Medical Device Regulation

Erste Hilfe bei altem Code

Nicht erst mit der neuen Medical Device Regulation (MDR) rückt die Qualitätssicherung bei der Software für medizinische Geräte in den Blickpunkt. MDR und andere Normen schreiben vor, dass die Hersteller die Qualitätssicherung über den gesamten Lebenszyklus eines Produkts sicherstellen müssen. Bei aktuellen Systemen ist das in der Regel kaum problematisch. Ganz anders gestaltet sich die Situation, wenn ältere Geräte um neue Funktionen erweitert werden sollen: wer im Unternehmen kennt noch den Code oder weiß, wo die – hoffentlich vorhandene – Dokumentation zu finden ist?
Hier helfen Tools wie Imagix 4D, die die Struktur eines Programms analysieren und so die Entwickler unterstützen.

von Klaus Lambertz, Geschäftsführer der Verifysoft Technology GmbH

Die digitale Transformation hat auch in der Medizintechnik massiv an Fahrt aufgenommen. Fast jedes neue Gerät verfügt über Software, drahtlose Verbindungen und die Möglichkeit, Daten aus Sensoren auszulesen. Dadurch ergeben sich in Therapie und Diagnostik wichtige neue Möglichkeiten, aber auch neue Risiken. Szenarien, in denen Angreifer medizinische Geräte attackieren, gehören nicht mehr in die Rubrik Science-Fiction, sondern werden real. Es gilt also, Gefahren für Patienten und Bedienpersonal durch bestmögliche Qualität der Geräte zu minimieren. Darauf zielen die zahlreichen Normen ab, die in der Medizintechnik relevant sind. Mit der Medizinprodukteverordnung EU 2017/745 (MDR, Medical Device Regulation) werden die Vorgaben der Qualitätssicherung noch mehr in den Mittelpunkt gerückt.

MDR, IEC 62304 oder ISO 14971 fordern unisono, aber meist ohne konkrete Hilfestellung, dass ein Hersteller Qualitätssicherungs- und Risikomanagement-Prozesse implementieren muss. Bei Embedded-Systemen und Stand-Alone-Software müssen im Rahmen der Qualitätssicherung zwei Bereiche unterschieden werden. Zum einen geht es während der Entwicklung darum, Fehler beim Code zu vermeiden – die Verifizierung - und die geforderte Funktionalität sicherzustellen. Dieses wird als Validierung bezeichnet. Auf der anderen Seite müssen die Systeme über ihren Lebenszyklus hinweg betrachtet werden. Denn Produkte, die wesentlich auf Software basieren, können im Laufe ihrer Lebensdauer erheblichen Änderungen unterworfen sein. Etwa durch Updates oder neue Funktionen, die hinzugefügt werden. Oder auch, wenn eine bei der Entwicklung genutzte Bibliothek durch eine neuere Version ersetzt wird. In den einschlägigen Normen und Vorschriften werden beide Aspekte berücksichtigt. So macht zum Beispiel die MDR deutlich: „Bei Produkten, zu deren Bestandteilen Software gehört, oder bei Produkten in Form einer Software wird die Software entsprechend dem Stand der Technik entwickelt und hergestellt, wobei die Grundsätze des Software-Lebenszyklus, des Risikomanagements einschließlich der Informationssicherheit, der Verifizierung und der Validierung zu berücksichtigen sind.“

