Diplomarbeit

Valerii Kebenko

Das risikoorientierte und automatisierte Testen einer medizinischen Software mit Hilfe von Black-Box-Regressionstests

10.12.2012

Erstprüfer: Prof. Dr. Sibylle Schupp
Zweitprüfer: Prof. Dr.-Ing. Andreas Timm-Giel
Betreuer: Dipl.-Inf.(FH) Eric Thomas, MBA

Eidesstattliche Erklärung

Ich erkläre hiermit, dass die vorliegende Diplomarbeit ohne fremde Hilfe selbstständig verfasst wurde und nur die angegebenen Quellen und Hilfsmittel benutzt worden sind.

Wörtlich oder sinngemäß aus anderen Werken entnommene Stellen sind unter Angabe der Quelle kenntlich gemacht.

Diese Diplomarbeit wurde bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt oder veröffentlicht.

Hamburg, 10.12.2012

_____

Name

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Seit Jahren sind medizinische Geräte aus dem klinischen Alltag nicht mehr wegzudenken. Schließlich gestalten sie die Diagnostik und Therapie bei Patienten effizient und erfolgreich - sofern sie fehlerfrei funktionieren.

Während früher in der Medizin meist reine Hardware-Geräte zum Einsatz kamen, werden in der heutigen Zeit größtenteils sog. PEMS (Programmierbare Elektronisch Medizinische Systeme) eingesetzt. Diese Systeme werden durch Software gesteuert. Ihre Bandbreite reicht hierbei von Gerätesoftware („Embedded"-Systemen) bis hin zu „Standalone"-Softwarelösungen.

Bei der Entwicklung oder Anpassung von Softwaresystemen besteht immer das Risiko, dass es zu Fehlern in der Software kommt. Bis zu einem gewissen Grad und in Abhängigkeit des Einsatzgebietes wird dies erwartet und toleriert. Ein Fehlverhalten in medizinischer Software kann allerdings kritische Folgen haben. Neben erhöhten Wartungskosten können Softwarefehler auch zu einem erhöhten Gefährdungspotential für Patienten und Anwender führen. Dies stellt eine ernstzunehmende Gefahr für die Gesundheit und eventuell für das Leben der Betroffenen dar.

Genau aus diesem Grund unterliegt die Entwicklung von medizinischer Software normativen Vorgaben und hohen Qualitätsansprüchen. Die Erfüllung der geforderten Standards ist folglich zwingende Voraussetzung für die Veröffentlichung eines Medizinproduktes. Deshalb werden bei der Entwicklung medizinischer Software Qualitätssicherungsmaßnahmen, wie z.B. das Testen durch normative Anforderungen vorgegeben. DIN EN 62304 beschreibt Anforderungen an den Lebenszyklus einer Medizingeräte-Software. Für Software, bei der keinerlei Gefährdungen für Patienten oder Anwender bestehen, sind Qualitätssicherungsmaßnahmen jedoch optional. Aus Sicht der Softwareentwicklung ist es aus allerdings nicht sinnvoll, eine ungetestete Software – und somit unbekanntes Fehlerpotential – in den Markt einzuführen. Aus dieser Situation entsteht ein Konflikt zwischen den wirtschaftlichen Interessen eines Unternehmens, welches Produkte möglichst schnell und unkompliziert in den Markt einführen möchte, und den qualitativen Ansprüchen an moderne Software, welche möglichst gut getestet und fehlerfrei sein sollte. Da anzunehmen ist, dass ein Unternehmen im Zweifel seine wirtschaftlichen Interessen in den Vordergrund stellt, ist es unabdingbar, Bedingungen für möglichst effiziente und effektive Qualitätssicherungsmaßnahmen im Softwarebereich zu schaffen.

Im Rahmen dieser Diplomarbeit wird ein Testkonzept erarbeitet, welches effiziente und effektive Qualitätssicherungsmaßnahmen entwicklungsbegleitend ermöglicht. Hierzu wird ein risikoorientierter und automatisierter Black-Box Regressionstest konzipiert und am Beispiel des

Unternehmens seca, dem Marktführer für medizinisches Messen und Wiegen, implementiert. Das Testkonzept wird aus der Sicht des operativen Testmanagements erstellt. Strategische Aspekte des Testmanagements werden in dieser Arbeit nicht näher betrachtet.

seca ist der Marktführer im medizinischen Messen und Wiegen und muss aufgrund dieser Marktposition effizient und effektiv arbeiten. Außerdem entwickelt es medizinische Software der Sicherheitsklasse A und ist somit ein geeigneter Kandidat für die Problemstellung dieser Arbeit.

Im folgenden Kapitel werden die allgemeinen Grundlagen des Softwaretestens dargestellt. Als erstes werden der Begriff Softwarequalität, sowie die Methoden der Qualitätssicherung und normative Anforderungen an eine medizinische Software erläutert. Anschließend wird auf das Softwaretesten näher eingegangen. Hier werden die Prüftechniken nach verschiedenen Kriterien klassifiziert und anschließend der Regressionstest, GUI-Test, sowie Testautomatisierung näher beschrieben. Abschließend stellt das Kapitel unterschiedliche Fehlermetriken vor und erläutert diese.

Kapitel 3 führt zunächst eine Bestandsanalyse bei dem Unternehmen seca durch. Diese beschreibt den Entwicklungs- und Testprozess, sowie das Unternehmen und die entwickelten medizinischen Softwareprodukte. In der folgenden Fehleranalyse werden mit Hilfe eines Fehlerstrommodells die Fehlerverteilungen bei den seca-Softwareprodukten bestimmt.

Im vierten Kapitel stellt diese Arbeit auf Basis der Bestand- und Fehleranalyse ein Testkonzept für den risikoorientierten und automatisierten Black-Box Regressionstest dar. Das Kapitel beschreibt zunächst die Kriterien für die Auswahl eines Testautomatisierungstools. Anschließend wird der Testprozess, bestehend aus der Identifizierung, Priorisierung und Vervollständigung der Testbereiche, sowie die praxisrelevante Umsetzung der Automatisierung dargestellt.

In Kapitel 5 wird das zuvor erarbeitete Konzept am Beispiel einer medizinischen Software der Firma seca implementiert. Am Ende des Kapitels werden die Ergebnisse der Konzeptumsetzung präsentiert.

Eine Zusammenfassung und ein Ausblick auf mögliche Verbesserungen runden diese Arbeit im abschließenden Kapitel 6 ab.

2 Grundlagen des Softwaretestens

2.1 Software-Qualität

Den Begriff Software-Qualität definiert die DIN EN 9126 folgendermaßen:

„Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen.“

Demnach kann die Software-Qualität nicht mit Hilfe eines Kriteriums beschrieben werden. Im folgenden Kapitel werden die wichtigsten Merkmale des Qualitätsbegriffs erläutert.

2.1.1 Qualitätsmerkmale

Die Qualität wird durch mehrere Eigenschaften charakterisiert. In der Praxis haben sich die folgenden acht Merkmale als relevant erwiesen [HOFF 07]:

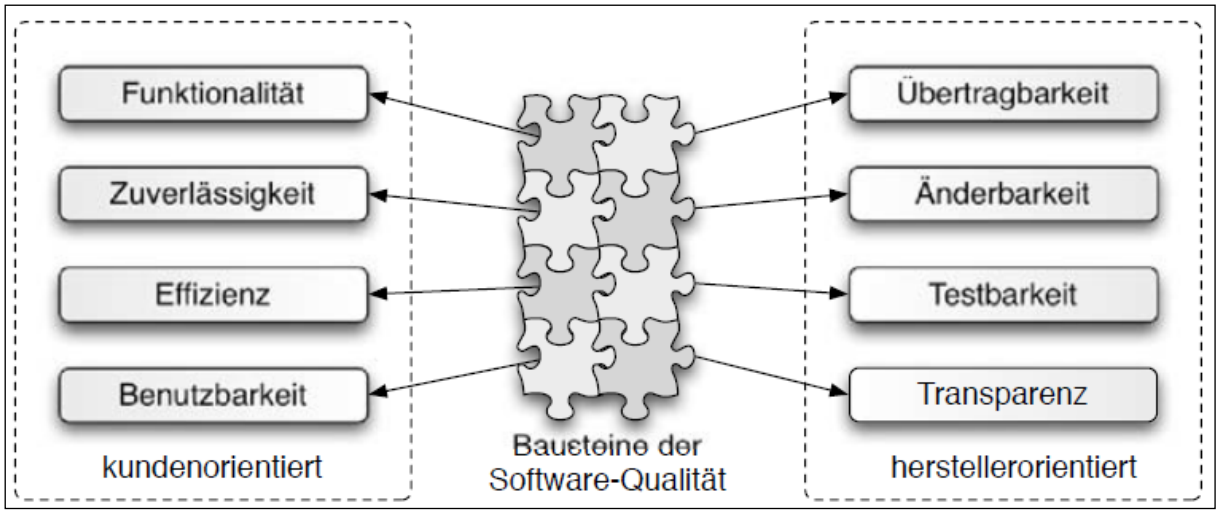


Abbildung 2.1: Qualitätsmerkmale eines Software-Produktes [HOFF 07]

Funktionalität

Die Funktionalität beschreibt, in welchem Maß der Funktionsumfang einer Software tatsächlich erfüllt wird. Falsch umgesetzte oder nicht berücksichtigte Anforderungen, aber auch bereits fehlerhafte Anforderungsspezifikation führen schließlich zu funktionalen Fehlern.

Zuverlässigkeit

Wird die Software in einer sicherheitskritischen, z.B. medizinischen, Umgebung eingesetzt, kann sich das Versagen des Systems unmittelbar auf die Gesundheit des Menschen auswirken. Hier hat also die Zuverlässigkeit der Software eine zentrale Bedeutung. Eine hohe Zuverlässigkeit kann nur unter der Berücksichtigung anderer Qualitätsmerkmale erreicht werden.

Effizienz

Eine Software muss sowohl beim Verwenden der Systemressourcen, als auch beim zeitlichen Verhalten effizient sein. Vor allem die Laufzeit unterliegt bei fast jedem Software-System gewissen Anforderungen. Während bei den meisten Anwendungen die tolerierte Antwortzeit durch den Mensch-Maschine Dialog bestimmt wird, müssen bei Echtzeitsystemen hohe Zeitanforderungen erfüllt werden.

Benutzbarkeit

Die Benutzbarkeit fasst alle Möglichkeiten des Informationsaustausches zwischen dem Mensch und Maschine. Heutzutage besitzen die Software-Systeme individuell auf das jeweilige System abgestimmte Bedienkonzepte. Da nicht nur Experten die Software verwenden, muss die Bedienung und Steuerung verständlich und intuitiv sein.

Übertragbarkeit

Die Übertragbarkeit bestimmt in welchem Maß und unter welchem Aufwand die Software an neue Anforderungen angepasst werden kann. Ein typisches Beispiel ist hier die Portierbarkeit der Anwendung auf neue Betriebssysteme.

Änderbarkeit

Dieses Qualitätsmerkmal bestimmt in wieweit sich die Software nach der Inbetriebnahme verändern lässt. Vor allem muss es möglich sein, Softwarefehler nachträglich zu beheben, ohne dabei viel Zeit und viele Kosten zu verursachen.

Testbarkeit

Während das Merkmal Änderbarkeit ein Kriterium für eine zeitnahe und kostengünstige Fehlerbehebung ist, stellt die Testbarkeit die Voraussetzung für eine rechtzeitige Fehlererkennung. Bedingt durch die Komplexität der heutigen Anwendungen, ist es unmöglich für alle möglichen Eingabekombinationen die Ausgaben zu überprüfen. Bestimmte Techniken helfen dabei geeignete Testfälle zu konstruieren.

Transparenz

Die Transparenz sagt aus wie strukturiert und übersichtlich das Innere der Software aufgebaut ist. Das ist vor allem dann wichtig, wenn Änderungen in die Software eingebracht werden müssen.

Während die Qualitätsmerkmale Funktionalität, Zuverlässigkeit, Effizienz und Benutzbarkeit sind für den Endanwender der Software von größter Bedeutung sind, spielt die Übertragbarkeit, Änderbarkeit, Testbarkeit und Transparenz für der Sicht der Softwareherstellers eine entscheidende Rolle.

2.1.2 Medizinische Software

Bedingt durch das Anwendungsfeld einer medizinischen Software, birgt diese ein Gefährdungspotential gegenüber Patienten und Anwender in sich. Dementsprechend werden durch entsprechende Dokumente regulatorische Anforderungen festgelegt, die bei der Softwareentwicklung und Zulassung berücksichtigt werden müssen (Tabelle 2.1).

Benennung	**Datum**	**Fundstelle**
Richtlinie 93/42/EWG über Medizinprodukte	14.06.1993	Amtsblatt der Europäischen Gemeinschaft L 169
Richtlinie 2007/47/EG Änderungsrichtlinie u.a. zu 93/42/EWG	05.09.2007	Amtsblatt der Europäischen Gemeinschaft L 247
Medizinproduktegesetz (MPG)	14.6.2007	Bundesgesetzblatt
DIN EN 60601-1 Medizinische elektrische Geräte Teil 1: Allgemeine Festlegungen für die Sicherheit einschließlich der wesentlichen Leistungsmerkmale	07.2007	Beuth Verlag
DIN EN ISO 13485 Medizinprodukte, Qualitätssicherungssysteme Besondere Anforderungen für die Anwendung von EN ISO 9001	10.2007	Beuth Verlag
DIN EN ISO 14971 Anwendung des Risikomanagements auf Medizinprodukte	07.2007	Beuth Verlag
DIN EN 62304 (VDE 0750-101) Medizingeräte-Software Software-Lebenszyklus-Prozesse	03.2007	Beuth Verlag

Tabelle 2.1: Normative Vorgaben, die bei der Entwicklung von Medizinprodukten zu berücksichtigen sind.

Speziell für den Softwareentwicklungsprozess wird durch die DIN EN 62304 geprägt. Die Norm erfolgt das Ziel bei der Entwicklung und Wartung von medizinischer Software die Risiken für den Patienten und andere Personen durch die Einhaltung von bestimmten Prozessen zu verringern. Diese Norm stellt somit Anforderungen an die Prozesse der Softwareentwicklung, die bei angemessener Umsetzung erwarten lassen, dass die Software über das erwartete Maß an Sicherheit und Zuverlässigkeit verfügt.

Die Norm setzt das Vorhandensein eines Qualitätsmanagementsystem und eines Risikomanagementprozesses voraus. Diese werden durch die DIN EN 13485 bzw. DIN EN 14971 festgelegt.

Die DIN EN 13485 wurde auf der Grundlage der allgemeineren DIN EN 9001 erarbeitet. Sie ist jedoch weniger umfassend. Einige Anforderungen der DIN EN 9001 ausgeschlossen, weil sie für Medizinprodukte als nicht geeignet betrachtet werden. Dies betrifft vor allem die in der

DIN EN 9001 geforderte Kundenorientierung und Kundenzufriedenheit, sowie den Verbesserungsprozess. Als Erklärung dafür wird in der Norm angeführt, dass „Kundenzufriedenheit“ eine subjektive Anforderung ist und die kontinuierliche Verbesserung des Qualitätsmanagementsystems kein angemessenes Ziel von Bestimmungen für Medizinprodukte sein kann.

Die DIN EN 14971 definiert einen Prozess, bei dem Gefährdungen erkannt, Risiken abgeschätzt, bewertet, kontrolliert und die Wirksamkeit dieser Kontrolle überwacht werden können. Die Norm kann damit als ein Verfahren für ein effektives Risikomanagement dienen. In ihrer Grundlage basiert die Norm auf der Annahme, dass ein Risiko hauptsächlich durch zwei Kernkomponenten definiert wird:

- die Wahrscheinlichkeit des Auftretens eines Schadens (Schadenswahrscheinlichkeit) und,
- die Folgen des Schadens (Schadenshöhe).

Ob ein Risiko vertretbar ist, wird also zum größten Teil von diesen beiden Kernkomponenten und des Bewertungsschemas definiert.

Im weiteren Verlauf dieser Arbeit werden die DIN EN 14971, sowie die DIN EN 62304 „Medizingeräte-Software – Software-Lebenszyklus-Prozesse“ näher erläutert.

2.1.3 Qualitätssicherungsmaßnahmen

Es existieren viele Methoden und Techniken, die eingesetzt werden können, um die Qualität der Software zu sichern bzw. zu verbessern.

