Diagramme und Bilder eingefügt werden. Auch bei Wikis, wie bei Tickets, bleibt jede Änderung in der Geschichte erhalten. Wir können später immer suchen und nachweisen, wer wann welche Änderung gemacht hat. Es gibt mehrere Wiki-Produkte; eines ist *Confluence* von Atlassian.

Mit Tickets und Wikis haben wir Rückverfolgbarkeit für einzelne Entwicklungsschritte und für die gesamte Spezifikation des Produktes erreicht. Wir brauchen allerdings auch noch die Rückverfolgbarkeit für den Quellcode. Wir müssen herausfinden können: Wer hat diese Code-Änderung gemacht? Wann hat er sie gemacht? Warum hat er sie gemacht? Wie sah zu der Zeit der Rest des Codes aus? Wie sah zu diesem Zeit die Spezifikation aus? Und natürlich möchten Entwickler zu Testzwecken den vollständigen Stand des Quellcodes wiederherstellen können, die zu einer bestimmten Version der App gehören. Diese Fragen können moderne Werkzeuge der Quellcode-Versionierung beantworten. Hier nennen wir stellvertretend für einige gute moderne Systeme nur *git*[16] von Linus Torvalds, welches bei neuen Projekten der Marktführer ist. Git protokoliert jede Code-Änderung zusammen mit dem Zeitpunkt der Änderung, dem Namen des Entwicklers und dem exakten Zustand des gesamten Quellcodes. Wenn wir eine gute Quellcode-Versionierung zusammen mit einem Ticket-System und einem Wiki für die Spezifikation benutzen, können wir auch feststellen, was der Entwickler machen wollte, an welchen Ticket er arbeitete, und wie die gesamte Spezifikation zu diesem Zeitpunkt aussah.

Git ermöglichen es auch, einem anderen Entwickler die Code-Änderung übersichtlich darzustellen und eine Prüfung des Quellcodes durch einen anderen Entwickler obligatorisch zu machen. Git kann den Code, der von einem Entwickler verändert wird, von dem zentralen Code *(trunk)* trennen. Die meisten Entwickler in der Entwicklungsabteilung sind nicht befugt, ihre Code-Äderungen im *trunk* auszuführen, sondern arbeiten in einer Verzweigung des Quellcodes *(branch).* Der Antrag zur Überführung einer Code-Änderung in den *trunk* wird in Entwicklerjargon „*pull request*" genannt. Ein anderer Entwickler, meist mit mehr Erfahrung, überprüft die Änderung und lehnt sie ab, wenn sie nicht gut ist, oder akzeptiert sie und fügt sie in den *trunk* ein. Alle *pull requests* sind protokoliert und suchbar, und machen dadurch sowohl die Änderung als auch ihre Kontrolle des Quellcodes rückverfolgbar.

Moderne Entwicklungsmethoden empfehlen ein frühes automatisches Testen des Produktes. Alle Teile des Programmes sollten in ihrer Funktion überprüft werden. So ein teilweiser Test heißt „unit test". Weiterhin kann es auch Tests des ganzen zusammengesetzten Programms geben, und Tests der Benutzeroberfläche. Ein Test ist nicht Teil der späteren App, aber er wird vom Programmierer früh erstellt. Die beliebte *Test-Driven-Development* Methode erfordert sogar, zuerst den Test zu schreiben. Dieser muss zunächst natürlich fehlschlagen. Erst wenn die eigentliche Funktionalität implementiert ist, kann der Test ein positives Ergebnis haben.

Einige besonders überzeugte Vertreter des *Test-Driven-Development* gehen so weit, zu sagen, diese Tests reichen als Spezifikation des Programmes aus. Für die Spezifikation eines Medizinproduktes reichen Tests als Spezifikation natürlich nicht aus. Dennoch erleichtern Tests die Arbeit des QA-Teams und der eine oder andere Test kann bei der Erstellung einer detaillierten Produktspezifikation helfen und eventuell sogar ein vernünftiges Akzeptanzkriterium darstellen.

Es ist wichtig, dass die Tests von jedem Programmierer zu jedem beliebigen Zeitpunkt und ohne großen Mehraufwand ausführbar sind. Dazu gibt es sogenannte *Continuous-integration-server,* wie zum Beispiel *Jenkins, Bamboo* von Atlassian oder das in Xcode version 5 integrierte *Continuous Integration.* Jedes Mal wenn die Quellcode-Versionierung eine Code-Änderung registriert, wird der gesamten Code samt der neuen Änderung überprüft. Hierzu werden automatisch alle existierenden Tests ausführt. Dies erfolgt auf einem speziellen Rechner im Hintergrund, während die Entwickler bereits an einer neuen Funktion arbeiten kann. Schlägt der Test fehl, wird die Code-Änderung von der Quellcode-Versionierung abgelehnt, und der Entwickler wird aufgefordert, den Code (oder den Test) zu berichtigen. Diese Tests sind meist recht umfassend und erlauben es die Software in verschiedensten Konfigurationen parallel zu testen. Smartphone-Apps