Probleme kommen mit dem Alter

Neue Produkte sind hinsichtlich der Qualitätssicherung über den Lebenszyklus hinweg unproblematisch, wenn entsprechende Prozesse im Unternehmen verankert wurden. Vor allem bei den heute üblichen agilen Entwicklungsmethoden wie Continuous Integration/Continuous Deployment spielt die Dokumentation eine große Rolle. Auch ist das Prinzip des „Clean Code“, des sauberen und von allen überflüssigen Verwinkelungen freien Codes, ein wichtiger Bestandteil vieler Entwicklungsabteilungen geworden. Ein Element dabei ist das so genannte Refactoring, mit dem der Code verbessert werden soll. Code ist von Beginn an nicht perfekt, alle Teile müssen permanenten Reviews unterzogen werden. Die Aufgabe des Refactorings ist es, den Code in eine für die Entwickler wünschenswerte, also leicht nachvollziehbare Form zu bringen. Dabei verfolgt das Refactoring zwei Hauptziele: Die Erweiterbarkeit des Codes so einfach wie möglich zu machen und gleichzeitig die Wartbarkeit zu gewährleisten. Zudem soll erreicht werden, dass der Code ganz oder in Teilen auch in späteren Projekten wiederverwendet werden kann. Im Unterschied zum Debugging hat das Refactoring allerdings keine Auswirkungen auf das Verhalten des Programms. Der Code wird nicht funktional verändert. Streng genommen werden beim Refactoring gefundene Fehler oder Sicherheitsprobleme nicht beseitigt, sondern nur für eine Bereinigung vorgemerkt.

Geschieht das Refactoring im laufenden Entwicklungsprozess als integrative Maßnahme, ist der Aufwand überschaubar. Anders jedoch bei älteren Systemen: Je länger eine Anwendung ohne Optimierung des Codes im operativen Betrieb ist, desto schwerer wird dieser Code durchschaubar. Denn über den Lebenszyklus einer Anwendung hinweg müssen immer wieder Änderungen und Anpassungen vorgenommen werden, die das Verhalten der Software beeinflussen. Der Aufwand bei Wartung und Modernisierung steigt rapide an, da das Wissen um die Architektur und Funktionalitäten schwindet.

Der Extremfall ist Legacy-Code: Legacy-Code ist in Hinblick auf Wartung oder Erweiterung eine Herausforderung. Meist ist der Code extrem unübersichtlich. Oft fehlen die grundlegendsten Dokumentationen. Und die damals verantwortlichen Entwickler sind im Ruhestand oder in alle Winde zerstreut. Dennoch muss die Software um neue, aktuelle Funktionalitäten erweitert werden. Oft sind dabei Fehler sind zu beseitigen, die bislang unentdeckt blieben. Für die damit betrauten Entwickler beginnt dann eine detektivische Spurensuche in der alten Struktur, die durchaus auch archäologische Qualitäten fordert.

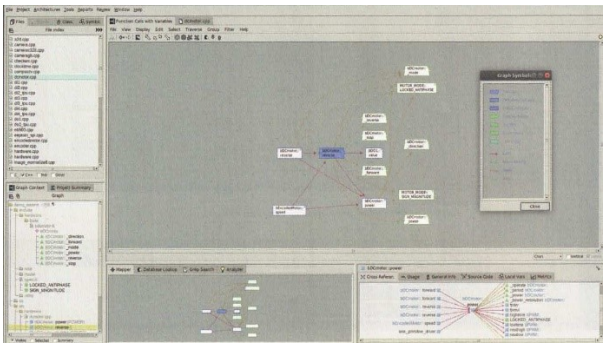

Abb 1: Funktions-Aufruf-Diagramme zeigen die Reihenfolge aufgerufener Funktionen und weitere Informationen.

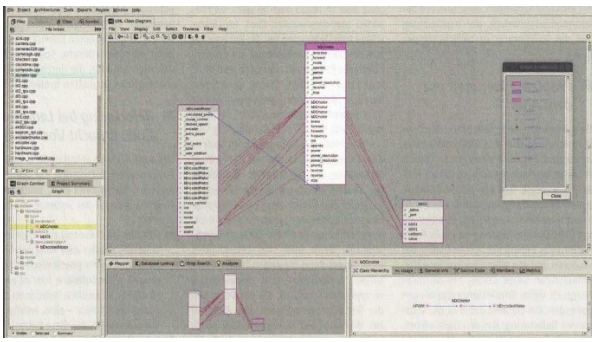

Abb. 2: Das UML-Klassen-Diagramm gibt Eigenschaften von Klassen und Beziehungen zwischen diesen in UML-Notation wieder.