Die *konstruktiven Maßnahmen* sind darauf ausgerichtet frühzeitig im Entwicklungsprozess, also bereits bevor eine Software entwickelt wurde, die Qualitätsanforderungen einer Software zu erfüllen. Dazu zählen z.B. Berücksichtigung von Software-Richtlinien und die Erstellung von Dokumentationsstandards.

Im Gegensatz zu konstruktiven Maßnahmen, prüfen die *analytischen Maßnahmen* der Qualitätssicherung das bestehende Software-System bezüglich ihrer Qualitätseigenschaften. Die analytischen Maßnahmen werden in die Teilgebiete *Software-Test, statische Analyse* und *Software-Verifikation* eingeteilt.

Diese Arbeit beschäftigt sich mit dem Software-Test. Die Methoden und Techniken, die bei dem Software-Test eingesetzt werden, zielen dazu die untersuchte Software mit vordefinierten Eingabewerten auszuführen und anschließend die produzierte Ausgaben mit Referenzergebnissen zu vergleichen. Der Software-Test wird im folgenden Kapitel näher beschrieben.

2.2 Klassifizierung von Testtechniken nach Unterscheidungskriterien

In diesem Kapitel wird auf den Begriff *Softwaretesten* eigegangen. Es werden Ziele und die verschiedenen Klassifizierungsarten des Softwaretestens erläutert.

Ein Softwaretest kontrolliert die Qualität der Software und bewertet, in wie weit die festgelegten Anforderungen von dieser realisiert werden. Jede Abweichung von den Anforderungen ist ein Fehler und muss behoben werden. Der Test selbst ist ein entwicklungsbegleitender Prozess und keine einzelne Tätigkeit, die zum Schluss der Entwicklung durchgeführt wird. Der Softwaretest ist vereinfacht gesagt, ein Vergleich der Eingabedaten und den daraus resultierenden Ist-Ergebnissen mit den festgelegten Soll-Ergebnissen. [HOFF 08] So wird dieser auch von der IEEE Standard Glossary of Software Engineering Terminology definiert:

„An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component."[IEEE 90]

Mit steigender Komplexität der Softwareentwicklung hat auch das Testen viele neue Ansätze erhalten und wird in seiner Definition weiter gefasst. Die Klassifizierung der Tests leitet sich aus spezifischen Zielen und Eigenschaften ab. Die Abbildung 2.2 zeigt drei mögliche Klassifikationsarten: Prüfebene, Prüfkriterium und Prüfmethodik.

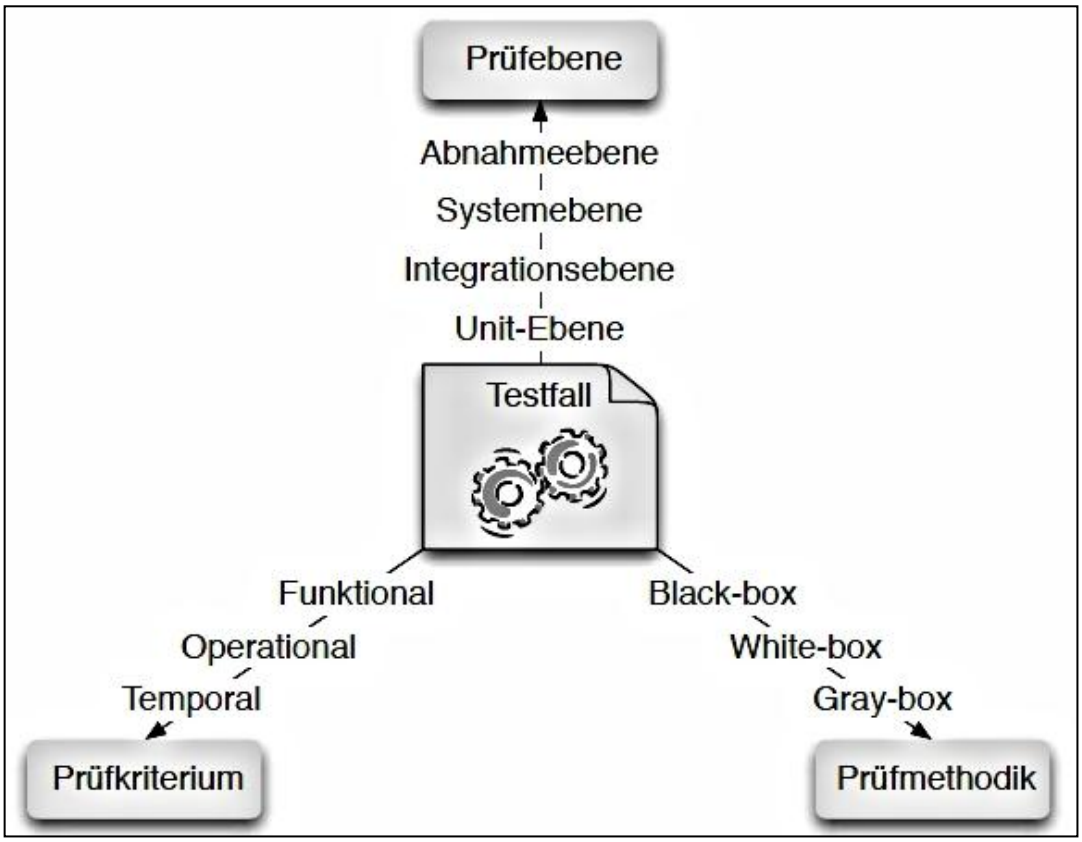


Abbildung 2.2: Klassifikationsarten [HOFF 08]

2.2.1 Prüfebene

Die Prüfebene kann nach der Programmstruktur und der zeitlichen Entwicklungsphase unterteilt werden. Die Klassifizierung innerhalb der Prüfebene orientiert sich dabei an den gängigsten Vorgehensmodellen der Softwareentwicklung, dem V- oder dem W-Modell. Testfälle lassen sich in drei Klassen einordnen: [HOFF 08]

- Unit-Test/ Komponententest
- Integrationstest
- Systemtest

Das V-Modell ist eine Weiterentwicklung des Wasserfallmodells und wird seit 1979 angewandt. Die linke, nach unten führende Linie steht für die Entwicklungsstadien, die mit der Anforderungsdefinition beginnt und der Programmierung der einzelnen Komponenten abschließt. Dem gegenüber stehen die Teststadien, um die das Wasserfallmodell erweitert wurde. [SPILL 08] Es entsteht ein charakteristisches V, welches dem Modell seinen Namen gibt (Abb. 2.3).

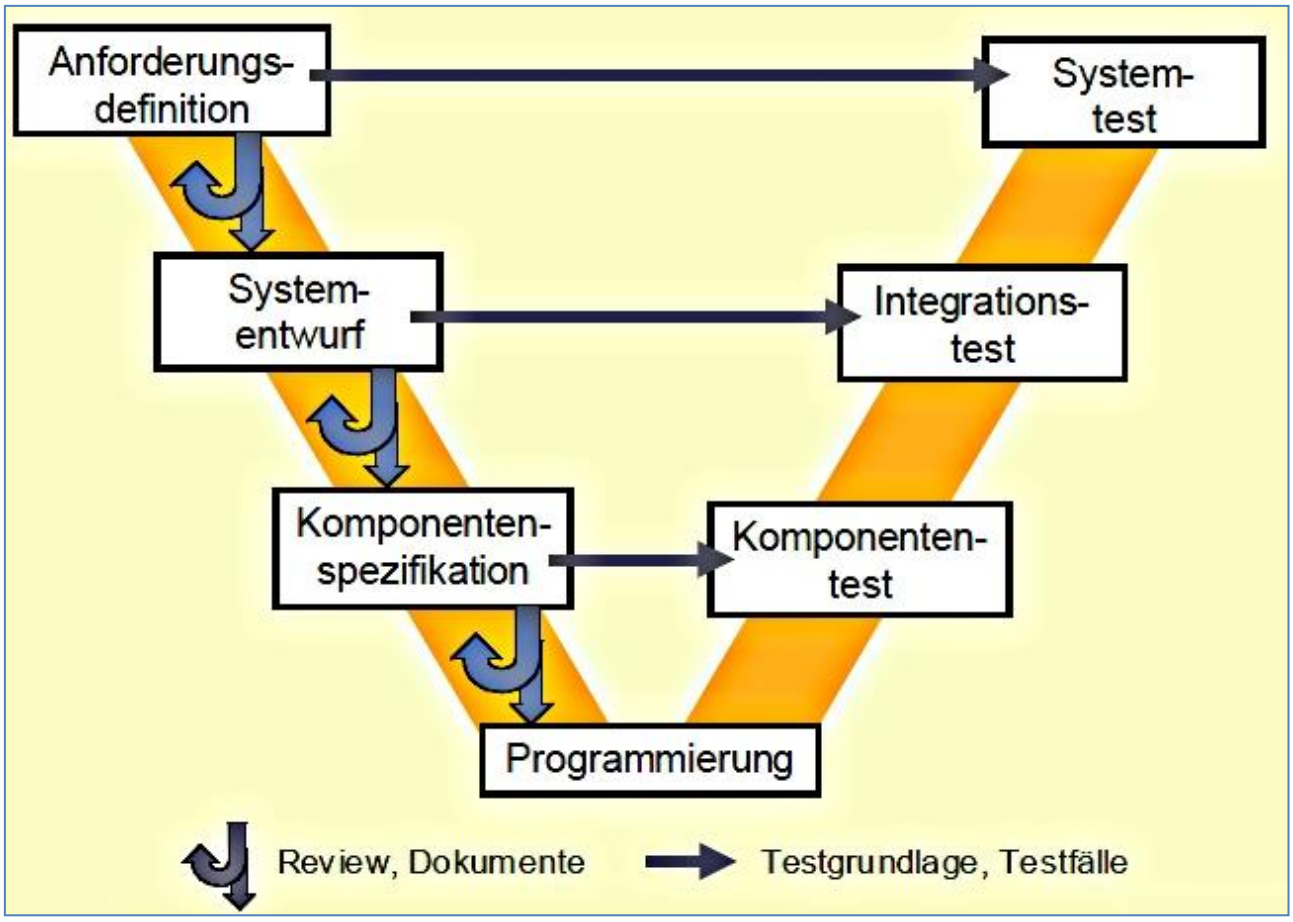


Abbildung 2.3: Das V-Modell [SPILL 08]

Diese Gegenüberstellung trägt zu einer hohen Testabdeckung bei. Denn die Spezifikationen der einzelnen Entwicklungsstadien bilden die Grundlage für die Tests der entsprechenden Testebene.

Ein großes Defizit des V-Modells ist, das die ersten Testaktivitäten im Entwicklungsprozess spät, nämlich nach der Programmierung, beginnen.

Diese und weitere Schwächen des V-Modells haben zur Entwicklung des W-Modells geführt. Der wesentliche Unterschied zwischen den Modellen besteht darin, dass die anfänglichen Testaktivitäten, z.B. Erstellen der Testspezifikationen, schon während der Entwicklungsstadien erfolgen (Abb. 2.4). [SPILL 08]

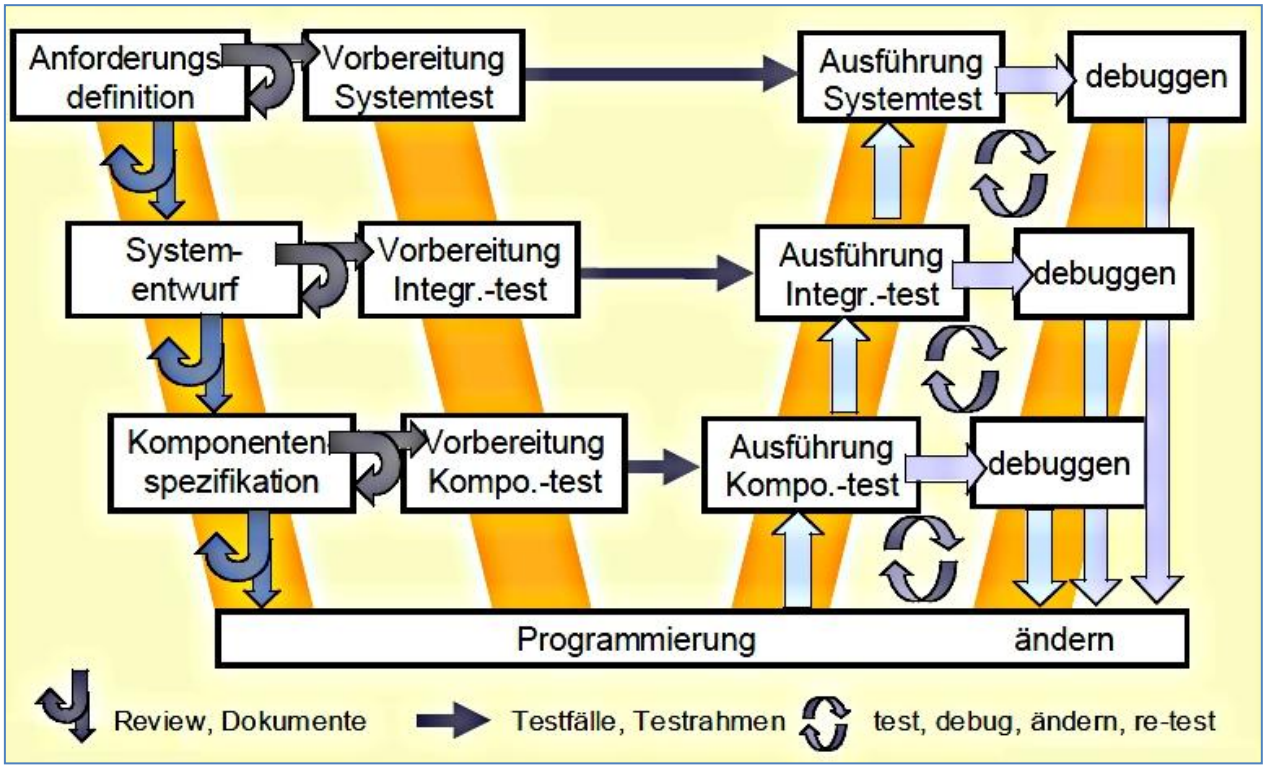


Abbildung 2.4: Das W-Modell [SPILL 08]

Die einzelnen Klassen der Prüfebene werden im Folgenden erläutert. Hier ist anzumerken, dass der Abnahmetest als ein Bestandteil der Prüfebene angesehen werden kann. Da die wesentlichen Aspekte des Abnahmetests bereits mit dem Systemtest abgedeckt werden, wird darauf nicht näher eingegangen. [HOFF 08]

Unit-Test/ Komponententest

Sobald eine Programmeinheit einen bestimmten Umfang erreicht hat um getestet zu werden, spricht man von „Unit“. Da die Größe der Programmeinheit nicht eindeutig definiert ist, wird der Unit-Test allgemein als Komponententest beschrieben. [HOFF 08] Beim dem Test werden einzelne Methoden oder Algorithmen auf ihre Funktionalität überprüft. So kann die Korrektheit der einzelnen Komponenten unabhängig voneinander geprüft werden.

Integrationstest

Der auf den Komponententest folgende Integrationstest prüft die Funktionsfähigkeit der einzelnen Komponenten miteinander. [SPILL 10] Dabei ist entscheidend nach welchem Konzept die einzelnen Bausteine zusammen gesetzt werden. Der Integrationstest kann in drei Hauptkategorien unterteilt werden:

- Big-Bang-Integration
- Strukturorientierte Integration
- Funktionsorientierte Integration

Die Big-Bang-Integration ist dadurch gekennzeichnet, dass alle Module nach der Entwicklung auf einmal integriert werden. Dieser Aspekt hat sowohl Stärken als auch Schwächen. Da auf eine Integrationsstrategie verzichtet wird, entfällt der Einsatz von Testtreibern und Platzhaltern, die bei einer schrittweisen Integration eingesetzt werden müssen. Ein Nachteil besteht darin, dass die Integration erst nach dem Abschluss des letzen Moduls stattfindet und mögliche Fehler daher spät erkannt werden. Das Lokalisieren und Beheben der Fehler wird dadurch erschwert. Nichtsdestotrotz gehört die Big-Bang-Integration zu den meist eingesetzten Techniken.

