# Verwendung von COTS-Software für medizinische Geräte

Chris Hobbs, Senior Developer, Safe Systems
QNX Software Systems Limited
chobbs@qnx.com

## Inhalt

In vielen Branchen senken die Hersteller ihre Entwicklungskosten durch die Verwendung von COTS-Software- und Hardwarekomponenten (COTS = commercial off the shelf[1]). Der ständige Druck, neue und immer umfangreichere Produkte noch schneller zur Marktreife zu bringen, lastet natürlich auch auf den Herstellern medizinischer Geräte. Diese stehen dem allgemeinen Trend aber eher skeptisch gegenüber, befürchtet man doch nicht ohne Grund, dass die Verwendung von COTS-Software zu einer undurchsichtigen Technologie-„Suppe" (SOUP = software of uncertain provenance[2]) führen könnte. Dies wiederum geht zu Lasten der Sicherheit und führt womöglich zu Ärger bei Zulassungen durch die FDA oder andere Regulierungsbehörden.

Tatsächlich muss man bei der Verwendung von COTS-Software besondere Vorsicht und Sorgfalt walten lassen. Aber weder der Standard IEC 62304 noch die Anforderungen an die funktionale Sicherheit schließen ihre Verwendung generell aus. Letztlich könnte die Integration fertiger Komponenten sogar die perfekte Lösung sein, aber nur wenn man ihre Auswahl an strengste Kriterien bindet und die gesamte Systemvalidierung ebenso konsequent durchführt. Man muss sehr genau überprüfen, ob man die „Suppe" wirklich restlos „klären" kann. „Klar" bedeutet aber, dass man Zugriff auf den Quellcode, die vollständige Fehlerhistorie und verlässliche und langfristige Nutzungsdaten hat. Dann spricht auch bei sicherheitskritischen medizinischen Anwendungen nichts gegen die Verwendung von COTS-Software.

## Die üblichen Herausforderungen

Die Hersteller medizinischer Geräte stehen vor den gleichen Herausforderungen wie alle Entwickler komplexer Systeme: Zeit, Qualität, Komplexität der Features und Kosten. Dazu kommen noch die besonderen Anforderungen an die funktionale Sicherheit und die notwendigen Marktzulassungen via FDA, MDD, MHRA, Health Canada bzw. den zuständigen Behörden in den jeweiligen Ländern, in denen die Geräte zum Einsatz kommen sollen. Diese Zulassungen sind definitiv das K.O.-Kriterium für jedes medizinische Gerät - denn ohne sie geht gar nichts!


*Abbildung 1. Entwicklung ist ein ständiger Kompromiss zwischen Funktionalität, Zeitrahmen und Qualität. Ändert sich eines, so betrifft das auch alle anderen. Verkürzt man den Zeitrahmen, so muss man bei der Funktionalität oder bei der Qualität Abstriche machen, womöglich bei beiden. Die Gesamtkosten für Entwicklung und Auslieferung sind ein Resultat dieser Kompromisse.*

[1] Als „commercial off-the-shelf", oder auch „components-off-the-shelf" (englisch für *Kommerzielle Produkte aus dem Regal*) oder kurz COTS werden seriengefertigte Produkte aus dem Elektronik- oder Softwaresektor bezeichnet, die in großer Stückzahl völlig gleichartig (ugs. „*von der Stange*") aufgebaut und verkauft werden.

[2] „Software unklarer oder undurchsichtiger Herkunft". Dieser Begriff wird im englischsprachigen Veröffentlichungen häufig verwendet, z.B. im Standard IEC 62304

Aber letztlich ändern die notwendigen Abnahmen nichts an den grundsätzlichen Herausforderungen des Softwaredesigns, auch die Wahl der richtigen Vorgehensweise bleibt davon unberührt. Wir haben im Wesentlichen zwei Möglichkeiten: a) Wir entwickeln alles oberhalb des Betriebssystems oder des BSP[3] selbst, oder b) wir integrieren fertige Komponenten in unser System, sofern es möglich und vorteilhaft erscheint.

Eine komplette Eigenentwicklung macht nur bei einfachen Systemen Sinn, deren Funktionsumfang so gering ist, dass die Verwendung eines vollständigen Betriebssystems unnötig ist. Sobald es aber komplexer wird, steigen die Entwicklungskosten und vor allem die Risiken rapide an. Dann macht es sicherlich mehr Sinn, ausgewählte Komponenten zu integrieren. Das Problem ist natürlich herauszufinden, welche der verfügbaren Software-Lösungen in unser System integriert werden können, ohne Kompromisse bei der funktionalen Sicherheit oder gar bei den Voraussetzungen für die Zulassung einzugehen. Zudem muss man natürlich auch nachweisen können, dass das Gesamtsystem alle Anforderungen erfüllt!

## Deterministische und nichtdeterministische Systeme

Theoretisch ist jedes Softwaresystem streng deterministisch, d.h. jeder Zustand und jede Zustandsänderung im System kann vorhergesagt, dokumentiert und getestet werden. In der Praxis sind viele Systeme aber so komplex geworden, dass wir sie besser als nichtdeterministisch betrachten sollten[4]. Wir kennen eben nicht alle möglichen Zustände und Zustandsänderungen.

Eine absolut kritische Folgerung daraus ist, dass man sich bei der Validierung eines Softwaresystems nicht mehr allein auf Tests verlassen kann! Das ist leicht nachvollziehbar: Bei einem beliebigen System, dessen Zustände nicht vollständig bekannt und dokumentiert sind, kann jedweder Test nur das Vorhandensein von Fehlern aufzeigen. Man kann aber unmöglich beweisen, dass keine Fehler vorhanden sind! Medizinische Software ist natürlich keine Ausnahme. Tatsächlich weist die AAMI in *Medical device software – Part 1: Guidance on the application of ISO 14971 to medical device software* extra darauf hin, dass man nicht in die Falle tappen sollte, „sich bei der Risikokontrolle auf Tests zu verlassen, da 100%iges Testen unmöglich ist"[5].