Refactoring bei Legacy-Code braucht Vorbereitung

Um alten Code so aufzubereiten, dass ein planvolles Refactoring mit sinnvollem Aufwand durchgeführt werden kann, sollten zuerst folgende Aspekte der Software untersucht werden:

- Dateien: Im C-Kontext sind Dateien entweder Header oder kompilierbare Dateien, also die physikalischen Komponenten der Software. Hier ist wichtig zu wissen, in welchen Relationen diese zueinander stehen - etwa, welche gemeinsamen Header diese haben.

- Subsysteme: Welche Subsysteme gibt es und in welchen Relationen stehen diese zueinander? Welche Architektur liegt den Subsystemen zugrunde?

- Datentypen: Als Typen werden üblicherweise Pointer, Enums, Classes, Structs und dergleichen bezeichnet. Hier interessieren besonders die Beziehungen zwischen Typen und Variablen.

- Funktionen: Die Aufrufhierarchie von Funktionen innerhalb eines Projekts ist elementar wichtig, um den Code zu verstehen. Dabei sollten sowohl eingehende als auch ausgehende Aufrufe berücksichtigt werden. Auch der Kontrollfluss zwischen den Funktionen ist relevant, also an welcher Stelle in eine andere Funktion gesprungen wird. Dazu wiederum müssen Verzweigungen und Schleifen im Programm bekannt sein, da diese den Kontrollfluss steuern.

Diese Analysen sind ab einer gewissen Projektkomplexität manuell nicht zu leisten. Allein die Beziehungen zwischen den unterschiedlichen Dateien eines Projekts zu erschließen ist eine fehleranfällige Fleißarbeit. Der Einsatz geeigneter Tools ist unausweichlich, um so weit als möglich automatisierbare Analysen vorzunehmen. Ein bewährtes Werkzeug für die Untersuchung von Quelltexten in C/C++ und Java ist das von Imagix entwickelte und von Verifysoft Technology vertriebene Tool Imagix 4D.

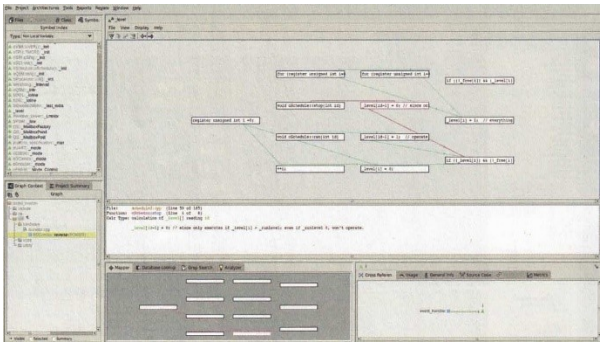

Abb 3: Der Berechnungsbaum einer Variante zeigt, Welche Werte und andere Variablen zu einer Variable beitragen, beziehungsweise welche anderen Variablen beinflusst werden.

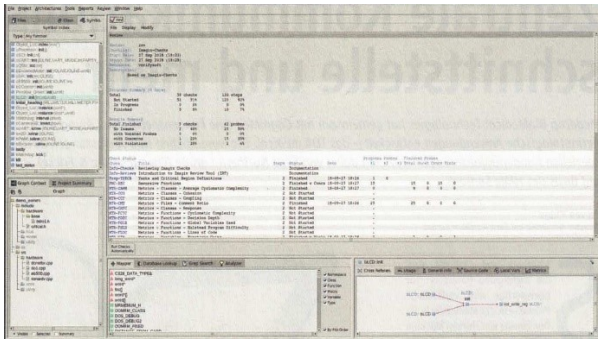

Abb.4: Mit der Review-Funktion unterstützt Imagix 4D ansonsten rein manuelle Prozesse als zentrales teilautomatisiertes Werkzeug.