Bei der strukturorientierte Integration werden, wie es der Name schon sagt, die Komponenten in einer bestimmten Struktur zusammen gebracht. Die strukturorientierte Integration kann in vier Typen unterteilt werden (Abb. 2.5): [HOFF 08]

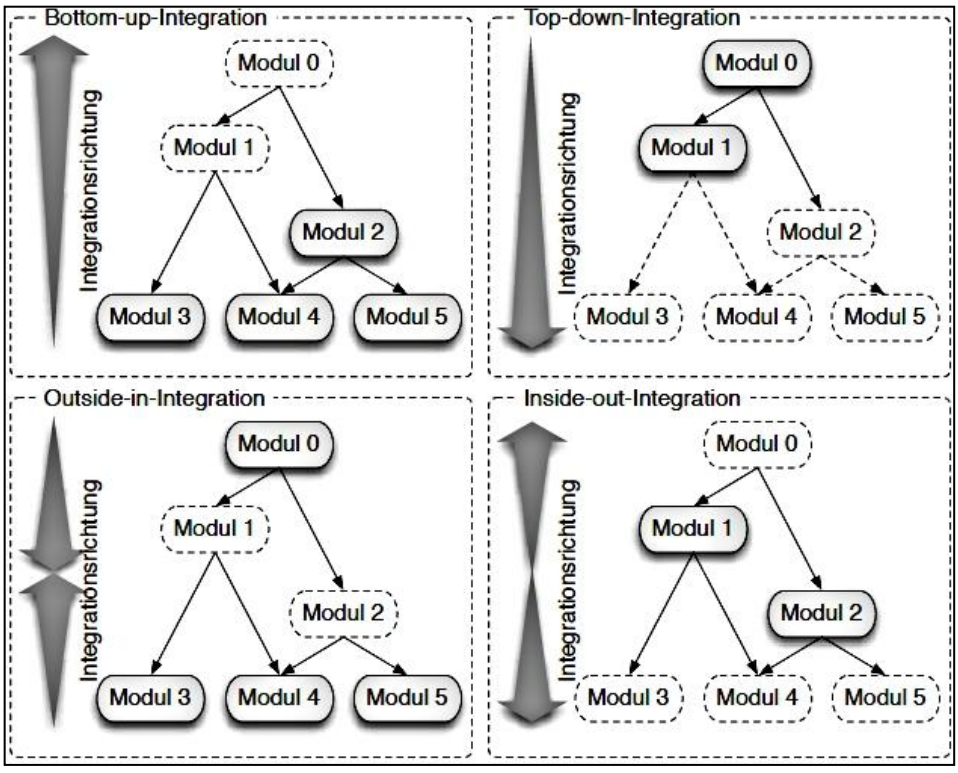


Abbildung 2.5: Integrationsstrategien [HOFF 08]

Bottom-Up-Integration: Dabei werden, einer Hierarchie folgend, zuerst die Basiskomponenten und anschließend die höhergestellten Komponenten integriert.

Top-Down-Integration: Hierbei handelt es sich um eine umgekehrte Bottom-Up-Integration. Die hierarchisch höhergestellten Bausteine werden dabei zuerst integriert.

Outside-In-Integration: Bei dieser Art der Integration werden sowohl die obersten als auch die untersten Komponenten der Hierarchiestruktur als erstes integriert. Im Anschluss folgt die Integration der dazwischen liegenden Module.

Inside-Out-Integration: Hier wird von der Mitte ausgehend nach außen integriert, so dass die hierarchisch hoch- und niedriggestellten Komponenten zuletzt hinzugefügt werden.

Bei der funktionsorientierten Integration werden die Komponenten, wie auch bei der strukturorientierten Integration, in einer bestimmten Reihenfolge zusammengesetzt. Die Integration wird dabei durch funktionale oder operative Faktoren des Systems beeinflusst. Diese Strategie wird ebenfalls in vier Unterarten unterteilt:

Termingetriebene Integration: Hier wird nach dem Prinzip „first come first serve“ integriert, d.h. die Komponenten werden nach ihrer Verfügbarkeit zusammengefügt. So lassen sich Schnittstellenfehler rechtzeitig erkennen.

Risikogetriebene Integration: Das Prinzip dieser Strategie ist „hardest first“, dabei werden die Komponenten zuerst integriert, deren Ausfall ein höheres Risiko nach sich bringt.

Testgetriebene Integration: Hierbei orientiert man sich an konkreten Testfällen. Es werden also zuerst die Komponenten integriert, die für die Testausführung notwendig sind.

Anwendungsgetriebene Integration: Diese Integrationsart ist der testgetriebenen Integration ähnlich. Der Unterschied besteht darin, dass die Komponenten nicht nach einem Testfall sondern nach einem „use case“, also einem Anwendungsszenario integriert werden. [HOFF 08]

Systemtest

Sobald die nötigen Komponenten integriert sind, kann mir dem Systemtest begonnen werden. Dabei wird das System als Ganzes auf die festgelegten Softwareanforderungen überprüft, was dieses aus der Sicht des Kunden/Anwenders geschieht. Das System wird über eine externe Schnittstelle (z.B. GUI) getestet, so dass die interne Struktur des Systems für den Test keine Rolle spielt. [WINT 01]

Im Vordergrund steht die Erkennung von Mängeln und Fehlern, die aufgrund von falscher oder lückenhafter Umsetzung von Anforderungen ins System eingebracht worden sind. Erschwert wird der Systemtest unter anderem durch missverständliche Anforderungen des Kunden oder durch unvorhersehbare Fehler. [SPILL 10]

2.2.2 Prüfkriterien

Das zweite Merkmal der Klassifizierung geht im Gegensatz zur Prüfebene auf die inhaltlichen Aspekte eines Tests ein. Die Softwaretests lassen sich dabei in drei Kategorien einteilen „Funktional“, „Operational“ und „Temporal“. Auf diese Kriterien soll im Folgenden näher eingegangen werden.

Funktionaler Test

Dieser Test stellt sicher, dass die vorgegebenen Eingabedaten korrekte Ausgabedaten nach sich ziehen. Dabei dienen funktionale Anforderungen als Referenz. Die funktionalen Anforderungen berücksichtigen laut DIN EN 9126 folgende Teilmerkmale: Angemessenheit, Richtigkeit, Ordnungsmäßigkeit, Interoperabilität und Sicherheit. Diese Merkmale müssen auch bei der Erstellung von Testfällen berücksichtig werden. [SPILL 10] In der Softwareentwicklung ist es üblich, dass der Funktionstest mit dem Begriff „Testen“ gleichgesetzt wird.

Operationaler Test

Bei diesem Test stehen die reibungsfreie Inbetriebnahme, die Benutzerfreundlichkeit sowie die Sicherheit der Daten im Vordergrund.

Temporaler Test

Der Temporale Test legt das Zeitverhalten des Systems offen. Dabei wird auf verschiedenen Lastebenen getestet. Bei dem Laufzeittest wird die Ausführungsgeschwindigkeit der einzelnen Funktionen im Normalbetrieb gemessen. Mit dem Lasttest bestimmt man das Systemzeitverhalten bei steigender Systemlast. An die Grenze wird das System mit dem Stresstest gebracht. Dabei bewertet man das Verhalten des Systems bei Überlast. [HOFF 08] [SPILL 10]

2.2.3 Prüfmethodik

Softwaretests lassen sich auch nach dem Informationsstand über den inneren Aufbau des zu testenden Systems klassifizieren. So ergibt sich eine Einteilung in Black-Box- und White-Box-Tests. [SPIL 08] Beide Methoden werden bei der Konstruktion von Testfällen eingesetzt.

Black-Box-Test

Bei dieser Methode werden die Testfälle nur anhand der Spezifikation, also ohne Kenntnisse des Quellcodes, erstellt. Der Quellcode wird dabei als eine Black-Box betrachtet und bleibt somit unberücksichtigt. Dieser Test prüft, ob der Entwickler die gestellten Anforderungen verstanden und richtig umgesetzt hat. Da der Black-Box-Test im weiteren Verlauf der Arbeit wichtig ist, werden die zugehörigen Konstruktionsverfahren im Folgenden näher beschrieben.

Äquivalenzklassentest:

Der Äquivalenzklassentest dient dem Ziel mit möglichst wenigen Testfällen eine hohe, wenn möglich vollständige, Abdeckung der gestellten Anforderungen zu erreichen. Die Durchführung erfolgt in zwei Schritten.

Im ersten Schritt erfolgt die Äquivalenzklassenbildung. Dabei werden die Wertebereiche der Eingabe- und Ausgabedaten festgelegt und anhand dieser die Äquivalenzklassen gebildet. Somit wird die Anzahl von möglichen Eingaben reduziert. Es wird zwischen gültigen und ungültigen Äquivalenzklassen unterschieden. Im Anschluss folgt die Testfallkonstruktion unter Berücksichtigung der vorher festgelegten Äquivalenzklassen. Aus jeder

Äquivalenzklasse wird ein beliebiger, repräsentativer Wert ausgewählt und für den Testfall verwendet. [HOFF 08]

Grenzwertanalyse:

Die Grenzwertanalyse basiert auf den Äquivalenzklassen und wird als deren Erweiterung angesehen. Der Unterschied besteht darin, dass Grenzwerte der gebildeten Klassen und nicht beliebige Werte für den Testfall übernommen werden. Erfahrungsgemäß treten Fehler im Programm häufig an den Übergängen der Äquivalenzklassen auf. [SPILL 10]

Zustandsbasierender Softwaretest:

Bei dieser Methode wird auf der Basis eines Zustandsautomaten ein Testfall gebildet. Dabei werden die Ereignisse, die verschiedene Zustandsänderungen des Systems hervorrufen in einem Zustandsautomaten dargestellt. [SPILL 10]

Use-Case-Test:

Anhand von Use-Case-Diagrammen werden die Anwendungsfälle veranschaulicht. Hier wird beschrieben wie ein Anwender mit bestimmten Aktionen zu einen gewünschten Ziel kommt, d.h. das Diagramm zeigt das System aus Kundensicht. [SPILL 10]

Entscheidungsbasierter Softwaretest:

Wie bei einem Use-Case-Test wird die Software auf der Systemebene getestet. Ein Ursache-Wirkungs-Graph zeigt dabei die logischen Zusammenhänge aus Ursachen und daraus resultierenden Wirkungen. Die Eingaben werden durch Operatoren AND, OR und NOT verkettet. Aus einem Ursache-Wirkungs-Graphen lässt sich eine Entscheidungstabelle erstellen. Sobald die Zahl der Eingaben und Aktionen sich erhöht, werden sowohl der Graph als auch die Tabelle unübersichtlich. [SPILL 10]

White-Box-Test

Bei dem White-Box-Test wird der Testfall, anders als beim Black-Box-Test, unter Berücksichtigung der inneren Funktionsweise und Struktur der zu testenden Anwendung erstellt. Die Testfälle müssen dabei den zuvor festgelegten Überdeckungsgrad des Quellcodes erreichen.

2.2.4 Dynamisches und statisches Testen

Die Einteilung nach Black-Box- und White-Box-Tests ist laut [LIGG 09] nicht ausreichend. Demnach sollte nach dynamischen und statischen Tests unterschieden werden.

Statischer Test: Im Gegensatz zum dynamischen Test, wird bei einem statischen Test die Software nicht ausgeführt. Hierbei handelt es sich um eine Analyse der Software, die z.B. von mehreren Entwicklern (Review) oder mit Einsatz von speziellen Werkzeugen durchgeführt wird. [SPILL 10]

Dynamischer Test: Das Grundprinzip des dynamischen Tests ist die Ausführung der zu testende Software mit definierten Eingabedaten. Dabei werden für jeden Testfall Eingabedaten und erwartete Ausgabedaten festgelegt. Der dynamische Test kann in Black-Box- und White-Box-Test unterteilt werden (Kapitel 2.2.3).

2.2.5 Limitierung der Software-Tests

Das Testen ist mühsam und stellt die Tester regelmäßig vor neue Herausforderungen. Die folgenden Gründe sind für die Schwierigkeiten beim Testen verantwortlich.

Missverständliche oder fehlende Anforderungen:

Das Kernstück des Testens ist der Abgleich zwischen den Ist- und den Sollergebnissen. Somit ist nur dann sinnvoll wenn der Sollzustand eindeutig bestimmt worden ist. Da zu Beginn eines Projektes die Anforderungsspezifikation nicht immer vollständig erfasst wird, ist eine Testfallkonstruktion nur eingeschränkt möglich.

Programmkomplexität:

Bei einem manuellen Test ist es unmöglich die Anwendung komplett zu testen. Deshalb ist es wichtig, dass die relevanten Bereiche der Anwendung identifiziert und getestet werden.

Mangelnde Werkzeugunterstützung:

Es gibt wenige Werkzeuge, die den Tester bzw. Entwickler bei der Testfallkonstruktion unterstützen. Die meisten Tools beschränken sich auf die automatisierte Testausführung. Das heißt, dass die Testfälle nur so genau sind, wie der Entwickler diese erstellt hat.

Fehlende Management-Unterstützung:

Das Management legt die Verantwortung für das Testen komplett in die Hände der Entwickler, so dass Testen bei der Planung unberücksichtigt bleibt. Dabei ist die Planung bzw. die Zeitabschätzung beim Test von großer Bedeutung.

Ausbildungs- und Fortbildungsdefizite:

Auch während der Ausbildung an Hochschulen wird das Testen vernachlässigt. Der Schwerpunkt liegt dabei eher auf dem Ergänzen von Testfällen.

Zeitprobleme:

Der Zeitaufwand beim Testen wird sehr häufig unterschätzt. Das führt dazu, dass die Anzahl von ausgeführten Testfällen reduziert wird. Es bleibt oft nur bei einem obligatorischen Test mit einem minimalen Aufwand.

Bei all diesen Maßnahmen wird lediglich das Vorhandensein von Fehlern aufgezeigt jedoch nicht die Richtigkeit der Software. [HOFF 08]

2.3 Regressionstest

Im Entwicklungsprozess wird die Software vielfach korrigiert, geändert und erweitert, es entsteht jedes Mal eine neue Version des ursprünglichen Softwareprodukts. Um sicherzustellen, dass die Änderung einwandfrei funktioniert bzw., dass der Rest der Anwendung von der Änderung nicht negativ beeinflusst wurde, muss diese erneut getestet werden. Denn durch eine Modifikation, können Fehler an ganz anderen Stellen in der Anwendung ans Licht kommen. [HOFF 08] Um diese zu detektieren wird ein Regressionstest eingesetzt. Er hat das Ziel sicherzustellen, dass durch die vorgenommenen Änderungen keine neuen Fehler entstanden sind und, dass das modifizierte System weiterhin die Anforderungen erfüllt.

Bei diesem Test werden alle oder bestimmte Testfälle wiederholt. Da die Zahl der Testfälle in Abhängigkeit von der Anwendung relativ hoch werden kann, muss bei einem Regressionstest

der Testumfang genau festgelegt werden. Die notwendigen Testfälle für den Regressionstest müssen ausspezifiziert und mit Soll-Ergebnissen versehen werden. Bei der manuellen Ausführung von Regressionstests werden dokumentierte Testschritte in der vorgegebenen Reihenfolge abgearbeitet. Die hierbei resultierenden Ausgaben der getesteten Software werden anschließend mit den erwarteten Ergebnissen verglichen. Ein direkter Bezug zu den Ergebnissen des vorherigen Testdurchlaufes findet also nicht statt.

Je größer die Anzahl der durchzuführenden Testfälle ist, desto größer ist die Wahrscheinlichkeit, dass der Tester eine Abweichung zwischen dem Soll- und dem Ist-Ergebnis übersieht. [SPILL 10] Aufgrund des Wiederholungscharakters und der Häufigkeit dieser Wiederholungen bietet es sich also an, für den Regressionstest Testautomatisierung einzusetzen. Dadurch können nicht nur Zeit und Kosten gespart, sondern auch die Qualität des Tests erhöht werden. Es ergeben sich somit wirtschaftliche und technische Vorteile. [LIGG 09]

2.4 GUI-Test

In der letzten Zeit hat der Anteil an Softwareprodukten, die eine GUI (Graphical User Interface) besitzen stark zugenommen. Viele Anwendungen können nur noch über die GUI bedient werden, so dass auch der Softwaretest hierüber ausgeführt werden muss.

Eine grafische Benutzeroberfläche reflektiert die Funktionalität der Anwendung und wird über diese bedient. Daher ist die Besonderheit beim Test der GUI, dass nicht nur die Oberfläche, sondern die komplette Anwendung geprüft werden kann.

Die Oberfläche einer Anwendung ist eine Ansammlung von verschiedenen GUI-Objekten. Die Objekte besitzen definierte Eigenschaften und reagieren auf bestimmte Aktionen. Bei dem GUI-Test werden diese bedient. Zum Beispiel bestehen die physikalischen Attribute der GUI-Objekte aus statischen und dynamischen Eigenschaften. Statische Eigenschaften, wie z.B. Beschriftungen, werden in der Phase des Anlegens der GUI-Objekte vergeben und verändern sich nicht während des Programmablaufs. Die dynamischen Eigenschaften verändern sich zur Laufzeit eines Programmes. Typische dynamische Eigenschaft eines GUI-Objektes ist z.B. der Status eines Radio-Buttons (aktiviert oder deaktiviert). Ein GUI-Test kann unter Beachtung bestimmter Anforderungen automatisiert werden.