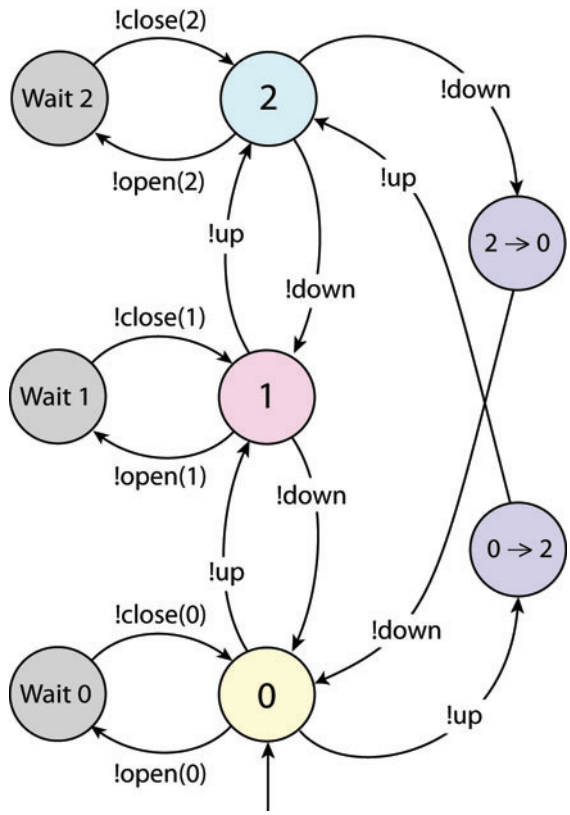


*Abbildung 2. Ein einfacher Aufzug mit einem Fehler: Es ist nicht garantiert, dass sich die Aufzugstür öffnet, ehe der Aufzug zu einem anderen Stockwerk gerufen wird*[6].

Schon anhand einer einfachen Aufzugsteuerung kann man die Grenzen des Testens aufzeigen: Abbildung 2 stellt das Zustandsdiagramm einer solchen Steuerung dar. Diese sendet Signale an den Aufzug und an die Türen am Aufzugsschacht. Die Steuerung ist sehr einfach, aber sie enthält einen nicht offensichtlichen Fehler. Hier haben wir uns, nur mal als Beispiel, darauf konzentriert, dass

[3] Board Support Package: Board-spezifische Software für ein bestimmtes Betriebssystem. Umfasst typischerweise Support um das OS laden zu können (inkl. Bootloader) sowie Betriebssystem-Gerätetreiber für die entsprechende Hardware.

[4] Die gesunkenen Kosten für leistungsfähige Multicore-Prozessoren haben immer mehr Features erst möglich gemacht – und damit auch zu einem rasanten Anstieg der Komplexität der Software geführt.

[5] AAMI, Medical device software – Part 1: Guidance on the application of ISO 14971 to medical device software, 2009. S. 55.

[6] In Anlehnung an Gerard J. Holzmann, *The SPIN Model Checker*. Boston: Addison-Wesley, 2004.

niemand in den Aufzugsschacht stürzen kann, weil sich eine Tür öffnet, ohne dass der Aufzug wirklich da ist. Leider haben wir aber nicht sichergestellt, dass niemand im Aufzug eingeschlossen wird.

Bei der Steuerung in Abbildung 2 können wir den Aufzug im Erdgeschoss besteigen, dann finden wir uns plötzlich in einer endlosen Fahrt von Stockwerk zu Stockwerk wieder. Die Türen öffnen sich nicht! Wir müssen jemanden außerhalb des Systems dazu bringen, einen „`!open`" Befehl abzusetzen, wenn der Aufzug ein Stockwerk erreicht – noch ehe die Steuerung den nächsten „`!up`" oder „`!down`" Befehl ausführt, oder aber die Steuerlogik dazu bringen, einen „`!open`" Befehl zwingend nach einer festgelegten Anzahl von Auf- oder Abwärtsbewegungen auszuführen.

Dieses einfache Beispiel zeigt eine der kritischen Herausforderungen der Verifizierung, selbst bei sehr einfachen Systemen: Wir haben schlicht vergessen zu fragen, ob alle möglichen Fahrten im Aufzug auch ein Ende haben. Und schon haben wir einen kritischen Fehler begangen. Es ist leicht nachvollziehbar, dass die Möglichkeit für solche Fehler mit steigender Komplexität eines Systems zunimmt.

Zusammengefasst könnte man sagen: Wenn es jemals möglich war, durch Testen nachzuweisen, dass ein System fehlerfrei ist, dann nur weil das System so einfach war, dass es nicht nur theoretisch sondern auch praktisch deterministisch war[7]. Man konnte dann alle Zustände und Zustandsänderungen kennen und man wusste somit auch alle kritischen Fragen, die man sich stellen musste. Unterstützen wir die Tests mit formalen Methoden zur Verifizierung von Modellen, so können wir die Korrektheit des Designs nachweisen.

Wie wir gesehen haben aber auch nur dann, wenn wir die richtigen Fragen stellen. Selbstverständlich können wir auf Tests nicht verzichten, aber wir müssen ihre Ergebnisse statistisch verstehen und sie mit anderen Maßnahmen ergänzen: Strikte Prozesskontrolle, Designvalidierung und statistische Auswertung der Fehlerhistorie.

[7] The Engineering Safety Management Yellow Book 3, *Application Note 2: Software and EN 50128*, herausgegeben von Railway Safety im Auftrag der englischen Bahnindustrie schlägt sogar vor, ein „Gerät, das so einfach aufgebaut ist, dass man alle internen Zustände durch Tests überprüfen kann, als Hardware anzusehen", siehe S. 3.

## Funktionale Sicherheit

Fehlfunktionen in medizinischen Geräten schaffen es – anders als Flugzeugabstürze oder Zugunfälle – normalerweise nicht auf die Nachrichtenseiten. Für einen betroffenen Patienten können die Folgen eines Fehlers aber dramatisch oder sogar tödlich sein. Gerätehersteller stehen moralisch, rechtlich und letztlich auch finanziell in der Pflicht, alles Menschenmögliche zu unternehmen, dass ihre Geräte keinen Schaden verursachen.

Trotz der enormen Anstrengungen, die man in die Sicherstellung der Funktionalen Sicherheit investiert hat, kommt es immer wieder und immer noch zu Fehlern. Beispielsweise hat die FDA zwischen 1990 und 2000 rund 200.000 Rückrufaktionen bei Herzschrittmachern auf Grund von Softwarefehlern verzeichnet. Zwischen 1985 und 2000 kam es zu 30.000 Todesfällen und 600.000 Verletzungen, ausgelöst durch medizintechnische Geräte. 8% davon konnten eindeutig Softwareproblemen zugeordnet werden[8].