können zum Beispiel mit einer Vielzahl verschiedener Modelle und Versionen des Betriebssystems getestet werden und Entwickler bekommen recht früh eine genaue Rückmeldung, wenn eine ihrer Codeänderungen in einer speziellen Konfiguration einen Test nicht besteht. Zusätzlich erlauben es automatisierte Tests die Benutzeroberfläche *(graphical user interface testing)* zu testen. In ihnen wird die Interaktion des Anwenders mit der Software simuliert. Gerade im Bereich von Smartphone-Apps sind diese automatisierten Tests sehr nützlich. In der Praxis ist es aber oft unmöglich, eine App vollständig zu testen.
Tests führen das kompilierte Binärprogramm aus und können Fehler nachweisen. Einen anderen Ansatz verfolgt die statische Code-Analyse. Statische Code-Analyse identifiziert Stellen im Quellcode, die mit hoher Wahrscheinlichkeit Programmierfehler beinhalten, die Abstürze oder unvorhergesehenes Verhalten zur Folge haben können. Statische Code-Analyse wird von einem speziellen Programm durchgeführt, das den Quellcode liest und bestimmte Eigenschaften des Programmtextes sucht, die auf Fehler hindeuten können. Einfache Eigenschaften wären zum Beispiel eine Variable, die ausgelesen wird, ohne dass wir ihr vorher ein Wert zugewiesen haben. Ein anderes Beispiel ist ein Ast des Programmes, der nie erreicht werden kann. Man könnte einwenden, dass eine statische Code-Analyse im Gegensatz zu einem menschlichen Gehirn den Quellcode nicht wirklich durchdenken kann. Dennoch zeigt die Praxis, dass statische Code-Analyse zum Teil große Zusammenhänge und die Abhängigkeiten der Code-Teile untereinander verstehen kann, und Fehler finden kann, die der Entwickler übersehen hat. Die Überprüfung durch statische Code-Analyse kann entweder während des Programmierens im Editor (wie Xcode oder Eclipse) stattfinden oder als Teil der *Continuous Integration* dem Entwickler Rückmeldungen geben.

5. Ein Wort der Warnung

Trotz des recht positiven Bildes, welches wir für moderne Softwareentwicklungswerkzeuge in der Entwicklung von Gesundheits-Apps zeichnen, möchten die Autoren dennoch ausdrücklich davor warnen, alle Gepflogenheiten der neueren App-Entwicklung positiv zu bewerten. Entgegen der Politik des *Release Early* muss während der Entwicklung einer Gesundheits-App vermieden werden, dass Beta-Testern einem unverantwortlichen Risiko ausgesetzt werden. In der Regel sollte sich das Beta-Testen auf die Gebrauchstauglichkeitprüfung beschränken, während das Testen der klinischen Funktionalität erst im Rahmen einer klinischen Evaluierung erfolgen kann, welche unter anderem eine abgeschlossene Risikoanalyse sowie abgeschlossene Softwareverifizierung und -validierung voraussetzt.
Eine enge Zusammenarbeit des QA/RA-Teams mit den Entwicklern ist unbedingt erforderlich. Viele der in diesem Artikel vorgestellten Werkzeuge können sowohl von Softwareentwicklern, von QA/RA Mitarbeitern und auch vom Projektmanagement gleichzeitig eingesetzt werden. Dies führt zu Transparenz, Rückverfolgbarkeit, und letztlich zur Erhöhung der Produktqualität.

6. Zusammenfassung

Wir haben zeigen können, dass die rechtlichen Anforderungen, die an die Entwicklung einer Gesundheits-App gestellt werden, sich mit der Mentalität und den Methoden moderner Softwareentwicklung vereinbaren lassen, sofern der Hersteller diese Methoden in die Standardvorgehensweisen seines Qualitätsmanagementsystem integriert und von gewissen Praktiken Abstand nimmt.

**Anschriften der Verfasser:*
Dr. Alexander Bunkenburg forschte an der Universität Glasgow über formale Methoden der Software-Entwicklung. Er hat in verschiedenen Firmen als leitender Software-Architekt im Umfeld Java-Internet und Smartphone Apps gearbeitet.
Melcior de Palau 103, 3, 1
08014 Barcelona, Spanien
alex@inspiracio.cat

Dr. Joachim Neumann, VISAMED GmbH, arbeitet seit 15 Jahren in der Hörgeräteindustrie und ist entwickelt seit 2013 Gesundheits-Apps, die den regulatorischen Anforderungen für eine CE-Kennzeichnung erfüllen.
Clot 76, 3 r
08018 Barcelona, Spanien
joachim@visamed.com
+34 645249886

Dipl.-Ing. Arne Briest, VISAMED GmbH arbeitet seit 15 Jahren in der Medizinprodukteindustrie in Europa und den USA in den Bereichen Produktentwicklung, Zulassung und Qualitätssicherung. Er hat Medizinprodukte weltweit zugelassen und die entsprechenden Qualitätsmanagement Systeme entwickelt.
Kastellstr. 8
76227 Karlsruhe, Deutschland
arne.briest@visamed.com
+49 1704888498

[1] Die geänderten Anforderungen für die CE-Kennzeichnung und Konformitätsbewertung auf Grund der Richtlinie 2007/47/EG, M. Klümper und E. Vollebregt, MPJ 2009 Heft 2, S. 99–105.