2.5 Testautomatisierungen

Wie schon im Kapitel 2.3 erwähnt, steigt der manuelle Testaufwand mit jeder Weiterentwicklung der Software. Eine Möglichkeit der Aufwand- und Kostenminimierung ist das automatisierte Testen. Die Testautomatisierung bringt folgende Vorteile mit sich [BRCA 03]:

- der Aufwand der Regressionstests wird minimiert
- durch die Zeitersparnis können Tests mehrmals und öfter durchgeführt werden
- es können Tests durgeführt werden die manuelle nur eingeschränkt oder gar nicht möglich wären
- besserer Einsatz der Ressourcen
- die Qualität und die Wiederholbarkeit von Test bleibt gleich oder wird erhöht

Zwar erhöht sich die Qualität und Leistungsfähigkeit der Software, die Testautomatisierung macht jedoch nicht immer Sinn. Folgende Tätigkeiten lassen sich automatisieren:

- Erstellung der Testfälle
- das Testen selbst
- Auswertung der Tests
- Dokumentation der Testfälle

All den Vorteilen stehen eine Reihe Nachteile gegenüber. Es können z.B. gewisse Tests nicht automatisierte werden. Außerdem besitzen Werkzeuge keine Kreativität. So können nur durch vorgegebene Ergebnisse Abweichungen von der Spezifikation aufgedeckt werden. Der automatisierte Test ist also nur so genau wie in den Automatisierungswerkzeugen festgelegt. Ein Tester ist flexibler und kann beim Test eventuell mehr Fehler entdecken. Eine Testautomatisierung macht außerdem keinen Sinn wenn z.B. die Tests selten ausgeführt werden oder sich die Software häufig und stark verändert. Daraus ergibt sich, dass der Einsatz der Automatisierungswerkzeuge schon in der Konzeptionsphase eines Projektes geplant werden muss. [BRCA 03]

2.6 Fehlermetriken

Eine systematische Analyse von Fehlern während und nach der Softwareentwicklung liefert wertvolle Informationen über den bisherigen Zustand der eingesetzten Entwicklungsprozesse und QS-Maßnahmen. Das kann helfen mögliche Verbesserungspotentiale abzuleiten. Als Messinstrument, um die Güte bisheriger Entwicklung und Qualitätssicherung zu messen eignen sich sog. Fehlermetriken.

In diesem Kapitel wird zuerst auf den Begriff Fehler und Fehlerverteilung eingegangen. Anschließend werden die Metriken *Fehlerstrommessung (FSM)* und *Defect Detection Effectiveness (DDE)* näher erläutert.

2.6.1 Fehler

Eine große Rolle in der Software-Qualitätssicherung spielt die Fehlervermeidung bzw. -behebung. Um zu unterschneiden welches Verhalten gewünscht und welches fehlerhaft ist, muss zu Beginn der Entwicklung das Sollverhalten festgelegt werden. Somit wird jede Abweichung vom Sollzustand als ein Softwarefehler definiert. [SPILL 08]

Um die Fehler klassifizieren zu können, ist es hilfreich eine Übersicht darüber zu erstellen, wo, welche Fehler und in welchem Umfang aufgetaucht sind. Eine Möglichkeit dieser Übersicht ist die sogenannte Fehlerverteilung. Diese stellt die Fehler in Abhängigkeit zu den unterschiedlichen Entwicklungsphasen dar. Aus verschiedenen Studien wird deutlich, dass der große Teil der Fehler auf die Spezifikation zurück zu führen ist. [BORL 06], [DENG 07], [ROSE 06]

Bei [DENG 07] finden sich 44% der Fehler in der Spezifikation, bei [BORL 06] sind es 56% und bei [ROSE 06] sogar 63%.

Es gilt: *„Je früher im Entwicklungsprozess der Fehler auftritt und je später er entdeckt wird, umso aufwändiger wird es, den Fehler zu beheben“*. [THALL 02] Aus diesem Grund sollten Fehler schnellstmöglich entdeckt und im Anschluss behoben werden.

2.6.2 Fehlerstrommessung

Die Fehlerstrommessung dient als Messinstrument, welches es ermöglicht Fehler systematisch zu erfassen und die Effektivität der eingesetzten QS-Maßnahmen im Bezug auf die Fehlererkennung zu bestimmen. [FREI 05] Die FSM wird in folgende Schritte unterteilt:

1. Voraussetzungen und Kontext
2. Datenerhebung und -aufbereitung
3. Identifizierung der Fehlerquellen und -senken
4. Ergebnisse und Auswertungen
5. Repräsentative Fehler

1. Voraussetzungen und Kontext

Um die FSM effektiv einsetzen zu können, muss geprüft werden, ob die hierfür notwendigen Voraussetzungen in dem Unternehmen erfüllt werden. Das Unternehmen muss über einen „gelebten“ Entwicklungsprozess verfügen. Darüber hinaus ist es erforderlich, dass zumindest der Großteil der im Entwicklungsprozess und Betrieb aufgetretenen Softwarefehler dokumentiert wird. Desweiteren muss festgelegt werden bei welchen Softwareprodukten die FSM eingesetzt wird.

2. Datenerhebung und -aufbereitung

Nach dem nun die Voraussetzungen geprüft und der Kontext der Untersuchung festgelegt wurde, müssen die notwendigen Daten bestimmt werden.

Alle Einträge zu der Software, die bis jetzt dokumentiert wurden, sind nach bestimmten Kriterien zu filtern. Da die Einträge meistens in Fehlerverwaltungstools verwaltet werden, hängt es von dem jeweiligen Tool ab, nach welchen Kriterien gefiltert werden kann. Z.B. in dem Tool Bugzilla müssen die Suchparameter für die Kriterien „Product“, „Severity“, „State“ und „Resolution“ entsprechend eingestellt werden. Es müssen Problemfälle bei den es sich nicht um Softwarefehler handelt, Ideen, Vorsichtsmaßnahmen, und Änderungsanforderungen raus gefiltert werden. Das Ziel ist es Einträge zu haben bei den es sich tatsächlich um Softwarefehler handelt. Am Ende der Datenerhebung sind die restlichen Bugzilla-Einträge, die nicht in als Softwarefehler identifiziert wurden, nochmal kritisch zu betrachten. Das soll mögliche Fehler bei der Klassifizierung der Einträge aufdecken.

3. Identifizierung der Fehlerquellen und -senken

Nachdem tatsächliche Softwarefehler für das jeweilige Software-Produkt gefiltert wurden, müssen die Fehler nun nach den Eigenschaften „*Fehlerquelle*“ und der „*Fehlersenke*“ klassifiziert werden. Dabei charakterisiert die Fehlerquelle die Phase der Entwicklung in welcher der Fehler eingebracht wurde. Die Fehlersenke beschreibt schließlich in welcher Phase der Fehler gefunden wurde. Die Fehlersenke ist somit an eine QS-Maßnahme gebunden.

Da die Fehlerklasse eines Fehlers nicht analytisch ermittelt werden kann, gelten Klassifikationen als zuverlässig, wenn Fehler von verschiedenen Personen gleich klassifiziert werden. Dabei werden zwei unabhängige Klassifikationen einer Menge von Fehlern verglichen. Als Faustregel gilt, dass eine gute Klassifikation vorliegt, wenn der Übereinstimmungsgrad bei der Klassifikation größer ist als 0,6. [EMAM 98]

4. Ergebnisse und Auswertungen

Die Ergebnisse werden tabellarisch oder graphisch dargestellt. Die anschließende Auswertung zeigt schließlich welche Fehlerquellen und Fehlersenken mehr oder weniger erfolgreich waren. Außerdem können Korrelationen zwischen den einzelnen Fehlerquellen und Fehlersenken dargestellt und ausgewertet werden.

5. Repräsentative Fehler

Die vorherige Auswertung der Ergebnisse wird schließlich durch eine Zusammenstellung repräsentativer Fehler verdeutlicht.

2.6.3 Defect Detection Effectiveness

Die Anzahl der aufgetretenen Fehler nach der Produktfreigabe (Release) in Relation zu den vor der Freigabe gefundenen Fehler, die sog. Defect Detection Effectiveness (DDE) ist ein wesentlicher Indikator für die Effektivität der jeweiligen eingesetzten QS-Maßnahmen und die Qualität des Produktes. Berechnet wird DDE folgendermaßen:

$$DDE = F_t / (F_t + F_r)$$

F_t = Fehleranzahl beim Test; F_r = Fehleranzahl nach dem Release

Bei einem DDE von 90% und höher, liegt die Effektivität der QS-Maßnahme in einem guten Bereich. [IEEE 03]

3 Analyse

3.1 Vorgehen

Das Ziel dieser Arbeit ist es, ein Konzept zum Testen einer medizinischen Software auszuarbeiten und diesen bei einer Software von seca zu implementieren. Eine Analyse des Ist-Zustandes im Unternehmen und die Identifizierung von Schwachstellen und Problemen sind notwendig, um Verbesserungsvorschläge formulieren zu können.

Im ersten Teil des Kapitels wird das projektspezifische Umfeld von seca durch eine Bestandsanalyse näher beschrieben. Dadurch können mögliche Schwachstellen und Risiken des Testprozesses schon vorab identifiziert werden. Die hierzu notwendige Datenerhebung wurde mit Hilfe von unternehmensinternen Dokumenten durchgeführt.

Im zweiten Teil des Kapitels wird eine Fehleranalyse durchgeführt. Dabei wird eine Fehlerstrommessung (FSM) auf drei medizinische Softwareprodukte von seca angewendet. Die FSM ist eine an dieser Stelle angemessene Metrik, weil sie deutlich macht, in welcher Entwicklungsphase Fehler entstanden sind und durch welche Qualitätssicherungsmaßnahmen (bezogen auf die Entwicklungsphasen) diese erkannt wurden. Die Metrik *Defect Detection Effectivness* wird anschließend eingesetzt, um die Effektivität der bisher durchgeführten Qualitätssicherungsmaßnahmen zu bestimmen. Die Ergebnisse der Fehleranalyse decken die Schwächen der bisher eingesetzten Qualitätssicherungsmaßnahmen auf. Diese werden anhand ausgewählter, repräsentativer Fehler verdeutlicht.

Die Datenerhebung zur Fehleranalyse geschah anhand von Einträgen des Bugtrackingtools Bugzilla. Die Identifizierung von relevanten Bugzilla-Einträgen, sowie die Klassifizierung und Zuordnung dieser nach Fehlerherkunfts- und Fehlererkennungsphasen haben zwei Mitarbeiter von seca unabhängig voneinander durchgeführt. Die anschließende Zuverlässigkeitsanalyse konnte zeigen inwieweit die ermittelten Ergebnisse der beiden Domänenexperten übereinstimmen.

Die abschließendenden Verbesserungsvorschläge dienen als eine Grundlage für ein Testkonzept, welches im nächsten Kapitel umgesetzt wird.

3.2 Bestandsanalyse

Die in diesem Kapitel präsentierte Bestandsanalyse orientiert sich an dem Unternehmen seca. Sie umfasst die Vorstellung des Unternehmens und der Fachabteilungen, den Aufbau des Software-Entwicklungsprozesses, die Darstellung der Risikoanalyse, sowie eine ausführliche Beschreibung des Testprozesses.

3.2.1 Das Unternehmen und die Abteilungen

Die seca GmbH & Co.KG ist ein Familienunternehmen mit einer über 170-jährigen Tradition im Bereich medizinische Waagen und Maßsysteme. Das Unternehmen ist Weltmarktführer für medizinisches Wiegen. seca Produkte werden in über 110 Länder vertrieben. Mit den neun Niederlassungen weltweit, sowie einem globalen Netzwerk von Exklusivdistributoren, verkauft seca seine Produkte in über 110 Länder.

Die Firmenzentrale befindet sich in Hamburg und umfasst die Abteilungen Forschung und Entwicklung, Quality Services, Marketing and Sales, Finance and Services, Materials Management and Logistics, sowie Produktion/Process Technology Management. Die Geschäftsführung liegt in den Händen der beiden Brüder Robert und Frederik Vogel.

Die Abteilung F&E ist ihrerseits in die Sparten Software Engineering (SE), Electronic Engineering (EE) und Mechanical Engineering (ME) untergliedert. Diese Arbeit entstand in Kooperation mit der Softwareabteilung, weshalb der vorliegende Abschnitt diese nun näher vorstellt.

Das Team SE umfasst 5 interne und 2 bis 4 externe (Dienstleister) Mitarbeiter und befasst sich mit der Entwicklung von Software, die im medizinischen Umfeld verwendet wird. Die entwickelten Produkte dienen u.a. zur Diagnostik- und Therapieunterstützung, zur Verwaltung der Patienten- und Messdaten, sowie Übertragung von Messdaten medizinischer Messgeräten zum Patientendatenmanagementsystem (PDMS) und werden zum größten Teil mit Hilfe von C#, unter Verwendung von .NET Frameworks, entwickelt. Neben diesen kundenrelevanten Produkten entwickelt die Abteilung an Softwarewerkzeugen, welche die Produktion und Wartung von Medizingeräten der Firma seca unterstützen. Außerdem werden von der Abteilung alle Prozesse und Werkzeuge, welche die Softwareentwicklung betreffen, ausgearbeitet. Dazu zählen z.B. Versionsverwaltung in Subversion, Fehlerreporterfassung in Bugzilla, sowie kontinuierliche Integration durch Hudson. Diese Prozesse und Werkzeuge gelten für das gesamte Unternehmen.

3.2.2 Softwareprodukte

seca hat bis zum jetzigen Zeitpunkt drei kundenrelevante Softwareprodukte auf den Markt gebracht. Die Produkte werden durch neue Funktionalitäten, Change Requirements (CR), sowie Fehlerbehebungen ständig weiterentwickelt, was bei jedem dieser Produkt zu etwa 2 bis 3 Releases pro Jahr führt.

Bei dem ersten Produkt, seca emr flash 101, handelt es sich um eine PC-Software, die zwar nicht als medizinisches Produkt klassifiziert ist, allerdings nach dem gleichen Entwicklungsmodell wie eine medizinische Software entwickelt wurde, um grundlegende Qualitätsanforderungen zu befriedigen.

Die Software wird wie die anderen beiden Produkte in Krankenhäusern, Arztpraxen und stationären Pflegeeinrichtungen eingesetzt und dient zum Empfang der Gewichts- und Langenmessungen, sowie deren Übertragung zu einem Patientendaten-Managementsystem (PDMS).

Das zweite Produkt, seca analytics mBCA 115, ist eine medizinische PC-Software, die hauptsächlich in Krankenhäusern, Arztpraxen und stationären Pflegeeinrichtungen zum Einsatz kommt. Die Software dient zur Erfassung und Verwaltung von Gewichts-, Längen- und bio-

elektrischen Impedanzmessungen, sowie zur automatischen Berechnung daraus ableitbarer Parameter, wie z.B. dem BMI (Body Mass Index) und der FM (Fettmasse). Die Ergebnisse werden grafisch dargestellt und unterstützen den behandelnden Arzt bei Diagnose- und Therapieentscheidungen.

Das dritte Produkt, seca 515/514 Embedded-Software, ist eine medizinische Software, die in dem medical Body Composition Analyzer (mBCA) integriert ist und das medizinische Gerät steuert. Sie hat im Vergleich zu der PC-Software seca analytics mBCA 115 einen geringeren Funktionsumfang, dient aber ebenfalls zur Unterstützung des behandelnden Arztes bei der Diagnose und Therapie. Da es das weltweit erste Bioimpedanzmessgerät ist, welches für den medizinischen Einsatz konzipiert und entwickelt wurde, hat es strategische Relevanz für seca. Das folgende Kapitel stellt das Produkt aus diesem Grund näher vor. Es werden nun ein technischer Entwurf vorgestellt, sowie Funktionen des Gerätes näher beschrieben.

Technischer Entwurf

Der mBCA besteht aus den Kernkomponenten BIA-Messmodul, Wägezelle und Funkmodul. Diese werden über eine RS232-Schnittstelle an ein Keith & Koep Embedded Baseboard angebunden. Auf dem integrierten Rechnermodul Trizeps 4 läuft das Betriebssystem Windows Embedded CE 6.0, auf dem die Embedded Software seca 515/514 ausgeführt wird. Außerdem besitzt das Baseboard eine USB-, Ethernet-, sowie Touchscreen-Schnittstelle. Die folgende Abbildung 3.1 zeigt eine schematische Darstellung des mBCA‘s.