Funktionale Sicherheit bezeichnet die Fähigkeit eines sicherheitstechnischen Systems zu tun, was es tun muss, um die Sicherheit des Gerätes zu gewährleisten: Das System erfüllt kontinuierlich seine eigentliche Aufgabe und gleichzeitig wird dafür gesorgt, dass niemals eine Person, eine Einrichtung oder die Umwelt insgesamt einem Risiko oder einer Schädigung ausgesetzt wird[9]. Kurz: Es tut das, wofür es entwickelt wurde, ohne irgendwelche Schäden zu verursachen. Ein Bestrahlungsgerät beispielsweise ist dann funktional sicher, wenn es weder dem Bedienpersonal noch anderen Personen in der Umgebung oder den gesunden Zellen des Patienten einen Schaden zufügt. Dass es den betroffenen Krebszellen schadet, ist keine Einschränkung der funktionalen Sicherheit, dafür wird es ja schließlich gebaut.

Es existieren mehrere Standards, die funktionale Sicherheit im jeweiligen Kontext beschreiben und die Prozesse und Arbeitsschritte festlegen, die man während des gesamten Lebenszyklus eines Produktes genauestens einhalten muss. Zu den bekannteren Standards gehören IEC 61508 (elektrische, elektronische und programmierbare

[8] Daniel Jackson *et al.*, eds. *Software for Dependable Systems: Sufficient Evidence?* Washington: National Academies Press, 2007. S. 23.

[9] Chris Hobbs *et al.*, „Building Functional Safety into Complex Software Systems, Part I". QNX Software Systems, 2011.

elektronische (E/E/PE) Systeme), ISO 26262 (Automobil) und die Serie der CENLEC 5012*x* Standards (Eisenbahn). Diese spezifizieren funktionale Sicherheit und Sicherheits-Integritätslevel (SIL) für Systeme in ihren jeweiligen Bereichen. Zudem legen sie die Prozesse und Maßnahmen für den Nachweis der funktionalen Sicherheit fest.

**Was genau ist funktionale Sicherheit? Ein Beispiel**

Stellen Sie sich ein Gerät vor, das gefährliche Strahlung aussendet, beispielweise Röntgenstrahlung. Würde eine Krankenschwester den Schutzschild wegschwenken können, solange die Strahlung aktiviert ist, würde sie unakzeptablen Risiken ausgesetzt.

Dieses Problem könnten wir auf verschiedene Art und Weise lösen: Wir könnten das Gerät so bauen, dass es unmöglich wäre, den Schutzschild zu bewegen, solange der Schalter an ist. Der Schalter könnte den Schwenkmechanismus physisch blockieren. Diese Lösung würde funktionieren, aber sie hätte nichts mit funktionaler Sicherheit zu tun, da sie rein intrinsisch und passiv ist. Man verlässt sich nicht auf das sachgemäße Funktionieren eines kontrollierenden Subsystems.

Alternativ könnten wir eine aktive Komponente einbauen, die die Bewegung des Schildes erkennt und die Strahlung deaktiviert, ehe es zu einer Gefährdung kommt. Die Sicherheit des Gesamtsystems hängt nun vom Funktionieren dieser Komponente ab. Dieses Subsystem stellt die eigentliche funktionale Sicherheit dar. Um sichere Systeme zu erhalten, kann und sollte man passive und aktive Maßnahmen kombinieren, was man bei realen Geräten meistens auch tut.

Bezogen auf unser Problem der Verwendung von COTS-Software müssen wir annehmen, dass der sichere Betrieb eines medizinischen Gerätes irgendeine aktive Sicherung benötigt, also funktionale Sicherheit.

## IEC 62304

IEC 62304 entwickelt sich weltweit zum *de facto* Standard für den Software-Lebenszyklus-Prozess im medizinischen Bereich. Ursprünglich von der FDA entwickelt, wurde er inzwischen mit der EU-Richtlinie 93/42 EWG harmonisiert[10]. Anders als beispielsweise IEC 61508 oder EN 50128 legt IEC 62304 keine der üblichen Limits für Fehlerraten fest, auch das Festlegen eines Sicherheits-Integritätslevels wird im Gegensatz zu Standards wie IEC 61508 nicht verlangt (welcher ohne Integritätslevel wie z.B. IEC 61508 SIL3 bedeutungslos ist). IEC 62304 beschränkt sich auf ein „Rahmenkonzept für Lebenszyklus-Prozesse mit den notwendigen Maßnahmen und Aufgaben für die sichere Entwicklung und Wartung von Software für Medizingeräte"[11]. Er macht zum Qualitäts- und Risikomanagement zwei Annahmen: a) Die Software für das medizinische Gerät wird „unter einem Qualitätsmanagementsystem entwickelt und gewartet" (z.B. ISO 13485 oder ISO 90003). Zudem, b) dass das Risikomanagement nach ISO 14971 erfolgt und dass einige kleinere zusätzliche Anforderungen nach IEC 62304, Abschnitt 7 erfüllt werden.

Bezeichnenderweise schreibt IEC 62304 nicht vor, wie man die Anforderungen zu erfüllen hat. Er legt kein Softwareentwicklungsmodell fest und definiert keine Dokumente, die man für den Nachweis der Konformität zur Verfügung stellen muss. IEC 62304 stützt sich auf ISO 14971, dort werden Anforderungen für das Risikomanagement für Medizinprodukte definiert. Keiner dieser Standards definiert Risikostufen, keiner definiert bzw. empfiehlt Verfahren zur Abschätzung von Softwarefehlerwahrscheinlichkeiten nach traditionellen statistischen Methoden. Wir können uns das für uns geeignetste Entwicklungsmodell ebenso frei aussuchen, wie die Verfahren, mit denen wir letztlich unsere Aussagen zur funktionalen Sicherheit unseres Systems nachweisen wollen.

Dafür legt IEC 62304 die Prozesse (inklusive Risikomanagement), Aktivitäten und Aufgaben für den gesamten Lebenszyklus der Software fest. Dieser endet nicht etwa mit dem Release sondern geht über Wartung und Fehlerbehebung so lange weiter, so lange die Software eben verwendet wird. Zudem werden Sicherheitsklassen abhängig vom Gefährdungspotential festgelegt: A (keine Gesundheitsgefährdung), B (Möglichkeit

[10] Cristoph Gerber, „Introduction into software lifecycle for medical devices", Stryker Navigation: Präsentation (4. Sept. 2008)

[11] International Electrotechnical Commission, *IEC 62304: Medical Device Software - Software Lifecycle Processes*. First edition, 2005-2006. Genf, International Electrotechnical Commission, 2006. Introduction.