Grafische Aufbereitung der Struktur

Imagix 4D analysiert den Quelltext einer Software und bereitet die für das Refactoring relevanten Informationen grafisch auf. Damit steht den Entwicklern eine Repräsentation des gesamten Projekts zur Verfügung, die alle Relationen in der jeweils benötigten Detailtiefe darstellt. Je nach Fragestellung verfügt das Werkzeug über unterschiedliche Darstellungsweisen. Zum Überblick über die Abhängigkeiten aller in einem Projekt vorhandenen Subsysteme werden die Informationen in Form einer Design-Struktur-Matrix aufbereitet. Darin lässt sich zum Beispiel die Granularität der Subsysteme vom Wurzelverzeichnis bis auf die Ebene einzelner Funktionen herunterbrechen. Zum besseren Verständnis der Subsystem-Architektur wiederum kann diese als Diagramm dargestellt werden. Bei unübersichtlichen Architekturen, etwa mit sehr vielen Dateien direkt im Stammverzeichnis, helfen Filter, den richtigen Fokus zu finden. Zahlreiche weitere Ansichten, etwa für die Darstellung der Funktionsabhängigkeiten oder der Kontrollflüsse, geben den Entwicklern weitere detaillierte Informationen. Ein weiteres wichtiges Merkmal ist die Suche nach Auffälligkeiten im Code, um gezielt die Qualität der Anwendung zu steigern. Dazu zählen unter anderem Rekursionen, Deadlocks, nicht verwendete Variablen oder unpassende Typenkonvertierungen.

Mit diesem Wissen ist es möglich, den vorhandenen Code zu verstehen und seine Funktionalität nachzuvollziehen. Im nächsten Schritt kann der Code dann im Rahmen des Refactorings bereinigt und in eine schlüssige Form gebracht werden. Zudem ist es durch den Einsatz von Imagix 4D mit vertretbarem Aufwand möglich, eine umfassende Dokumentation zu erstellen – für möglicherweise notwendige Zertifizierung unverzichtbar. Auf dieser Basis lässt sich die Anwendung dann mit neuen Funktionen versehen. Zudem ist so die Basis vorbereitet, um das betagte Projekt mit aktuellen Ansätzen der agilen Entwicklung weiter zu betreiben.

Fazit

Die Qualitätssicherung über den gesamten Lebenszyklus ist in der Medizintechnik nicht neu, gewinnt aber durch aktuelle technologische Entwicklungen und die MDR an Gewicht. Um die Qualität auch bei langlebigen oder im Feld weit verbreiteten Produkten sicherzustellen, muss zunächst der Code bekannt sein – in vielen Fällen ein reales Problem. Das Refactoring hat sich nicht ohne Grund als integraler Bestandteil der agilen Entwicklungsmethoden etabliert. Auch Altanwendungen profitieren davon – besonders, wenn sie noch nicht am Ende ihres Lebenszyklus angekommen sind. Dafür muss der vorhandene Quellcode jedoch genau analysiert werden, da das Refactoring keinesfalls das Verhalten der Software ändern soll. Trial-and-Error-Ansätze sind hier fehl am Platz.

Ohne geeignete Werkzeuge ist die Analyse komplexer Anwendungen kaum zu leisten, Fehler lassen sich nicht mit wirtschaftlich vertretbarem Aufwand ausschließen. Durch eine grafische Aufbereitung der Architekturen und den zugrunde liegenden Strukturen bekommen die Entwickler ein gutes Verständnis davon, wie eine Anwendung aufgebaut ist und wo die richtigen Einstiegspunkte in das Refactoring sind. So können auch alte Systeme auf einen Stand gebracht werden, bei dem die Vorgaben an das Qualitäts- und Risikomanagement erfüllbar sind. Davon profitieren alle: Mehr Sicherheit für die Patienten, eine längere Produktlebensdauer für die Hersteller.

Weitere Informationen

https://www.verifysoft.com/de_imagix4d.html