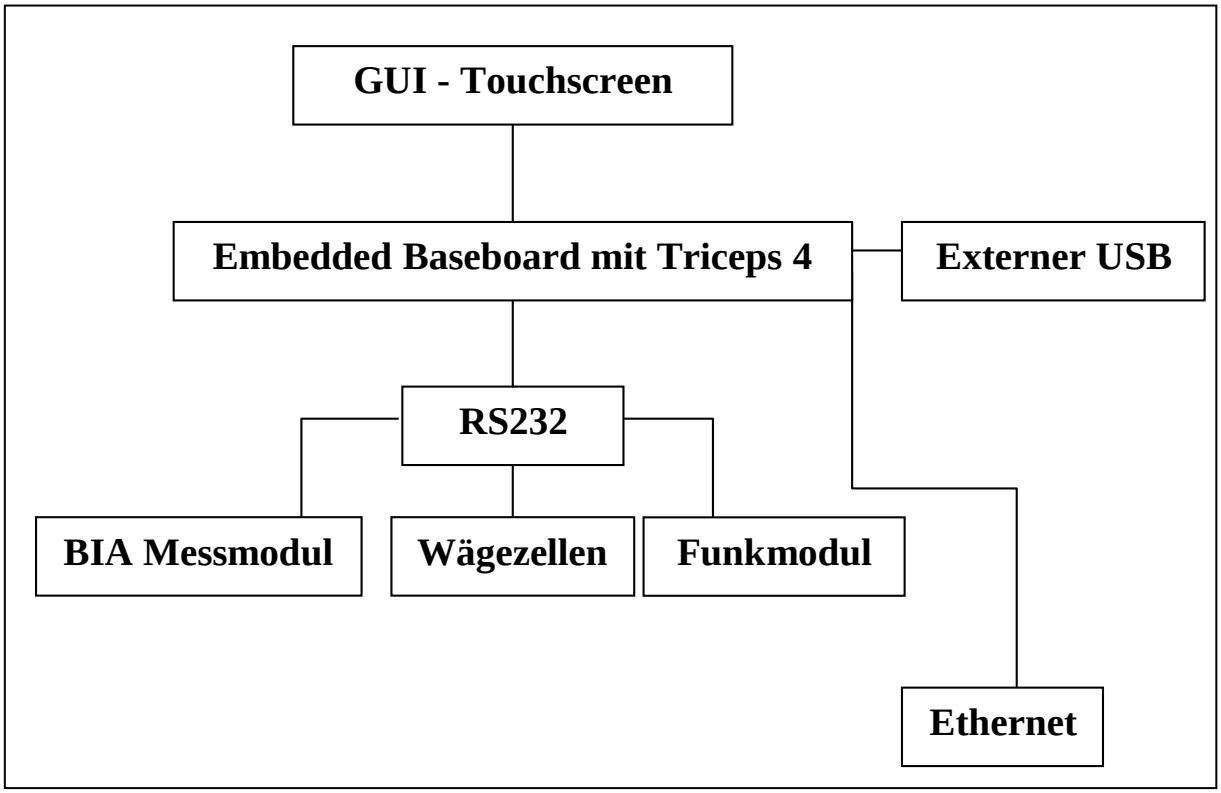


Abbildung 3.1: Der vereinfachte Aufbau des mBCA’s. Dabei sind die Kernkomponenten des Gerätes

Funktionsbeschreibung

Der mBCA kombiniert vier Grundfunktionalitäten: Erfassung von Gewicht und Größe, Bioelektrische Impedanzanalyse, Patientenakten Verwaltung und Auswertung der Messergebnisse. Die folgenden Abschnitte stellen diese Funktionen vor.

Erfassung von Gewicht und Größe
Der mBCA verfügt über eine elektronische Waage, dessen Gewichtserfassung über 4 Wägezellen erfolgt. Die Erfassung der Größe erfolgt durch manuelle Eingabe oder durch Funkübertragung von einem seca-Längenmessgerät.

Bioelektrische Impedanzanalyse
Die bioelektrische Impedanzanalyse (BIA) erfolgt nach der 8-Punkt Methode. Die Einleitung des geringen Wechselstromes und die Messung der Impedanz erfolgen pro Körperhälfte jeweils über ein Paar Fußelektroden und Handelektroden, also über insgesamt 8 Kontaktpunkte. Nähere Informationen zur BIA liegen im Anhang A1 vor.

Patientenakten-Verwaltung
Patientenakten können direkt am mBCA angelegt und in einer seca-Datenbank am PC oder alternativ auf einem USB-Speicherstick gespeichert werden. Desweiteren ist es möglich die Patienten mit Hilfe der PC-Software seca 115 zu erstellen und zum mBCA zu übertragen. Das Editieren von bereits gespeicherten Patienten ist allerdings ausschließlich mit der PC-Software seca 115 erlaubt.

Auswertung
Die Auswertung der BIA-Messungen basiert auf wissenschaftlich etablierten Formeln. Die Ergebnisse der Auswertung werden dabei als Kennzahlen dargestellt, die teilweise in Normbereiche eingestuft sind. Die Formeln zur Ermittlung der Kenngrößen, sowie die Eckwerte der Normbereiche wurden von seca aus mehreren Studien ermittelt. Nähere Informationen können dem Anhang A2 entnommen werden.

Anwenderdaten-Verwaltung
Die Zugangsdaten für den Benutzer des mBCA‘s werden in der PC-Software seca 115 angelegt und in der seca-Datenbank gespeichert. Im Rahmen der Erstellung von Benutzerkonten generiert die seca 115 automatisch eine User-PIN. Diese PIN benötigt den Anwender, wenn er vom mBCA aus auf die seca-Datenbank zugreift.

Datenübertragung und Netzwerkfunktionen
Zwischen dem mBCA, der PC-Software seca 115 und anderen Messgeräten können die Daten über bis zu drei Schnittstellen ausgetauscht werden: Netzwerkanschluss (Ethernet), Internes seca-Funkmodul (360°) und USB-Schnittstelle zum Anschluss eines USB-Speichersticks.

3.2.3 Geltungsbereich der Software

Bei den verschiedenen Softwareprodukten, die von seca für den Markt entwickelt werden, handelt es sich entweder um ein Medizinprodukt (nach § 3 Nr.1 bis 3 MPG bzw. nach Artikel 1 Punkt 2 RL 93/42/EWG) oder es handelt sich um kein Medizinprodukt. Auch wenn es bei einer Software nicht um ein Medizinprodukt handelt, wird dieses bei seca nach normativen Anforderungen zur Entwicklung von Medizingerätesoftware entwickelt. Grund hierfür ist zum Einen, dass die Anforderungen an die Software im medizinischen Kontext international

variieren. Ferner dient diese Vorgehensweise der Einheitlichkeit und der Qualitätssicherung. Als positiver Nebeneffekt, muss die Abteilung so entwickelte Software später nicht nachdokumentieren, falls die Software durch inhaltliche Änderungen später doch als Medizingerät eingeschätzt wird.

3.2.4 Risikomanagement

Gemäß der Norm für die Entwicklung medizinischer Software DIN EN 62304 und der europäischen Richtlinie für Medizinprodukte 93/42/EWG, wird bei seca für die medizinische Software ein Risikomanagementprozess nach der DIN EN 14971 durchgeführt (Abbildung 3.2).

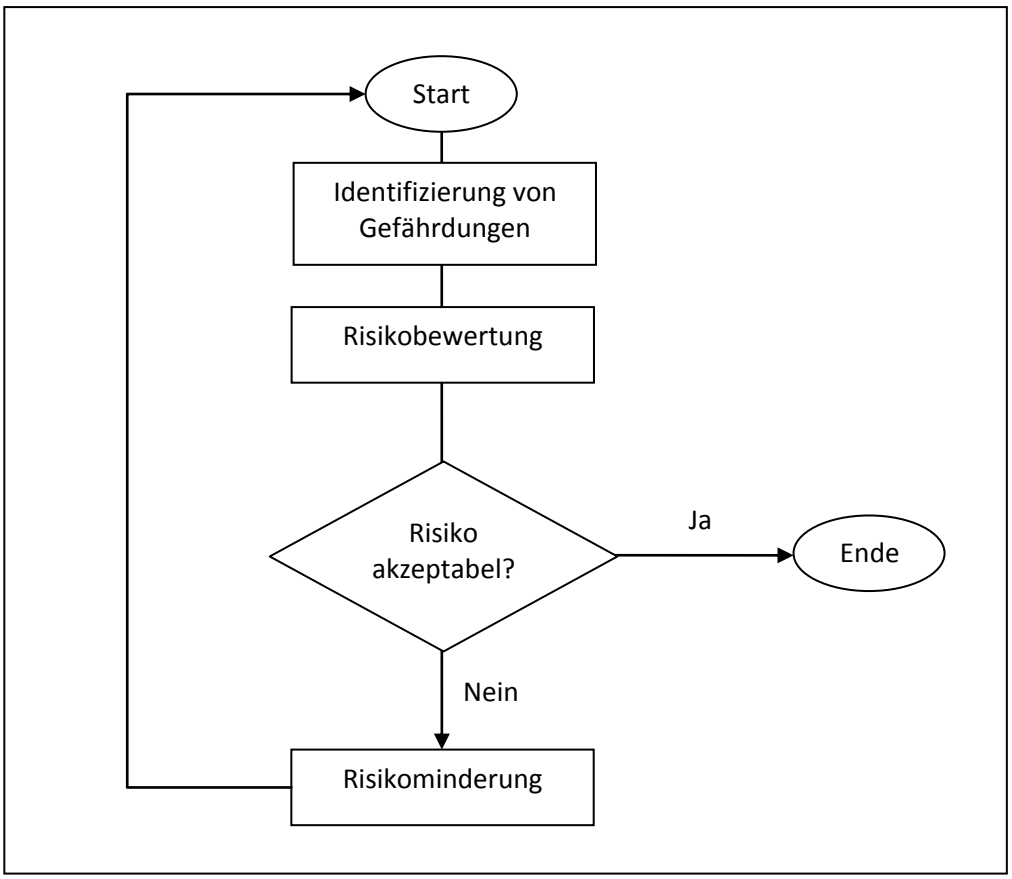


Abbildung 3.2: Risikomanagementprozess nach der DIN EN 14971

Hierzu führt seca zunächst eine *Risikoanalyse* durch. Dazu wird als erstes die Zweckbestimmung der Software analysiert und alle relevanten Gefährdungen in einem interdisziplinären Team identifiziert und kategorisiert. Die relevanten Kategorien, die bei seca eingesetzt werden, sind folgende:

Gefährdung im Zusammenhang mit der Anwendung und beitragende Faktoren

- Kommunikation zwischen Mensch und Maschine
- Gefährdungen infolge von Funktionsfehlern, falsche Wartung und Alterung, sowie beitragende Faktoren
- Gefährdungen infolge von Softwareproblemen

Nachdem die Gefährdungen bestimmt wurden, werden alle vernünftigerweise vorhersehbaren Quellen, die zu diesen Gefährdungen führen könnten, ermittelt.

Die Gefährdungen werden mit Hilfe einer *Risikobewertung* untersucht. Dazu werden bei seca hauptsächlich vier Kriterien verwendet, die verschiedene Abstufungen aufweisen:

- Potentielle Folgen: Medizinische Folgen auf den Patienten. „gering“ bis „katastrophal“.
- Exponierung: Die räumliche und zeitliche Überschneidung des Risikonehmers mit der Gefährdung. „niedrig“ oder „hoch“.
- Wahrnehmbarkeit: Die Wahrnehmbarkeit der Gefährdung für den Risikonehmer. „gut“ oder „schlecht“.
- Eintrittswahrscheinlichkeit: Das Eintreten einer Gefährdung in einer gewissen Zeitspanne. „unwahrscheinlich“ bis „hoch“.

Das Grundrisiko der Gefährdung wird anschließend aus der Kombination aller Kriterien mit Hilfe eines vorher definierten Risikographen ermittelt (Tabelle 3.1) und in einer entsprechenden Tabelle dokumentiert

Potentielle Folge	**Exponierung**	**Wahrnehmbarkeit**	**Eintrittswahrscheinlichkeit**				
			unwahrscheinlich	**selten**	**klein**	**möglich**	**hoch**
katastrophal			**6**	**7**	**8**	**9**	**10**
schwer	**hoch**	**schlecht**	**5**	**6**	**7**	**8**	**9**
		gut	**4**	**5**	**6**	**7**	**8**
	niedrig	**schlecht**	**3**	**4**	**5**	**6**	**7**
		gut	**2**	**3**	**4**	**5**	**6**
mäßig	**hoch**	**schlecht**	**1**	**2**	**3**	**4**	**5**
		gut		**1**	**2**	**3**	**4**
	niedrig	**schlecht**			**1**	**2**	**3**
		gut				**1**	**2**
gering							**1**

Tabelle 3.1: Risikographen mit den Kriterien Potentielle Folge, Exponierung, Wahrnehmbarkeit und Eintrittswahrscheinlichkeit.

Alle Risiken, die im grünen Bereich liegen, sind akzeptabel. Der gelbe Bereich, der sog. ALARP-Bereich (**A**s **L**ow **A**s **R**easonably **P**racticable), kennzeichnet Risiken, die ein Maß an Sicherheit bieten, der noch vernünftigerweise praktikabel ist.

Die Risiken im roten Bereich sind nicht akzeptabel und somit müssen *Risikominderungsmaßnahmen* angewendet werden, um diese zu reduzieren. Dazu werden die Möglichkeiten zur Risikominderung analysiert und schließlich die ausgewählten Maßnahmen umgesetzt. Das Restrisiko, nach der Durchführung der Risikominderungsmaßnahmen, wird erneut mit Hilfe des Risikographen bewertet.

Nachdem alle Maßnahmen zur Risikominimierung definiert und bewertet wurden, wird in einem interdisziplinären Gremium, zusammengesetzt aus fachlichen Experten (QM, Projektleitung, Software, PM, ggf. Arzt), entschieden, ob auch das Gesamtrestrisiko, welches insgesamt von dem Medizinprodukt verursacht wird, akzeptabel ist.

3.2.5 Anforderungsmanagement

Die Anforderungen an die Software werden bei seca aus verschiedenen Dokumenten zusammentragen, die in einem übergeordneten Prozess entstehen. Die folgenden Dokumente können als Ausgangspunkt für eine Softwareentwicklung dienen:

- **Lastenheft für die Software:** Ein Lastenheft für die Software wird von dem Auftraggeber (z.B. dem Produkt Management) geschrieben und wird vorzugsweise dann verwendet wenn der Auftraggeber gezielt eine ein neues Softwareprodukt in den Markt einführen möchte.
- **Pflichtenheft für das Produkt in dem die Software eingesetzt wird:** Auf Basis des Lastenheftes von dem Auftraggeber wird ein Pflichtenheft für das Produkt geschrieben in dem die Software eingesetzt werden soll. Es wird in der Regel verwendet wenn die Software nicht einzeln beauftragt wird, sondern Bestandteil eines Produktes ist.
- **Änderungsmitteilung:** Bei bestehenden Produkten werden kleinere Änderungen, wie z.B. Change Requirements und Bugfixes nicht über ein spezielles Lastenheft spezifiziert, sondern sind Bestandteil einer Änderungsmitteilung, die bei seca in dem Tool Bugzilla festgehalten wird.
- **Risikokontrollmaßnahmen:** Maßnahmen zur Risikominderung.

Aus allen diesen Anforderungen wird schließlich ein Softwarepflichtenheft erstellt. An dieser Stelle soll angemerkt werden, dass seca in dem Pflichtenheft nur die Anforderungen an die zu entwickelnde Software pflegt. Alle Beschreibungen, die den internen Aufbau der Software betreffen, werden gesondert dokumentiert werden (z.B. Softwarearchitektur). Für diese Entscheidung dienen die beiden folgenden Fragen:

1. Kann der Entwickler diesen Punkt in der Software ohne Rücksprache mit anderen Abteilungen ändern?

2. Kann auf eine Überprüfung dieser Anforderung während der Systemprüfung verzichtet werden?

Können beide Fragen mit „ja“ beantwortet werden, sollte die Anforderung nicht im Pflichtenheft, sondern in einem anderen Dokument aufgeschrieben werden.