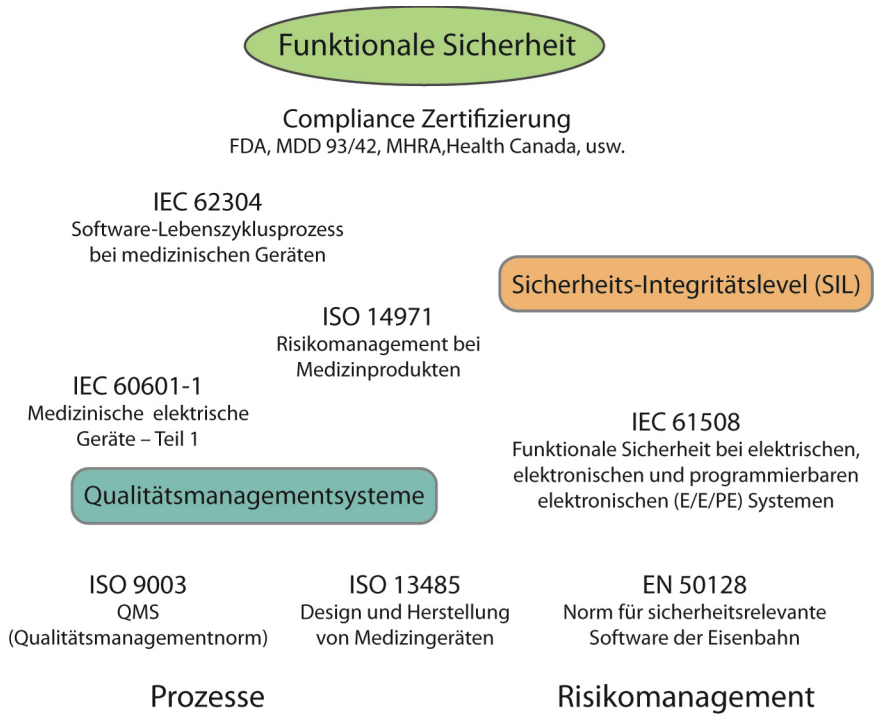


*Abbildung 3.Einige der Standards, die zur funktionalen Sicherheit für medizinische Geräte beitragen.*

geringfügiger Schädigung), C (Möglichkeit schwerwiegender oder tödlicher Schädigung).

Schließlich, und das hat nun besondere Bedeutung für die hier diskutierte Fragestellung, wird in IEC 62304 ausdrücklich auf die Verwendung von SOUP eingegangen. Sie wird dort folgendermaßen definiert:

> Software, die fertiggestellt und allgemein verfügbar ist und nicht speziell für das Gerät entwickelt wurde, in dem sie verwendet wird (allgemein als „Off-the-shelf"-Software bekannt). Ebenso Software, die bereits früher entwickelt wurde und für die keine ausreichende Dokumentation der Entwicklungsprozesse verfügbar ist.[12]

Es ist wichtig anzumerken, dass IEC 62304 annimmt, dass a) „Off the shelf"-Software verwendet wird (kommerziell vertrieben oder nicht) und b) sogar zwei Definitionen für „Software unklarer Herkunft" (SOUP) gegeben werden: Software, die nicht speziell für das fragliche Medizingerät geschrieben wurde bzw. Software mit fehlender oder unzureichender Dokumentation des Entwicklungsprozesses. IEC 62304 verbietet die Verwendung von SOUP nicht. Im Gegenteil, einige Abschnitte gehen speziell auf die Verwendung solcher Software ein: In Abschnitt 5.1.1 „Software development plans" findet man beispielweise „der Plan muss ... Software-konfigurationen und Change-Management inklusive SOUP-Konfigurationen berücksichtigen" und SOUP ist ausdrücklich das Thema in 5.3.4 „Specify SYSTEM hardware and software required by SOUP item"[13].

Die Frage ist also nicht, ob man COTS- oder SOUP-Software in Medizingeräten verwenden kann, sondern a) wie man entscheiden kann, ob eine spezielle COTS-Software für ein spezifisches Gerät geeignet ist und b) wie man nachweisen kann, dass diese Software die funktionale Sicherheit des Gerätes nicht beeinträchtigt. Um das beantworten zu können, müssen wir zunächst etwas genauer definieren, was wir unter SOUP verstehen und wie sich SOUP und COTS unterscheiden.

## SOUP und „clear SOUP"

Manche Softwareanbieter machen es sich bei der Unterscheidung zwischen COTS und SOUP etwas zu leicht: COTS ist demnach Software, hinter der ein Hersteller steht, der seinen guten Ruf und nicht zuletzt auch seinen Geschäftserfolg riskieren würde, wenn die gelieferte Software nicht wie beschrieben funktionieren würde. Hinter SOUP hingegen steht niemand.

Diese Unterscheidung ist durchaus sinnvoll, man kauft ja beispielsweise Medikamente auch lieber bei einer Apotheke als auf irgendeiner dubiosen Webseite. Letztlich ist die Unterscheidung aber irrelevant, da COTS so gut wie immer auch SOUP bedeutet. Die Prozesse, die bei der Entwicklung eingehalten wurden - oder eben auch nicht, der Quellcode, die Fehlerhistorie, also praktisch alles, was wir bei einer selbst entwickelten Software selbstverständlich hätten, alles dies steht uns nicht zu Verfügung.

[12] IEC 62304 3.29 SOUP.

[13] IEC 62304 5.1.1 Software development plans.

Deshalb sollten wir lieber „undurchsichtige Suppe“ (SOUP) und „klare Suppe“ (clear SOUP) unterscheiden. Diese Unterscheidung basiert nicht auf kommerziellen Kriterien (kommerziell vertrieben oder nicht) sondern allein auf der Verfügbarkeit von Ergebnissen und Dokumenten, die wir für den Nachweise unserer Aussagen zur funktionalen Sicherheit und dem Nachweis des Sicherheitslevels unseres System benötigen.

**Qualität und Zulassung**

Hohe Produktqualität und die Markzulassung durch die Regulierungsbehörden lassen sich nicht trennen, aber sie sind auch nicht identisch. Man kann sich leicht ein Gerät oder ein Werkzeug, z.B. eine Spritze vorstellen, die ihre primäre Funktion (Das Spritzen von Medikamenten) hervorragend erfüllt, die aber niemals eine Zulassung bekommen würde, weil die Herstellung nicht steril erfolgt.

Umgekehrt kann man sich eine Spritze vorstellen, die alle Abnahmebedingungen erfüllt (scharf, steril, sicher in der Handhabung) aber häufig beim Sterilisationsprozess während der Fertigung zerbricht. Obwohl sie für Arzt und Patient keine Gefährdung darstellt, erfüllt sie keineswegs die Ansprüche an eine vernünftige Produktqualität.