[2] Medizinproduktgesetz, http://www.gesetze-im-internet.de/mpg/

[3] Gesundheits-Apps: Ein neuer Trend und seine rechtlichen Herausforderungen. H. Düwert und M. Klümper, MPJ 2013 Heft 1, S. 23 – 30.

[4] Bundesdatenschutzgesetz, http://www.gesetze-im-internet.de/bdsg_1990/

[5] Telemediengesetz, http://www.gesetze-im-internet.de/tmg/

[6] Heilmittelwerberecht, http://www.gesetze-im-internet.de/heilmwerbg

[7] FDA Guidance from September 25, 2013 for Industry and Food and Drug Administration Staff – Mobile Medical Applications http://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/UCM263366.pdf

[8] TestFlight http://testflightapp.com

[9] Hockey http://hockeyapp.net

[10] Android Studio von Google, basiert auf IntelliJ, http://developer.android.com/sdk/installing/studio.html

[11] Production of Large Computer Programs. In: IEEE Educational Activities Department (Hrsg.): Herbert D. Benington, IEEE Annals of the History of Computing. 5, Nr. 4, 1. Oktober 1983, S. 350–361

[12] V-Modell Kevin Forsberg and Harold Mooz, „The Relationship of System Engineering to the Project Cycle," in Proceedings of the First Annual Symposium of National Council on System Engineering, October 1991: 57–65.

[13] Release early. Release often. And listen to your customers. In: The Cathedral and the Bazaar, Raymond, Eric, 1997

[14] The Unified Modeling Language Reference Manual. James Rumbaugh, Grady Booch, Ivar Jacobson, 1999

[15] JIRA. https://www.atlassian.com/software/jira

[16] Pro Git, Scott Chancon, apress, 2009

Impressum

MPJ **Medizinprodukte Journal**
Journal für Wissenschaft und Praxis, Handel und Anwender
www.medizinprodukte-journal.de

Das MPJ wurde 1993 gegründet.
Erster und bis 2008 alleiniger Herausgeber:
Dr. Gert H. Schorn, Meckenheim/Bonn

Herausgeber:
Dr. Annika S. Bien, Bad Homburg v. d. H.
Dr. Volker Lücker, Essen
Dr. Heike Wachenhausen, Lübeck

Redaktion:
Dr. Bettina Hellwig (verantwortlich)
Am Guckenbühl 13, 78465 Konstanz
Telefon: (07531) 36 99-480
E-Mail: bhellwig@bettina-hellwig.de

MPJ-Website:
Thiemo Steinrücken
Emilienstraße 28, 12277 Berlin
Telefon: (0 30) 72 01 77 66
E-Mail: ulthiemo@t-online.de

Verlag:
Wissenschaftliche Verlagsgesellschaft Stuttgart,
Birkenwaldstraße 44, D-70191 Stuttgart
Postfach 10 10 61, D-70009 Stuttgart
Telefon: (0711) 25 82-0 / Fax: -290
www.wissenschaftliche-verlagsgesellschaft.de

Geschäftsführung:
Dr. Christian Rotta, André Caro

Anzeigen:
Leitung Media: Kornelia Wind (verantwortlich)
Birkenwaldstraße 44, D-70191 Stuttgart
Telefon: (0711) 25 82-245 / Fax: -252
Mediaberatung und -disposition: Karin Hoffmann
Telefon: (0711) 25 82-242 / Fax: -263
khoffmann@wissenschaftliche-verlagsgesellschaft.de
Mediaberatung: Dr. Axel Sobek
Telefon: (02235) 77 07-54 / Fax: -53
asobek@wissenschaftliche-verlagsgesellschaft.de
Anzeigentarif: Zurzeit gültig Nr. 21 vom 1.10.2013

Abonnenten-Service:
Wissenschaftliche Verlagsgesellschaft Stuttgart,
Postfach 10 10 61, D-70009 Stuttgart
Telefon: (0711) 25 82-353/352/357 / Fax: -290
service@wissenschaftliche-verlagsgesellschaft.de

Bezugsbedingungen:
Das „MPJ Medizinprodukte Journal" erscheint viermal jährlich. Preis im Abonnement jährlich € 178,00 zuzüglich Versandkosten (Inland € 14,80; Ausland € 19,80); Studentenabonnement € 95,00 plus Versandkosten; Einzelheft € 45,00. Preisänderungen vorbehalten. Bestellungen nehmen alle Buchhandlungen sowie der Verlag entgegen. Ein Abonnement gilt, falls nicht befristet bestellt, zur Fortsetzung bis auf Widerruf. Kündigungen des Abonnements können nur zum Ablauf des Jahres erfolgen und müssen bis zum 15. November des laufenden Jahres beim Verlag eingegangen sein.

Printed in Germany.
ISSN 0944-6885

Druck und buchbinderische Verarbeitung:
Ungeheuer+Ulmer, Körnerstraße 14–18,
D-71634 Ludwigsburg