Soweit angemessen und anwendbar, müssen die Anforderungen gemäß einer Checkliste aus der EN 62304, Punkt 5.2.2, in das Softwarepflichtenheft aufgenommen werden (Anhang A3). Die Anforderungen müssen dabei so formuliert werden, dass sie eindeutig sind, sich nicht gegenseitig widersprechen und sich durch eine Prüfung verifizieren lassen. Außerdem müssen sie mit einer eindeutigen Nummer versehen werden und auf Anforderungen aus dem Lastenheft, dem Pflichtenheft des Gesamtgerätes, der Änderungsmitteilung, sowie der Risikokontrollmaßnahmen und anderer Quellen (z.B. weitere Ideen des Entwicklers) zurückverfolgt werden können.

Nach dem die Softwareanforderungen zusammengetragen wurden, muss überprüft werden ob die im Kapitel 3.2.4 durchgeführte Risikoanalyse die Risiken der Software immer noch ausreichend mit berücksichtigt. Gegebenenfalls muss diese angepasst werden.

Mögliche Ursachen für eine Gefährdung/Risiko bedingt durch die Software sind:

- Fehlerhafte oder unvollständige Spezifikation der Funktionalität,
- Software Defekte in der Funktionalität,
- Ausfall oder unerwartete Ergebnisse einer SOUP,
- Hardware Ausfälle oder andere Software Defekte, die zu einer unvorhersehbaren Software Funktionsweise führen können und
- vernünftigerweise vorhersehbarer Missbrauch.

3.2.6 Entwicklungsmodell

Das Team Softwareentwicklung (SE), der Abteilung F&E, verfügt über ein definiertes Software-Entwicklungsmodell, welches an die DIN EN 62304, welche bereits im Kapitel 2.1.2 vorgestellt wurde, angelehnt ist.

Die Entwicklung der Software erfolgt in Meilensteinen, die in einer vorgegebenen Reihenfolge durchlaufen werden, wobei ein Rücksprung zum früheren Meilenstein erlaubt ist. Es ist ebenso erlaubt, die Meilensteine nur "teilweise zu erreichen". Also "Testspezifikation für Feature A", "Architektur für Feature A", "Implementierung von Feature A", "Test von Feature A". Das kann für Feature B dann wiederholt werden. Ein Meilenstein ist jedoch erst dann erreicht, wenn alles implementiert wurde.

Zum Ende jedes Meilensteines wird auf einem entsprechenden Formular bestätigt, dass alle notwendigen Aktivitäten durchgeführt wurden.

Meilenstein 1 des Modells, „Initialsierung“, definiert die Voraussetzungen, die erfüllt werden müssen, damit die Entwicklung der Software begonnen werden darf.
Es müssen alle Anforderungen an die Software, sowie ein Qualitätsmanagementsystem, welches bei seca in „übergeordneten Prozessen“ auf der Basis der DIN EN 13485 und DIN EN 9001 aufgestellt wurde, vorliegen.

Außerdem wird der Risikomanagementprozess gemäß DIN EN 14971 durchgeführt. Darauf aufbauend muss der zu entwickelnden Software vorab eine Software-Sicherheitsklasse gemäß DIN EN 62304 zugeordnet werden. Die potentiellen, medizinischen Folgen bestimmen die Sicherheitsklasse in welche die Software eingeordnet wird (Tabelle 3.2).

Software-Sicherheitsklasse	potentiellen, medizinischen Folgen
Klasse A	Keine Verletzung oder Schädigung der Gesundheit möglich
Klasse B	Keine schwere Verletzung ist möglich
Klasse C	Tod oder schwere Verletzung ist möglich
Klasse 0 *	nicht-medizinische Software

Tabelle 3.2: Dargestellt sind die Sicherheitsklassen A bis C. Die Klasse 0* ist seca-spezifisch und kommt nicht aus der DIN EN 62304.

An dieser Stelle soll darauf hingewiesen werden, dass alle entwickelten Softwareprodukte von seca bislang nicht höher als in Sicherheitsklasse A eingestuft wurden, wonach laut der DIN EN 62304 Validierung und Verifizierung der Software nicht zwingend erforderlich ist. Trotzdem beschränkt sich seca nicht drauf und erweitert die Vorgaben durch QS-Maßnahmen, da die Entwickler nach eigenen Aussagen überzeugt davon sind, dass die Markteinführung ungetesteter Software ein untragbares Risiko für das Unternehmen darstellt.

Meilenstein 2, „Analyse“, beinhaltet die Punkte von 5.1.1 bis 5.2.6 der DIN EN 62304, die für eine Software der Sicherheitsklasse A erfüllt werden müssen. Hier werden die Anforderungen an die Software analysiert und auf der Basis dessen ein Softwarepflichtenheft erstellt, freigegeben und mit Hilfe von dem Versionsverwaltungstool Subversion verwaltet werden.

Das Ausarbeiten von Verfahren für eine Integrationsprüfung einzelner Software-Komponenten ist laut der DIN EN 62304 für die Software der Sicherheitsklasse A nicht zwingend notwendig. seca setzt trotzdem auf eine Prüfung der Integration mit Hilfe des Hudson-Servers. Sicherheitsklasse A fordert zudem nicht, dass eine Systemtestspezifikation erstellt wird. Das Testen ist jedoch einer der wichtigsten Bestandteile des Entwicklungsprozesses, weshalb eine Prüfspezifikation für die Systemprüfung erstellt, freigegeben und ebenfalls in Subversion eingecheckt wird.

Meilenstein 3, „Design“, beinhaltet zwar die Punkte von 5.3.1 bis 5.4.1 der DIN EN 62304, diese müssen allerdings nur für die Sicherheitsklassen B und C berücksichtigt werden. Dennoch wird an dieser Stelle eine Spezifikation für die Softwarearchitektur erstellt und freigegeben.

Meilenstein 4, „Realisierung“, enthält die Punkte 5.5.1 bis 5.6.2 der DIN EN 62304. Ähnlich dem Meilenstein „Design“, müssen außer der Implementierung alle Softwareeinheiten keine weiteren Punkte für eine Software der Sicherheitsklasse A erfüllt werden. Trotzdem werden bei seca die Softwareeinheiten mit Hilfe von statischer Codeanalyse (StyleCop und FxCop) verifiziert. Auch Modultests und Code-Reviews werden, nach dem Ermessen des Entwicklungsteams, allerdings nicht regelmäßig, durchgeführt. Die SOUP-Komponenten werden ebenfalls, sporadisch, geprüft.

Auch wenn die Integration mit Hilfe des Hudson-Servers kontrolliert wird, fehlen sowohl ein festgelegtes Integrationskonzept (z.B. Big Bang oder Buttom-Up), als auch die entsprechende Testspezifikation für die Integrationsprüfung.

Der Meilenstein 5, „Validierung“, beinhaltet die restlichen Punkte 5.6.3 bis 5.8.8 der DIN EN 62304. Diese müssen allerdings wieder nur von der Software der Sicherheitsklasse B und C zwingend erfüllt werden. Trotzdessen erfüllt seca die meisten Punkte, auch wenn die entwickelte Software bei seca die Sicherheitsklasse A hat.

So wird die im Meilenstein 2 freigegebene Testspezifikation für die Softwaresystemprüfung abgearbeitet und protokoliert. Aus den Testprotokollen geht klar hervor, wer die Tests, unter welchen Testbedingungen durchgeführt hat. Es ist außerdem ersichtlich, welche Testfälle durchgeführt wurden, welche davon erfolgreich waren und welche nicht den erwarteten Ergebnissen entsprochen haben.

Abschließend übernimmt die QA-Abteilung die Softwareprüfung und es werden in übergeordneten Prozessen sowohl ein sog. Smoketest als auch ein Abnahmetest in Kooperation mit der Fertigung durchgeführt, welche für die Produktion des Endproduktes verantwortlich ist.

Nachdem die QA ihre Freigabe erteilt hat, ist die Software freigegeben und kann somit „released“ werden.

3.2.7 Testmanagement

Eingesetzte Prüftechniken

Wie im Kapitel zuvor erwähnt, setzt das Team SE unterschiedliche QS-Maßnahmen während des Entwicklungsprozesses ein. Dabei werden die einen Maßnahmen regelmäßig, die anderen wiederrum nur sporadisch eingesetzt, was durch mehrere Umstände bedingt ist.

So werden White-Box-Tests deutlich seltener als Black Box-Tests eingesetzt, weil seca mehrere Software-Komponenten und z.T. ganze Produkte durch externe Dienstleister entwickeln lässt, wodurch der für die White-Box-Tests notwendige Quellcode nicht immer verfügbar ist. Hierbei fehlt für die Erstellung eines White-Box-Tests häufig auch das Fachwissen des Entwicklers, der den Quellcode geschrieben hat. Dieses Fachwissen müsste sich an dieser Stelle, verbunden mit einem Zeitaufwand, angeeignet werden. Dies ist jedoch nicht notwendigerweise effektiv.

Die Black-Box-Tests werden dagegen verstärkt eingesetzt, weil die dafür geeignete Spezifikation in Form eines Pflichtenheftes vorliegt. Da die festgelegten Anforderungen schon am Anfang des Entwicklungsprozesses ermittelt werden, können somit Testfälle, die auf der Basis Black-Box-Tests erstellt werden, früh im Entwicklungsprozess ausgearbeitet werden (Meilenstein 2).

So werden die Systemtests, die überwiegend auf der Basis der Black-Box Technik entstehen, verstärkt eingesetzt. Die Modultests, die meistens auf der Basis White-Box Technik definieret werden, finden dagegen nur sporadisch statt.

Integrationskonzepte und spezielle Integrationstests, welche vor allem den Datenaustausch zwischen den jeweiligen Modulen über die entsprechenden Schnittstellen prüft, werden nicht eingesetzt. Da die einzelnen Module meistens durch externen Entwickler integriert werden, ist

es schwierig ein Konzept von seca's Seite vorzugeben. Für einen Integrationstests müssten entsprechende Testfälle ausgearbeitet werden, die aber ein Wissen über die Module voraussetzen.

Systemtest

Der Systemtest ist momentan die einzige QS-Maßnahme, welche laut dem Software-Entwicklungsprozess von seca, im Meilenstein 5, „Validierung", vorgeschrieben ist und deshalb konsequent eingesetzt wird.

Der Test wird von den Mitarbeitern des Teams SE, mit Zuhilfenahme von Mitarbeitern aus anderen Abteilungen (z.B. seca-Service) manuell ausgeführt. Dabei übernehmen die Mitarbeiter der anderen Abteilungen meistens weniger kritische funktionale Tests, während das Team SE aufgrund der fachlichen Expertise die kritischen funktionalen Tests durchführt.

Die fehlgeschlagenen Tests werden in dem Bugtrackingtool Bugzilla, nach dem hierfür definierten Prozess, dokumentiert und anschließend bewertet. Aus dieser Bewertung geht hervor, welche Fehler noch vor der kommenden Freigabe der Software behoben werden und welche auf spätere Releases zu verschieben sind.

Der Systemtest ist abgeschlossen, sobald die Testspezifikation ganz durchgearbeitet wurde. Wenn die Testspezifikation nicht ganz durchgearbeitet werden kann, entscheidet die Projektleitung, ob die vorhandene Testabdeckung ausreichend ist.

Am Ende des Systemtests erstellt der Tester Testprotokole, die folgenden Informationen enthalten müssen:

- Version der geprüften Software.
- Die Prüfergebnisse (OK, nicht OK) und gegebenenfalls Abweichungen von den erwarteten Ergebnissen.
- Aufzeichnungen über den Prüfaufbau, um die Prüfung wiederholen zu können (Verwendetet Hardware, Versionsstände, verwendete Werkzeuge, etc.).
- Name des Prüfers.

Entwicklungsbegleitendes Testen

Grundsätzlich wird bei der Integration jeder Softwareänderung, egal ob das Change Requirement oder Fehlerbehebung, entwicklungsbegleitend geprüft, ob diese Änderung nach der Implementierung weiterhin korrekt funktioniert.

Während über die Umsetzung eines Änderungswunsches von dem jeweiligen Projektmanager entschieden wird, muss bei einem Problem, welches während der Softwareentwicklung oder später im Betrieb entdeckt wurde, von dem Team SE entschieden werden, ob es sich bei dem aufgenommenen Problem tatsächlich um ein Softwareproblem handelt.

Bei der Entscheidung ob das aufgenommene Problem tatsächlich als ein Softwareproblem eingestuft werden kann, sollen folgende Kriterien berücksichtigt werden:

- Ist die Software für den Fehler verantwortlich?
- Kann der Fehler durch Softwareänderungen behoben werden?
- Besteht eine Gefährdung für Patienten oder Anwender?
- Werden zugesicherte Eigenschaften (Anforderungen) der Software nicht erfüllt?
- Hat die Änderung einen Mehrwert für den Benutzer? (Dann wird die Rückmeldung evtl. nicht als Problem, sondern als Änderungswunsch gewertet.)

Falls festgestellt werden sollte, dass es sich tatsächlich um ein Softwareproblem handelt, muss von dem jeweiligen Projektleiter darüber entschieden werden, ob und wann dieses Problem behoben werden soll.

Aus der Summe der Änderungen in einen bestimmten Zeitrahmen resultiert ein neuer „Release Candidat“, der mit Hilfe eines Systemtests überprüft wird und schließlich zu 2-3 Releases pro Jahr führt.

Intuitives Testen

Neben den formalen Tests, die dokumentiert werden müssen, prüfen die Tester die Software auch ohne eine spezifizierte Prüfvorschrift. Das geschieht häufig nicht gezielt, sondern „beiläufig“. Es werden dabei unter anderem einige nicht-funktionale Anforderungen, wie beispielweise die Antwortzeit und Benutzbarkeit, sowie andere Auffälligkeiten und Schwachstellen des Systems überprüft.

Fehlerverwaltung

Bugzilla gehört neben den Testprotokollen oder Reviewberichten, zu den konstruktiven Mitteln der Qualitätssicherung. Diese gewähren eine Rückverfolgbarkeit des Entwicklungsprozesses, die besonders bei der Entwicklung einer medizinischen Software von einer zentralen Bedeutung ist.

Das Bugtrackingtool Bugzilla wurde bei seca vor ca. 3 Jahren eingeführt. Der Fehlererfassungs- und Korrekturprozess wird allerdings erst seit ca. einem Jahr wirklich „gelebt“. Seit ca. einem halben Jahr werden alle Probleme und Änderungswünsche, die von externen Dienstleistern bearbeitet werden, ebenfalls bei seca in Bugzilla gepflegt.

Jeder Fehlerbericht oder Änderungswunsch wird in Bugzilla als ein Eintrag erfasst. Bevor dieser eingestellt wird, überprüft der Eintrag-Verfasser, ob ein gleicher oder ähnlicher Eintrag bereits existiert, was Redundanzen verhindern soll.

Beim Einstellen des Bugs müssen nun eine Reihe von Informationen eingetragen werden, die für eine möglichst eindeutige Zuordnung des Bugs zu seiner Umgebung notwendig sind. Beispielsweise sagt *Severity* etwas über den Schweregrad des Fehlers aus. Typischerweise ist der Schweregrad von der Auswirkung abhängig, d.h. je schlimmer die Auswirkung desto größer der Schweregrad. Folgende Schweregrade werden bei seca verwendet:

- Error major: Ein schwerwiegender Fehler der im nächsten Release gefixt werden muss.
- Error moderate: Ein „normaler“ Fehler, der im nächsten Release gefixt werden sollte.
- Error minor: Ein kleinerer Fehler, der gefixt wird sobald Zeit dafür ist.
- Prevention: Diese Maßnahme verhindert, dass ein Fehler auftritt.
- Change Requirement: Dieser Bericht beinhaltet keinen Fehler sondern eine Änderungsanforderung.
- Idea: Dieser Bericht beinhaltet keinen Fehler sondern eine Idee, wie das Produkt verbessert werden kann.

Nun soll der Lebenszyklus eines Bugs erläutert werden. Dieser folgt einem Pfad, der durch den „state“ (Status) und „resolution“ (Lösung, Resolution) charakterisiert wird (Abbildung 3.3).