Für ein COTS-System wie z.B. Microsoft Windows mag es sehr wohl einen genau dokumentierten Entwicklungsprozess geben, der Hersteller arbeitet vermutlich auch ganz genau nach diesem exakt definierten und vollständig dokumentierten Prozess. Natürlich besitzt er auch den Quellcode, den er jederzeit bis ins Detail untersuchen könnte, die Fehlerhistorie ist wohl ebenfalls komplett erfasst und dokumentiert worden. Nachdem diese Informationen aber weder für uns, noch für andere öffentliche Untersuchungen freigegeben sind, können wir beim Windows-Betriebssystem nicht von „clear SOUP“ sprechen.[14]

Im Gegensatz zum obigen Beispiel machen Open-Source-Projekte wie Apache oder Linux sowohl den Quellcode als auch die Fehlerhistorie jedem frei zugänglich, der sich dafür interessiert und diese genauer untersuchen möchte. Aufgrund der jahrelangen aktiven Verwendung kennt man die Charakteristiken der Software sehr gut. Auch wenn die Software eigentlich „von unbekannter Herkunft“, also „SOUP“ ist, kann sie genau wie selbst entwickelte Software geprüft werden: Man kann sie einer Symbolischen Ausführung oder Code Coverage Analysen unterziehen und aufgrund der langen und umfangreichen Betriebshistorie sind Auffälligkeiten bei statistischen Untersuchungen besonders wichtig.

Somit könnte man meinen, Open-Source-Software wäre „clear SOUP“ – wir können sie untersuchen, verifizieren und validieren, als hätten wir sie selbst geschrieben.

Das klingt zunächst sehr attraktiv, aber leider ist Open-Source-Software auch nicht die optimale Lösung für medizinische Geräte. Das Problem ist, dass die Prozesse, unter denen die Software entwickelt wurde, nirgendwo sauber definiert und dokumentiert sind – aber genau das fordert der Standard IEC 62304. Wir können nicht nachvollziehen wie die Software entworfen, programmiert und verifiziert wurde. Aber ohne dieses Wissen ist der Nachweis unserer Aussagen zur funktionalen Sicherheit nahezu unmöglich. Dazu kommt noch, dass SOUP- oder COTS-Software häufig mehr Funktionalität in das System einbringt, als man eigentlich bräuchte. Das führt zu nie verwendeten Codeteilen, einem Zustand, von dem Standards zur funktionalen Sicherheit, wie z.B. IEC 61508, explizit abraten.

Klar ist aber: Wenn ein Anbieter von COTS-Software den Quellcode des Produktes und seine Fehlerhistorie veröffentlicht, dann nähern wir uns „clear SOUP“. Manche Anbieter gehen einen Schritt weiter und liefern nicht nur „clear SOUP“, sondern auch noch das Rezept dafür, indem sie ihren Kunden den genauen Entwicklungsprozess inklusive der gesamten Historie offenlegen – eine wichtige Belegesammlung, die wir verwenden können, um unsere Aussagen zur Zuverlässigkeit und Verfügbarkeit der Software zu untermauern. Andere gehen sogar noch weiter und geben alle Nachweise zur Untersuchung frei, die sie für ihre Produktzertifizierungen (z.B. IEC 61508 SIL3) bereitstellen mussten.

[14] Die Analogie mit einem Medikament aus der Apotheke oder von irgendeiner dubiosen Website passt hier nicht mehr. Medikamente aus der Apotheke sind nicht so undurchsichtig wie „SOUP“. Jeder Inhaltsstoff, jeder Prozess zur Herstellung sämtlicher Inhaltsstoffe müssen für allgemeine Untersuchungen zur Verfügung stehen. Deshalb sind die Patente für pharmazeutische und biochemische Firmen so ungeheuer wichtig: Sie dürfen keine Betriebsgeheimnisse für sich behalten und die Patente sind somit ihr einziger Schutz.

**OS-Architektur**

Das Betriebssystem, auf dem die COTS-Software läuft, muss die Aussagen des Anbieters bezüglich der funktionalen Sicherheit unterstützen.

Deshalb müssen wir das Betriebssystem und speziell die Betriebssystemarchitektur genau untersuchen, da diese absolut kritisch für die Betriebssicherheit des Systems ist. Wichtige Eigenschaften, die erfüllt sein sollten:

Präemptierbare (unterbrechbare) Kerneloperationen: Um Echtzeitanforderungen erfüllen zu können, müssen Operation des Kernels unterbrechbar sein, die Zeitfenster mit nicht präemptierbaren Operationen dürfen nicht zu lang sein.

Speicherschutz: Die Betriebssystemarchitektur muss Anwendungen und kritische Prozesse in getrennten Speicherbereichen kapseln, damit sich Fehler nicht ungehindert im System ausbreiten können.

Prioritätsvererbung: Zum Schutz gegen die sogenannte Prioritätsumkehr muss das Betriebssystem gewährleisten, dass ein niederpriorer Prozess, auf den ein hochpriorer Prozess wartet, selbst kurzzeitig dessen hohe Priorität erbt. Und zwar exakt so lange, bis er die Aufgabe erledigt oder die Ressource freigegeben hat, auf die der hochpriore Prozess gewartet hat.

Zeitpartitionierung: Um die Verfügbarkeit garantieren zu können, sollte das Betriebssystem entweder fixe, oder noch besser, adaptive CPU-Zeitpartitionierung anbieten. Damit wird es möglich, kritischen Software-Komponenten CPU-Zeitbudgets zuzuweisen.

Ein adaptiv arbeitender Scheduler kann außerdem CPU-Zeit von Partitionen abziehen, die diese aktuell nicht ausnutzen und sie Partitionen zukommen lassen, die von zusätzlicher Rechenleistung profitieren würden.

Hohe Verfügbarkeit: Ein automatisch startender Software-Watchdog sollte Prozesse überwachen, notfalls stoppen und – sofern die Sicherheit davon nicht betroffen ist – sie neu starten, ohne dass ein kompletter Reset des Systems erforderlich ist.

Neben der Wichtigkeit für die erstmalige Zulassung eines Gerätes kann ein Rezept für „clear SOUP“ (dokumentierte Entwicklungs- und Validierungshistorie, Prozessdokumentation) auch für weitere notwendige Validierungen bei Produktupgrades von unschätzbarem Wert sein[15]. Es ist interessant zu lesen, dass

> laut einer FDA-Studie 242 von 3140 (7,7%) zwischen 1992 und 1998 erfassten Rückrufaktionen auf Softwareprobleme zurückzuführen waren. Von diesen kamen 192, nahezu 80 Prozent, erst während der Wartungsphase in das System.

Anders ausgedrückt: Die Fehler wurden erst nach Markteinführung verursacht, die Geräte haben bereits problemlos funktioniert und erst später kam es zu Fehlern, oder bislang unentdeckte Probleme traten plötzlich auf. Wenn wir also Software für sicherheitskritische medizinische Geräte (IEC 62304 Klassen B und C) entwickeln, warten und aktualisieren, dann sollten wir mit „clear SOUP“ und einem ebenso klaren Rezept arbeiten, das über einen längeren Zeitraum erfolgreich verwendet und vollständig dokumentiert wurde.

## Einkauf von COTS-Software

Letztlich geht es bei der Verwendung von COTS-Software für ein medizinisches Gerät um folgende Frage: „Welchen Nachweis liefert der Anbieter, dass sein Produkt genau das ist, was wir brauchen?“ Neben all den Fragen zum Funktionsumfang, Kosten, Support usw. interessiert uns vor allem, ob uns die COTS-Software dabei unterstützt, die notwendigen Zulassungen zu bekommen. Wenn wir davon ausgehen, dass COTS-Software, nachdem wir sie ja nicht selbst entwickelt haben, irgendeine Form von SOUP ist, müssen wir feststellen, welche Form von SOUP es ist. Sollte es sich um Software handeln, für die wir keine ausreichende Dokumentation des Entwicklungsprozesses zur Verfügung haben, so wird es sehr, sehr schwierig, die Verwendung in unserem System zu rechtfertigen. Zum einen verursacht der Nachweis der funktionalen Sicherheit zusätzliche Arbeit und Kosten, zum anderen wird die Software nie den im Standard IEC 62304 geforderten Nachweis der Einhaltung definierter Software-Lebenszyklus-prozesse erfüllen können.

[15] Anil Kumar, „Easing the IEC 62304 Compliance Journey for Developers to Certify Medical Devices“ *Medical Electronic Device Solutions*, 20. Juni 2011.

Wenn die Software hingegen „clear SOUP" ist, wird unsere Arbeit bedeutend einfacher: Das ist dann der Fall, wenn die Software zwar womöglich nicht direkt für die Verwendung in einem Medizingerät entwickelt wurde, aber eine lückenlose Dokumentation der Entwicklungsprozesse vorliegt. Dann haben wir einen heißen Kandidaten für unser Gerät gefunden.

## COTS-Checkliste

Im Folgenden stellen wir eine Checkliste vor, die bei der Überprüfung helfen soll, ob eine COTS-Komponente ein geeigneter Kandidat für unser Gerät ist, speziell ob es sich um „clear SOUP" handelt. Unsere Entscheidung wird hauptsächlich von der Unterstützung der funktionalen Sicherheit des Gerätes und natürlich von Konformitäts-anforderungen unseres Systems abhängen.

### Ansprüche an die funktionale Sicherheit

Wir starten mit den Aussagen des Softwareanbieters zur funktionalen Sicherheit seiner Software.

| | |
|---|---|
| Macht der Anbieter überhaupt irgendwelche Aussagen zur funktionalen Sicherheit? | |
| Decken sich die Aussagen mit unseren Ansprüchen an die funktionale Sicherheit? | |
| Sind Kontext und Grenzen der Aussagen definiert? Treffen die Aussagen beispielsweise für Dauerbetrieb oder nur für kurzfristigen Betrieb zu? | |
| Ist die Wahrscheinlichkeit des Auftretens kritischer Fehler in den Aussagen zur funktionalen Sicherheit spezifiziert?<br>Oder umgekehrt: Gibt es Aussagen zur Betriebssicherheit der Software? | |
| Definiert der Anbieter eine „ausreichende Betriebssicherheit"? Wie werden die Aussagen zur Betriebssicherheit quantifiziert? Besteht die Quantifizierung nur aus den (im Grunde bedeutungslosen) 99,999%, oder bekommt man sinnvolle Angaben zur Verfügbarkeit und Zuverlässigkeit im relevanten Umfeld? | |
| Gibt es quantifizierte Aussagen des Anbieters zur<br>• Verfügbarkeit: Zu wie viel Prozent reagiert die Software auf Ereignisse im geforderten Zeitrahmen?<br>• Zuverlässigkeit: Wie viele der Reaktionen sind korrekt? | |

### Prozesse

Ein genau definierter und dokumentierter Prozess für den gesamten Lebenszyklus der Software ist eine absolut notwendige Anforderung – ohne ihn brauchen wir bei der Auswahl gar nicht weiter-machen!

| | |
|---|---|
| Arbeitet der Softwarehersteller nach einem Qualitätsmanagementsystem? | |
| Erfüllt das Qualitätsmanagement die Anforderungen eines der folgenden Standards?<br>• Ein Standard aus der ISO 9000 Familie?<br>• ISO 15504 (Software Process Improvement Capability Determination (SPICE)?<br>• Capability Maturity Model Integration (CMMI)? | |
| Welche Prozesse verwendet der Hersteller für die Quellcodeverwaltung (Versionsverwaltung)? | |
| Wie werden aufgetretene Fehler dokumentiert, nachverfolgt und gelöst? Und zwar sowohl die während der Verifikation gefundenen, als auch die im realen Einsatz aufgetretenen Probleme. | |
| Werden Fehler klassifiziert und anschließend einer umfassenden Analyse unterzogen? | |

### Fehlerbaumanalyse

Die Fehlerbaumanalyse, z.B. mit Hilfe „Bayesscher Netze", ist ein wichtiges Verfahren zur Eliminierung von Designfehlern und zur Abschätzung der Wahrscheinlichkeit eines Ausfalls. Zudem ergeben sich wichtige Resultate, die von den Prüfern für die Zulassungen verwendet werden können[16].

| | |
|---|---|
| Wurde die Software einer Fehlerbaumanalyse unterzogen? | |
| Wurden sowohl „a priori" (von der Ursache zur Wirkung) als auch „a posteriori" (von der Wirkung zur Ursache) Nachweise geführt? | |
| Bekommen wir Einsicht in die Ergebnisse der Fehlerbaumanalyse? | |

[16] Chris Hobbs, „Fault Tree Analysis with Bayesian Belief Networks for Safety-Critical Software", 2010.