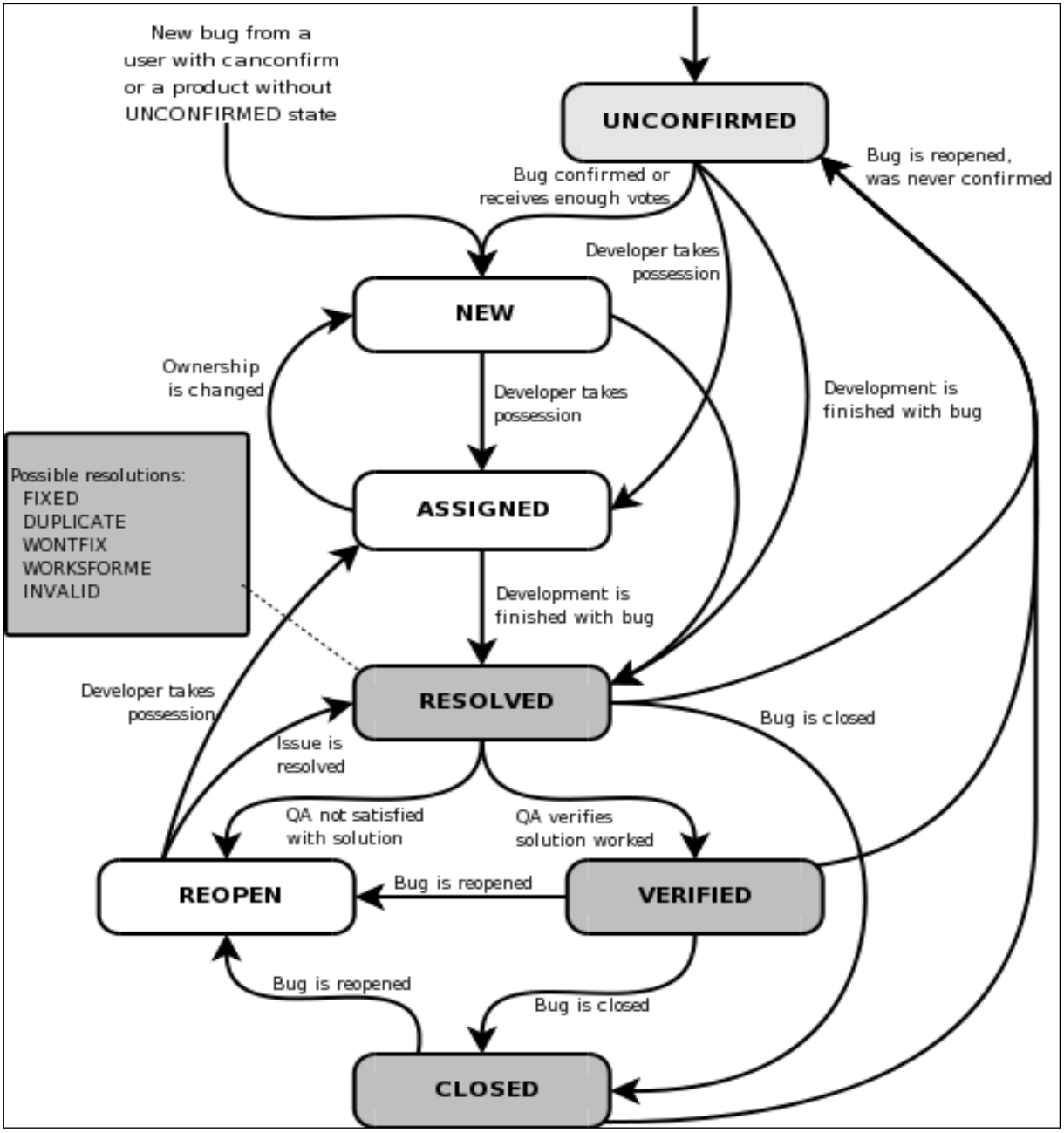


Abbildung 3.3: Lebenszyklus eines Bugzilla-Eintrags [BUGZ 12]

1. Unabhängig von dem vorausgewählten Schweregrad kriegt jeder Bug am Anfang den Status „NEW“. Die Resolution bleibt dabei leer.

2. Nimmt ein Entwickler oder eine andere Person, den Bug an, ändert sich der Status zu „Assigned“. Die Resolution bleibt weiterhin leer.

3. Stellt nun der Bearbeiter bei einem angeblichen Problem fest, dass das Fehlverhalten nicht reproduzierbar ist, setzt er den Status auf „RESOLVED“ und die Resolution auf „WORKSFORME“.

Ist der Fehler bekannt und bereits in Bugzilla eingetragen, ändert sich der Status zu „RESOLVED“ und die Resolution zu „DUPLICATE“. In diesem Fall muss der Bearbeiter die ID des Bug-Duplikats mit eingeben.

Handelt es sich bei dem angeblichen Problem um keinen Fehler, sondern um ein korrektes Verhalten, setzt der Bearbeiter den Status auf „RESOLVED“ und die Resolution auf „INVALID“.

Wenn es sich bei dem Bug tatsächlich um ein Fehlverhalten handelt, dieser aber (aus unterschiedlichsten Gründen) nicht behoben werden kann oder soll, ändert sich der Status zu „RESOLVED“ und die Resolution zu „WONTFIX“. In diesem Fall sollte eine fundierte Begründung in dem Kommentarfeld notiert werden.

Wenn es sich bei dem Problem tatsächlich um einen Fehler handelte bzw. wenn eine Änderungsanfrage umgesetzt wurde, ändert sich der Status auf „RESOLVED“ und die Resolution auf „FIXED“. Der Bug wird im Anschluss an einen Tester zugewiesen und geprüft.

4. Wurde der Bug mit dem Status „RESOLVED“ und Resolution „FIXED“ positiv getestet, ändert sich der Status des Bugs zu „CLOSED“. Die Resolution bleibt weiterhin auf „FIXED“.

Wurde der Bug mit dem Status „RESOLVED“ und Resolution „FIXED“ negativ getestet, ändert sich der Status des Bugs zu „REOPNED“, die Resolution ist dann leer. Der Bug könnte nun erneut dem Entwickler zu gewiesen werden und würde dem gleichen Ablauf wie ein Bug mit dem Status „NEW“ folgen.

Ein Bug mit Status „RESOLVED“ und den Resolutionen „DUPLICATE“, „INVALID“ oder „WONTFIX“ kann auf den Status „CLOSED“ gesetzt werden. Falls der Eintragsteller damit einverstanden ist.

Ein Bug mit Status „RESOLVED“ und der Resolution „WORKSFORME“, sollte von dem Eintragsteller evtl. im Zweifel näher beschrieben werden. Falls die neuen Informationen über den Bug dafür sorgen, dass er reproduziert werden kann, soll der Status auf „REOPENED“ gesetzt werden. Die Resolution ist bleibt dann leer. Falls es keine neuen Informationen bzgl. des Bugs gibt, wird der Status auf „CLOSED“ gesetzt.

Testspezifikation

Alle Testfälle für den Systemtest werden bei seca auf der Basis der Anforderungen des Pflichtenheftes, also als funktionale Black Box-Tests, manuell ermittelt und sollen für eine möglichst vollständige Anforderungsüberdeckung sorgen. Weil die Testfälle ebenfalls manuell ausgeführt werden, kann es vorkommen, dass bestimmte Softwareanforderungen durch die Testspezifikation nicht abgedeckt werden, so dass keine vollständige Anforderungsüberdeckung gegeben ist.

Neben den zugehörigen Anforderungen (Requirement), enthält jeder Testfall eine feste ID, eine Testbeschreibung (Test Description) und ein eindeutiges erwartetes Ergebnis (Expected Result), was in der Testspezifikation zu dokumentieren ist (Tabelle 3.3). Die Testbeschreibung enthält außer der Beschreibung auch Vorbedingungen und Eingabedaten.

ID	**Test Description**	**Expected Result**	**Requirement**
….	**….**	**….**	**….**

Tabelle 3.3: Struktur der Testbeschreibung

Außerdem wird jeder Testfall-ID die zugehörige Anforderungs-ID zugeordnet und in einer Tracebility Matrix dargestellt. Das sorgt für eine Rückverfolgbarkeit, was bei medizinischen Produkten von großer Bedeutung ist und laut der DIN EN 62304 für die Software der Sicherheitsklassen B und C sogar gefordert wird.

Da das Pflichtenheft funktionale Anforderungen an die Software beinhaltet, werden durch die Testspezifikation nur funktionale Anforderungen abgedeckt. Nicht-funktionale Anforderungen, wie z.B. Benutzbarkeit der Software, sind in dem Pflichtenheft nicht definiert und werden durch intuitives Testen geprüft.

Die Testspezifikation ist in einer bestimmten Reihenfolge verfasst, die einerseits durch die Anwendung der Software (anwendungsorientiert), andererseits durch die zuvor durchgeführte Risikoanalyse (risikoorientiert) bedingt ist. So wird die eingesetzte Software zu Beginn von einem System-Administrator in einer erwünschten Umgebung aufgesetzt und entsprechend den Vorgaben eingerichtet, was weniger kritisch im Bezug auf den Patienten ist. Wenn der Arzt anschließend die Software zur Unterstützung der Diagnostik und Therapie einsetzt, steigt das Risiko für den Patienten, der ab diesem Zeitpunkt involviert ist.

Demnach werden die weniger kritischen Testfälle am Anfang und die höher kritischen Testfälle zum Schluss abgearbeitet. Eine direkte Priorisierung der Testfälle wird in der Testspezifikation allerdings nicht vorgegeben. Denn bei dem Systemtest sollen nach Möglichkeit alle Testfälle ausgeführt werden.

Wenn gewisse Testfälle, z.B. aus einem Mangel an Zeit oder der bei begrenztem Risiko einer Änderung, nicht ausgeführt werden können, ist es wichtig, dass gerade die kritischen Testfälle, für die allerdings meistens viel Zeit benötigt wird, ausgeführt werden.

Eine weitere, wichtige Eigenschaft der Testspezifikation ist, dass die dort definierten Testbeschreibungen aus einer Reihe von bestimmten Benutzeraktionen bestehen, und demnach als GUI-Tests ausgeführt werden.

3.2.8 Zusammenfassung

Das Entwicklungsmodell bei seca ist durch exakte Meilensteine definiert, die sequentiell abgearbeitet werden. Trotz der Abfolge ist ein Rücksprung zur vorherigen Phase, z.B. zur Verfeinerung der Testspezifikation, möglich. In diesem Zustand entspricht es dem sog. erweiterten Wasserfallmodell. Wenn man die eingesetzten Testaktivitäten in das Modell mit einbezieht, kann das W-Modell, welches eine Erweiterung des V-Modells von Boehm (1982) darstellt, den Entwicklungsprozess bei seca besser beschreiben (Abbildung 3.4).

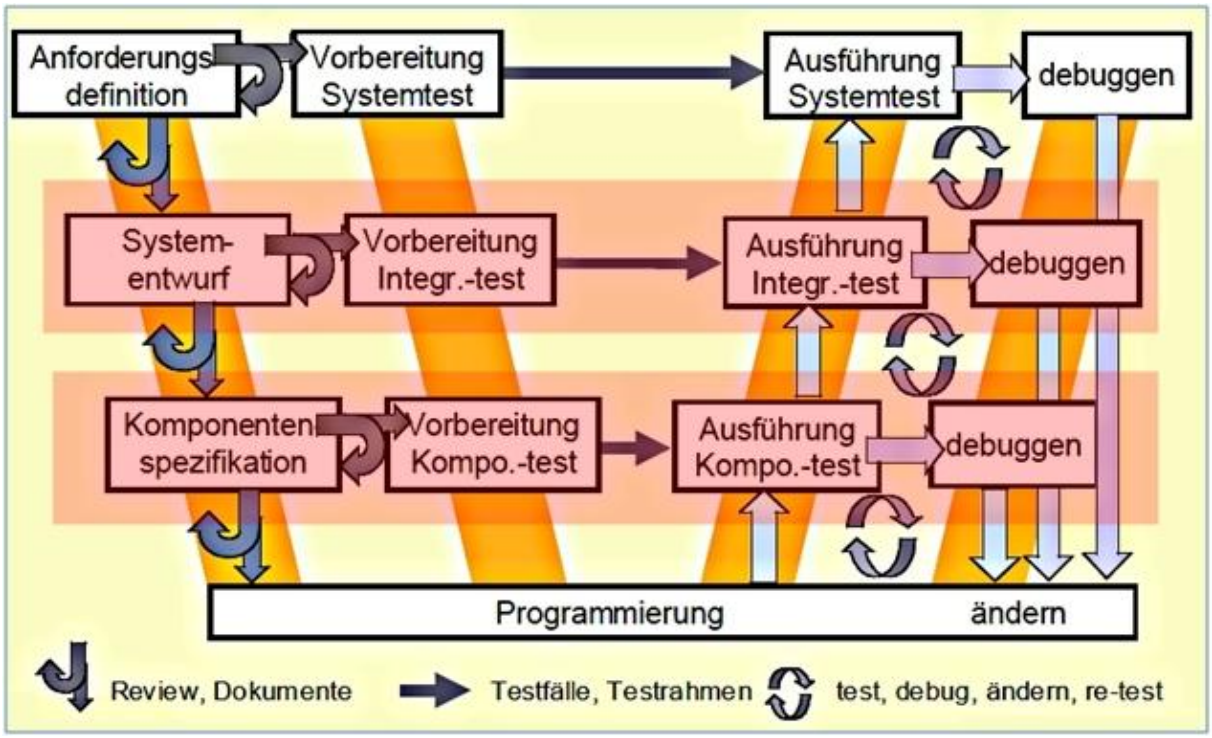


Abbildung 3.4: Das W-Modell der Softwareentwicklung. Die Markierungen kennzeichnen Aktivitäten, die in einem regulären W-Modell zwar beschrieben sind, bei seca allerdings nicht durchgeführt werden.

Im Unterschied zum regulären W-Modell werden bei seca für die Modul- und Integrationstests Testfälle nicht systematisch abgeleitet, was konsequentes Testen auf diesen Ebenen unmöglich macht. Da der Sinn und Zweck dieser Tests das frühzeitige Erkennen von Fehlern ist, folgt daraus, dass mögliche Fehler, die während der Designphase (Meilenstein 3) und der Implementierungsphase (Meilenstein 4) in die Software eingebracht wurden, einen größeren „Fehlerdruck“ auf den Systemtest mit sich bringen. Das könnte sogar dazu führen, dass gewisse Fehler erst nach dem Release erkannt werden.

Ein anderer kritischer Punkt beim derzeitigen Testvorgehen ist die nicht vollständige Anforderungsüberdeckung des Systemtests. Da die Tests manuell ausgeführt werden, ist es aus zeitlichen und wirtschaftlichen Gründen bei dem Systemtest nicht möglich alle Anforderungen abzudecken. Wenn gerade die nicht getesteten Anforderungen von dem Entwickler falsch oder nicht vollständig umgesetzt werden, sinkt die Effektivität des Systemtests.

Das derzeitige Verhalten beim entwicklungsbegleitenden Testen nach Softwareänderungen, die an einer bestehenden Software im Rahmen von CR’s und Fehlerbehebungsmaßnahmen durchgeführt werden, ist ebenfalls als kritisch zu betrachten. Obwohl bei jeder Modifikation getestet wird, ob die integrierte Änderung anforderungsgemäß umgesetzt wurde, wird nicht explizit geprüft, ob die bestehenden Softwareeinheiten ebenfalls korrekt funktionieren. Erst wenn ein Release bevorsteht, also nach einer Vielzahl von bereits integrierten Änderungen, wird ein Systemtest durchgeführt (Meilenstein 5), der Fehler, die bei der Integration einer Änderung in das bestehende System eingebracht wurden, aufdecken könnte. Derartige Fehler können allerdings häufiger nur unter einem größeren Zeitaufwand behoben werden, da festgestellt werden, muss nach welcher Änderung dieser Fehler in die Software eingebracht bzw. in der Software „demaskiert“ wurde. Kurz vorm Release, wo meistens ein größerer Zeitdruck

herrscht, könnten solche Fehler eine Verschiebung des Releases nach sich ziehen. Es wird deutlich, dass an dieser Stelle ein entwicklungsbegleitender Regressionstest fehlt.

Unter solchen Umständen wird es besonders heikel werden, wenn eine Software-Änderung nach dem Systemtest, also während der „heißen Phase“ des Projektes, durchgeführt werden muss und somit ein erneuter, ausführlicher Test aus Zeitgründen nicht mehr möglich wird. Dadurch steigt nämlich die Gefahr, dass durch die integrierte Änderung ein Fehler in die bestehende Software eingebracht wurde und dieser erst im Betrieb entdeckt werden kann.

3.3 Fehleranalyse

Die Bestandsanalyse im vorherigen Kapitel hat mögliche Schwachstellen der Softwareentwicklung und speziell der Qualitätssicherung aufgedeckt. Eine systematische Analyse bisher registrierter Softwarefehler dient als Basis für eine weiterführende Analyse des aktuellen Zustandes der Entwicklung und der Qualitätssicherung.