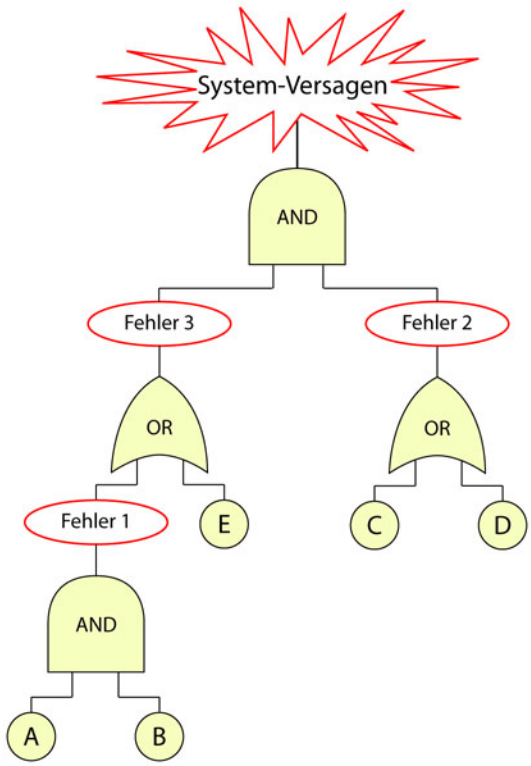


*Abbildung 3. Eine einfache Fehlerbaumanalyse. Die Fehler sind nummeriert (1,2, etc.), Buchstaben bezeichnen die Blätter des Baumes (A, B, usw.).*

## Statische Analyse

Statische Analysen sind unverzichtbar um verdächtige Codesegmente zu identifizieren und ihre Verwendung wird von Zulassungsbehörden wie der FDA explizit empfohlen[17], insbesondere „die Verwendung verschiedener statischer Analyse-Techniken wie z.B. Symbolische Ausführung, abstrakte Interpretation und Reverse-Engineering und deren Anwendung für die Analyse von Software für medizinische Geräte....“ Ziel ist die Steigerung der Fähigkeit des zur FDA gehörigen CDRH[18] Zentrums, „die Softwarequalität sowohl vor Markteinführung als auch danach beurteilen zu können” sowie die „Verbesserung des aktuellen Levels der statischen Analyse durch die Erhöhung der Präzision und Effizienz der Werkzeuge zur statischen Analyse, insbesondere für die Anwendung bei Medizingeräten“[19].

| | |
|---|---|
| Verwendet der Hersteller statische Analysen, um potentielle Fehler im Produkt finden zu können? | |
| Welche Werkzeuge zur statischen Analyse kommen zum Einsatz?<br>• Syntaktische Analysen gegen gängige Coding-Standards?<br>• Abschätzung der Fehlerwahrscheinlichkeit?<br>• Korrektheitsnachweise (z.B. durch Assertions im Quellcode)?<br>• Symbolische Ausführung? | |
| Welche Dokumente stellt der Hersteller der COTS-Software zur Untermauerung der Ergebnisse der statischen Analyse zur Verfügung? | |

## Nachvollziehbare Nutzungshistorie

Eine nachweisbare Nutzungshistorie ist für die Überprüfung der Aussagen zur Betriebssicherheit von COTS-Software unumgänglich – und für die Bildung der Aussagen natürlich genauso. Jeder, der ein Softwaresystem herstellt, für das er irgendwann einmal (und sei es in noch ferner Zukunft) einen Nachweis zur Betriebssicherheit liefern muss, sollte das Sammeln von Nutzungsdaten fix in sein Geschäftsmodell einbauen.

| | |
|---|---|
| Kann der Anbieter der COTS-Software Nutzungsdaten vorlegen? | |
| Wie weit reichen die Daten zurück? | |
| Wie aussagekräftig sind die Daten?<br>• Wie viele Daten liegen insgesamt vor?<br>• Decken die Daten nur einen kleinen oder einen großen Teil der gesamten Laufzeit ab?<br>• Woher bekommt der Hersteller die Daten? | |
| Liefert der Anbieter zusätzlich Daten aus der Fehleranalyse oder nur reine Nutzungsdaten? | |

## Dokumente und Nachweise

Entwurfsdokumente und Validierungsergebnisse sind die wesentlichen Unterschiede zwischen „SOUP” und „clear SOUP“. Wenn der Anbieter

[17] Beispielweise: „statische Analysen stellen eine effiziente Möglichkeit dar, Fehler schon vor der Ausführung des Codes finden...“ FDA, General Principles of Software Validation; Final Guidance for Industry and FDA Staff. 11. Jan. 2002.

[18] *Center for Devices and Radiological Health* (*CDRH*)

[19] FDA, Research Project: Static Analysis of Medical Device Software, aktualisiert 11. Feb. 2011.

einer COTS-Software keine umfangreiche Ergebnissammlung vorlegen kann, gibt es keinen wirklichen Grund seinen Angeboten einen Vorzug gegenüber Open-Source-Software zu geben.

| | |
|---|---|
| Welche Entwurfsdokumente liefert der Hersteller mit seiner Software mit? | |
| Stellt der Hersteller folgende Dokumente zur Verfügung?<br>• Dokumente zum Entwurf der Softwarearchitektur<br>• Dokumente zum Detailentwurf | |
| Welche Testpläne wurden verwendet und kann man die Ergebnisse einsehen? | |
| Welche zusätzlichen Validierungsmethoden hat man angewendet, sind die Methoden und Ergebnisse einsehbar? | |
| Führt der Anbieter eine Anforderungs-Rückverfolgbarkeits-Matrix von der Anforderungsdefinition bis zur Auslieferung? Ist diese für Kontrollen verfügbar? | |
| Welche Aufzeichnungen gibt es zum Lebenszyklus der Software?<br>• Änderungen?<br>• Probleme und ihre Lösung? | |

### Safety-Manual

Das Safety-Manual ist ein weiteres K.O.-Kriterium für die Auswahl: Gehört kein Sicherheitshandbuch zum Lieferumfang der COTS-Software, geben Sie sie zurück und versuchen Sie es bei einem anderen Anbieter!

| | |
|---|---|
| Stehen explizite Aussagen zur funktionalen Sicherheit im Safety-Manual? | |
| Werden Aussagen zum Kontext und zu Einschränkungen in Bezug auf die funktionale Sicherheit gemacht? Diese sollten das Betriebsumfeld und die Art der Verwendung umfassen. Ein paar Beispiele:<br>• „Diese Liste der unterstützten Prozessorarchitekturen ist vollständig“<br>• „Es ist nicht zulässig, Gleitkommaoperation in einem Signal-Handler auszuführen“<br>• „Kritische Budgets können maximal der Zeitfenstergröße entsprechen.“ | |
| Wird ein Training zur sicheren Nutzung des Produktes angeboten? | |

## Zertifizierte Komponenten

Ist man nun allen bisher gegeben Ratschlägen gefolgt und erfüllt die COTS-Software alle Anforderungen an „clear SOUP“, so ist aber immer noch nicht sicher, dass das endgültige Produkt wie geplant und rechtzeitig auf den Markt kommt. Noch besser wäre es, mit einem COTS-Softwareanbieter zu arbeiten, der bereits Erfahrung mit Genehmigungen hat und Komponenten zu verwenden, die bereits wichtige Zulassungen besitzen.

Obwohl Organisationen wie die FDA, MHRA, Health Canada bzw. die Behörden in den jeweiligen Ländern keine Komponenten abnehmen, sondern nur komplette Systeme und Produkte, können bereits (nach z.B. IEC 61508 oder IEC 62304) zertifizierte Komponenten den ganzen Zulassungs-prozess vereinfachen und die Markteinführung deutlich verkürzen.

Damit diese Komponenten ihre Zertifizierungen erhalten konnten, wurden sie notwendigerweise unter Einhaltung geeigneter Prozesse und einem Qualitätsmanagement entwickelt. Zudem wurden sie vom Hersteller strengen Tests und Validierungen unterzogen, der Hersteller verfügt somit über hilfreiche Nachweise für die Abnahme des endgültigen Gerätes. Schlussendlich kann ein Hersteller, der bereits Erfahrung mit Zulassungen hat, seinen Kunden wertvolle Ratschläge geben und Unterstützung leisten.

## Fazit

Weder der Standard IEC 62304 „Software for medical devices” noch die Anforderungen an die funktionale Sicherheit schließen die Verwendung von COTS-Software für medizinische Geräte aus. Man muss allerdings besondere Vorsicht und Sorgfalt walten lassen, aber dann könnte COTS-Software die perfekte Lösung sein – immer vorausgesetzt, dass die Auswahl der Komponenten strengsten Kriterien genügt und die Validierung des gesamten Systems ebenso akribisch erfolgt. Wenn wir wirklich genau darauf achten, dass wir nur „clear SOUP” verwenden, also Software, für die wir Quellcode, eine Fehlerhistorie und verlässliche und langfristige Nutzungsdaten haben, so werden wir feststellen, dass COTS-Software für viele sicherheitskritische Medizingeräte die optimale Lösung darstellt.

## Literaturhinweise


AMI. Medical device software – Part 1: Guidance on the application of ISO 14971 to medical device software. Association for the Advancement of Medical Instrumentation, 2009.

Berard, B. et al. *Systems and Software Verification*. Berlin: Springer, 2001.

Birman, Kenneth P. and Thomas A. Joseph. „Exploiting Virtual Synchrony in Distributed Systems". Cornell University. Februar 1987.

Birman, Kenneth P. Building Secure and Reliable Network Applications. Greenwich: Manning, 1996.

FDA. *General Principles of Software Validation; Final Guidance for Industry and FDA Staff*. 11. Jan. 2002. <ww.fda.gov/MedicalDevices/DeviceRegulationandGuidance/ GuidanceDocuments/ucm085281.htm>

FDA, *Research Project: Static Analysis of Medical Device Software*, aktualisiert 11. Feb. 2011. <www.fda.gov/MedicalDevices/ScienceandResearch/ucm243156.htm>

Gerber, Cristoph. „Introduction into software lifecycle for medical devices". Freiburg, Stryker Navigation. Präsentation (4. Sept. 2008).

Green, Blake. „Understanding Software Development from a Regulatory Viewpoint". Journal of Medical Device Regulation, 6:1 (Feb. 2009), S. 14-23.

Helminen, Atte. *Reliability estimation of safety critical software-based systems using Bayesian networks*. Helsinki: Säteilyturvakeskus (Finnish Radiation and Nuclear Safety Authority), 2001. <http://www.stuk.fi/julkaisut/tr/stuk-yto-tr178.pdf>

Hobbs, Chris. „Fault Tree Analysis with Bayesian Belief Networks for Safety-Critical Software". QNX Software Systems, 2010. <www.qnx.com>

Hobbs, Chris, *et al.* „Building Functional Safety into Complex Software Systems, Part I". QNX Software Systems, 2011. <www.qnx.com>

Hobbs, Chris, *et al.* „Building Functional Safety into Complex Software Systems, Part II". QNX Software Systems, 2011. <www.qnx.com>

Holzmann, Gerard J., *The SPIN Model Checker*. Boston: Addison-Wesley, 2004.

International Electrotechnical Commission. *IEC 62304: Medical Device Software – Software Lifecycle Processes*. First edition, 2005-2006. Genf: International Electrotechnical Commission, 2006.

Jackson, Daniel et al., eds. *Software for Dependable Systems: Sufficient Evidence?* Washington: National Academies Press, 2007.

Jackson, Daniel et al., eds. *Sufficient Evidence: A Briefing of the National Academies Study Software for Dependable Systems.* <cstb.org/pub_dependable>

Kumar, Anil. „Easing the IEC 62304 Compliance Journey for Developers to Certify Medical Devices". Medical Electronic Device Solutions. 20. Juni 2011. <www.medsmagazine.com/articles/view/118>

Lions, J. L. et al. *Ariane 501 Inquiry Board Report*. Paris: ESA, 1996.

NASA. *Agency Risk Management Procedural Requirements* (NP4 8000.4A). NASA, 16. Dez. 2008.

*QNX Neutrino RTOS Safe Kernel 1.0: Safety Manual: QMS0054 1.0.* QNX Software Systems, 2010. <www.qnx.com>

Reason, James. *Human Error*. Cambridge: Cambridge UP, 1990.

## Über QNX Software Systems

QNX Software Systems ist Hersteller innovativer Embedded-Technologien, dazu gehören Middleware, Entwicklungswerkzeuge und Betriebssysteme. Die komponentenbasierte Architektur des QNX® Neutrino® RTOS, die QNX Momentics® Tool Suite und die QNX Aviage® Middleware-Reihe bilden gemeinsam das zuverlässigste und skalierbarste Fundament für leistungsfähige Embedded-Systeme. Weltweit bekannte Firmen wie z.B. Cisco, Daimler, General Electric, Lockheed Martin oder Siemens nutzen QNX-Technologie für Telematik- und Infotainmentsysteme, Industrieroboter, Netzwerk-Router, medizinische Geräte, Sicherheits- und Verteidigungssysteme und viele andere betriebs- und sicherheitskritische Applikationen. Die QNX-Firmenzentrale ist in Ottawa, Kanada, die deutsche Niederlassung befindet sich in Hannover.

**www.qnx.com**