Zu diesem Zweck wurde als methodischer Ansatz die Fehlerstrommessung gewählt und auf die drei Softwareprodukte seca 101, seca 105/115 und seca 515/514 angewandt. Gerade diese Metrik erlaubt es relevante Eigenschaften zu den Softwarefehlern systematisch und effizient zu erfassen und anschließend zu analysieren. Ferner ermöglicht es die abschließende Auswertung.

Die Voraussetzung für die FSM ist ein in der Praxis angewandter Entwicklungsprozess, welcher die Entwicklung und Fehlererfassung abdeckt. Wie die Bestandsanalyse gezeigt hat, wird diese Voraussetzung von seca erfüllt.

3.3.1 Datenerhebung und -aufbereitung

Die für die FSM relevanten Daten konnten dem Fehlerverwaltungstool Bugzilla entnommen werden. Hierzu wurden Einträge in die Kategorien "Fehler" und "keine Fehler" unterteilt. Die folgende Tabelle 3.4 gibt einen Überblick über die Kriterien, die für die Kategorisierung ausschlaggebend waren.

	Severity	**State**	**Resolution**
Fehler	Error Minor, Error Moderate, Error Major	CLOSED, RESOLVED, VERIFIED	FIXED
keine Fehler	Idea, Prevention, Change Requirement	NEW, ASSIGNED	DUPLICATE, INVALID, WORKSWORME

Tabelle 3.4: Die Suchkriterien für Bugzilla.

An dieser Stelle ist zu beachten, dass es sich bei den Einträgen im Status „New“ und „Assigned“ selbstverständlich auch um Fehler handeln könnte, es allerdings zu dem Zeitpunkt

der Datenerhebung unklar war. Deshalb werden diese Einträge nicht für die FSM berücksichtigt. Hieraus ergab sich die folgende Fehlerverteilung:

Produkt	seca 115		seca 515/514		seca 101	
Fehler (absolut, relativ)	84	48%	23	27%	138	70%
keine Fehler (absolut, relativ)	92	52%	63	73%	58	30%
Alle Einträge	176	100%	86	100%	196	100%

Tabelle 3.5: Die absolute & relative Fehlerverteilung nach den Kategorien "Fehler" und "kein Fehler".

Die folgende Tabelle 3.6 legt dar, wie viele Veröffentlichungen (Releases) seit Beginn der Fehlererfassung (erfassten) und insgesamt für das jeweilige Produkt stattgefunden haben:

Produkt	seca 115	seca 515/514	seca 101
Anzahl der „erfassten“ Releases	2	1	3
Anzahl der „nicht erfassten“ Releases	3	1	0
Anzahl bisheriger Releases	5	2	3

Tabelle 3.6: Die erfassten, nicht erfasste und bisherige Releases für die drei seca Produkte.

Die Softwarefehler für alle drei Produkte wurden nun als CSV-Datei aus Bugzilla exportiert und mit Hilfe von Excel in eine Tabelle eingeführt.

Kritische Betrachtung von restlichen Bugzilla-Einträge

Nach der Analyse der Fehler, wurden auch andere Einträge, die nicht als Softwarefehler klassifiziert wurden analysiert. Es ist aufgefallen, dass sich hinter den einzelnen Change Requirements (CR) auch Anforderungsfehler versteckt haben. Dabei wurden Anforderungen, die wahrscheinlich fehlerhaft waren, durch neue, korrekte Anforderungen ersetzt, ohne dass die fehlerhaften Anforderungen in der Fehlerstatistik aufgetaucht sind. Es muss an dieser Stelle aber ausdrücklich gesagt werden, dass es sich um Einzelfälle (<5%) handelt und die CR’s nicht systematisch zu diesem Zweck genutzt wurden.

Auf der anderen Seite wurde im Gegensatz hierzu auch beobachtet, dass Fehler nicht in Bugzilla dokumentiert wurden, weil diese nach der Entdeckung zeitnah behoben werden.

Die gerade beschriebenen Effekte sind nur schwer messbar. Es wird angenommen, dass sich beide Effekte nahezu aufheben und somit die Ergebnisse der Fehleranalyse in ihren Tendenzen nicht maßgeblich verfälscht werden.

3.3.2 Identifizierung der Fehlerquellen und -senken

Es wurden drei Fehlerquellen bestimmt: Anforderungen, Implementierung und Andere. Die Fehlerquelle *Anforderungen* steht dabei stellvertretend für alle Aktivitäten, die zur Erstellung von Softwareanforderungen geführt wurden, also Meilensteine 1 und 2. Die Fehlerquelle *Implementierung* enthält alle Aktivitäten des Entwicklers, die zu der Umsetzung der Anforderungen durchgeführt wurden, also Meilensteine 3 und 4. Alle Fehler, die nicht zu den Fehlerquellen Anforderungen und Implementierung zugeordnet werden konnten, zählten zu der Fehlerquelle *Andere*.

Außerdem wurden drei Fehlersenken identifiziert: Systemtest, Release, Andere. Die Fehlersenke *Systemtest* fasst alle Aktivitäten zusammen, bei denen die Fehler mit Hilfe von Testfällen der Systemtestspezifikation bestimmt wurden. Hauptsächlich waren es die Aktivitäten aus dem Meilenstein 5. Die Fehlersenke *Release* steht stellvertretend für alle Aktivitäten, bei denen die Fehler nicht während regulärer QS-Maßnahmen (Z.B. Präsentationen und klinischer Einsatz). Die Fehlersenke *Andere* fasst alle anderen QS-Maßnahmen, die während der Entwicklung eingesetzt wurden. Dazu zählen Modultests, Reviews und sog. intuitives Testen.

Die anschließende Zuverlässigkeitsanalyse zeigte bei allen Produkten eine Übereinstimmung von über 90%, was eine sehr hohe Güte der Klassifikation symbolisiert.

3.3.3 Ergebnisse und Auswertungen

Die Ergebnisse des Fehlerstrommodels werden in der Tabelle 3.7 vorgestellt. Die Rohdaten liegen im Anhang A4 vor.

Produkt	**seca 115**			**seca 515/514**			**seca 101**		
Fehlerquellen	Anf.	Impl.	Andere	Anf.	Impl.	Andere	Anf.	Impl.	Andere
	20%	77%	2%	26%	65%	9%	25%	65%	10%
Fehlersenken	Syst.	Release	Andere	Syst.	Release	Andere	Syst.	Release	Andere
	65%	18%	17%	44%	30%	26%	61%	4%	35%

Tabelle 3.7: Die Verteilung der Fehler bezogen auf Fehlerquellen und -senken.

Bei den drei untersuchten Produkten können zwei eindeutige Tendenzen beobachtet werden: Zum einen wurden die meisten Fehler, 65-77%, bei der Umsetzung von Anforderungen in die Software eingebracht. Zum anderen wurden 54-70% aller Fehler während des Systemtests detektiert.

Die letztere Tendenz ist weniger überraschend, weil bei seca überwiegend der Systemtest eingesetzt wird. Es zeigt auch, dass diese QS-Maßnahmen auch eine bestimmte Effektivität aufweist. Um allerdings eine genaue Aussage über die Effektivität treffen zu können, muss der ermittelte Wert für die Fehlersenke Systemtest im Verhältnis zu der Fehlersenke Release gesetzt werden. Das bietet die Metrik *Defect Detection Effectivness* (Kapitel 2.6.3). Die genauen Berechnungen befinden sich im Anhang A5.

Bei seca 101 lag dieser Wert bei 94%, bezogen auf alle QS-Maßnahmen, sogar bei 96%. Das zeigt, dass vor allem die QS-Maßnahme Systemtest sehr effektiv bei der Detektion von Fehlern eingesetzt werden konnte.

Bei seca 115 konnte eine schlechtere Effektivität der QS-Maßnahmen ermittelt werden. Diese lag bei 78%, bezogen auf die Fehlersenke Systemtest, und bei 78%, bezogen auf alle QS-Maßnahmen.

Die QS-Maßnahmen bei seca 515/514 waren am wenigsten erfolgreich. Die Effektivität aller QS-Maßnahmen lag nur bei 70%, und der Fehlersenke Systemtest sogar lediglich bei 59% und damit von den drei untersuchten Produkten am geringsten.

Eine genaue Untersuchung der Fehler, die nicht durch die QS-Maßnahmen abgefangen werden konnten, soll mögliche Ursachen für die schlechtere Effektivität der QS-Maßnahmen, speziell bei seca 515/514, aufzeigen.

Die Tendenz bei den Fehlerquellen ist überraschend. Es wäre zu erwarten gewesen, dass die meisten Fehler bei der Zusammenstellung der Softwareanforderungen verursacht werden und nicht, wie die Ergebnisse zeigen, bei deren Umsetzung (Kapitel 2.6.1). Das spiegelt auf der einen Seite die hohe Qualität der Anforderungen und auf der anderen Seite die Probleme, die offensichtlich bei der Implementierung dieser Anforderungen entstanden sind. Über mögliche Gründe kann an dieser Stelle lediglich diskutiert werden. So könnte es sein, dass die komplexen Anforderungen von den Entwicklern nicht vollständig verstanden wurden und deshalb nicht korrekt umgesetzt werden konnten. Begünstigt werden könnte es durch die überwiegend externe Entwicklung der Produkte. Für eine begründete Aussage müssten weitere Untersuchungen der Fehlerquellen folgen, was allerdings nicht der Untersuchungsgegenstand dieser Arbeit ist.

Korrelationen zwischen Fehlerquelle und Fehlersenke

Die folgende Tabelle 3.8 zeigt den Zusammenhang zwischen den Fehlern, welche durch die Fehlerquelle *Implementierung* eingebracht wurden und den Fehlern, die durch die Fehlersenke *Systemtest* erkannt wurden.

Produkt	**seca 115**			**seca 515/514**			**seca 101**		
Fehlersenken	Syst.	Release	Andere	Syst.	Release	Andere	Syst.	Release	Andere
Fehlerquelle *Implementierung*	72%	18%	9%	60%	40%	0%	78%	3%	16%
Fehlerquellen	Anf.	Impl.	Andere	Anf.	Impl.	Andere	Anf.	Impl.	Andere
Fehlersenke *Systemtest*	11%	85%	4%	0%	100%	0%	8%	90%	2%

Tabelle 3.8: Die Korrelation zwischen der Fehlerquelle Implementierung und der Fehlersenke Systemtest

So handelt es sich bei 85-100% aller Fehler, die während der Systemtestaktivitäten gefunden wurden um Implementierungsfehler. Das ist kein überraschendes Ergebnis, wenn man bedenkt, dass gerade die Systemtestaktivitäten überprüfen sollen ob die Anforderungen an die Software richtig umgesetzt wurden. Umgekehrt wurden die Implementierungsfehler zwar überwiegend, mit 60-78%, aber nicht ausschließlich bei den Systemtestaktivitäten entdeckt.

Die Verteilung der Implementierungsfehler bezogen auf die Fehlersenke *Release* hat sich in Abhängigkeit von Produkten recht deutlich unterschieden. Während bei seca 515/514 40% der Implementierungsfehler nicht durch die QS-Maßnahmen des Entwicklungsprozesses abgefangen werden konnten, waren es bei seca 101 lediglich 3%. Das verdeutlicht, dass vor allem bei seca 515/514 die QS-Maßnahmen weniger erfolgreich eingesetzt werden konnten.

Bei der Gegenüberstellung der Fehlerquelle *Anforderungen* und Fehlersenke *Andere* kann ebenfalls eine Korrelation beobachtet werden (Tabelle 3.9).

Produkt	**seca 115**			**seca 515/514**			**seca 101**		
Fehlersenken	Syst.	Release	Andere	Syst.	Release	Andere	Syst.	Release	Andere
Fehlerquelle *Anforderungen*	35%	18%	47%	17%	17%	67%	18%	9%	74%
Fehlerquellen	Anf.	Impl.	Andere	Anf.	Impl.	Andere	Anf.	Impl.	Andere
Fehlersenke *Andere*	57%	43%	0%	67%	0%	33%	52%	33%	15%

Tabelle 3.9: Die Korrelation zwischen der Fehlerquelle Anforderungen und der Fehlersenke Andere.

So wurden 47-74% aller Fehler der Fehlerquelle *Anforderungen* durch die QS-Maßnahmen der Fehlersenke *Andere* entdeckt, während 9-18% dieser Fehler durch die QS-Maßnahmen nicht entdeckt werden konnten. Durch den Systemtest konnten nur wenige, 17-35%, der Anforderungsfehler entdeckt werden.

Dieses Ergebnis zeigt, dass vor allem Reviews, sowie intuitives Testen, welche zu der Fehlersenke *Andere* gezählt werden, besser geeignet sind, um die Fehler, die durch fehlerhafte Anforderungen verursacht werden aufzufinden.

Zum Schluss werden die Fehler den Fehlerquellen *Implementierung* mit der Fehlersenke *Release* ins Verhältnis gesetzt (Tabelle 3.10).

Produkt	**seca 115**			**seca 515/514**			**seca 101**		
Fehlersenken	Syst.	Release	Andere	Syst.	Release	Andere	Syst.	Release	Andere
Fehlerquelle *Implementierung*	72%	18%	9%	60%	40%	0%	78%	3%	16%
Fehlerquellen	Anf.	Impl.	Andere	Anf.	Impl.	Andere	Anf.	Impl.	Andere
Fehlersenke *Release*	20%	80%	0%	14%	86%	0%	50%	50%	0%

Tabelle 3.10: Die Korrelation zwischen der Fehlerquelle Implementierung & der Fehlersenke Release.

Bei den Fehlern, die nicht durch QS-Maßnahmen entdeckt werden konnten, handelt es sich zu 50-86% um Implementierungsfehler. Bei seca 515/514 lag dieser Anteil bei 86% und war damit am größten. Es handelt sich um Fehler, die grundsätzlich mit Hilfe Systemtests hätten detektiert werden können.

Repräsentative Fehler der seca 515/514

Im Folgenden werden einige Implementierungsfehler der seca 515/514, die nicht durch die eingesetzten QS-Maßnahmen detektiert werden konnten, näher beschrieben. Die Fehler, die durch fehlerhafte Anforderungen verursacht wurden, sollten nicht explizit durch den Systemtest abgefangen werden und werden deshalb nicht näher betrachtet werden.

Zwei Mitarbeiter des Teams SE haben drei repräsentative Fehler, der Fehlersenke *Release*, ausgewählt. Diese werden nun näher vorgestellt.

1. Bug 672: „no calculation of FM for ethnic groups South/Central Am. and Others"

Bei mehreren Testmessungen mit verschiedenen Patienten wurden bei Ethnien „süd-/mittelamerikanisch" und „andere" die Werte, sowie Normbereiche für die Fettmasse nicht angezeigt.

Der Fehler wurde durch den Systemtest nicht entdeckt, weil die Systemtestspezifikation die entsprechenden Anforderungen für die Auswertung von Patienten mit der Ethnien „süd-/mittelamerikanisch" und „andere" nicht abgedeckt.

2. Bug 681: „missing measurement point in BCC graphic for ethnicity African"

Dieser Fehler betrifft das Fehlen eines Messpunktes in einem Diagramm, welches die ermittelten Kenngrößen graphisch darstellen sollte. Der Fehler wurde bei einem afrikanischen Patienten, während einer Testmessung entdeckt.

Der Fehler ereignete sich, weil die Systemtestspezifikation die entsprechenden Anforderungen für die Auswertung von Patienten mit der Ethnie „afrikanisch" nicht abgedeckt hat.

3. Bug 789: „faulty translation for Finland“

Zwei Übersetzungsfehler wurden im Finnischen ebenfalls während einer Präsentation von seca 515/514 entdeckt. Beide Fehler bezogen sich auf die falsche Übersetzung im Auswertmodul.

Der Fehler ist durch den Systemtest nicht entdeckt worden, weil die entsprechende Testspezifikation die Anforderungen „Übersetzungen in verschiedenen Sprachen“ nicht abgedeckt hat.

3.3.4 Zusammenfassung

Die Fehleranalyse hat gezeigt, dass vor allem bei der Entwicklung der seca 515/514 die Effektivität der eingesetzten QS-Maßnahmen nicht ausreichend war. Es konnten 30% der Fehler nicht durch die derzeitigen QS-Maßnahmen abgefangen werden. Der Großteil dieser Fehler, nämlich 86%, wurde durch die fehlerhafte Umsetzung der Anforderungen in die Software eingebracht.

Die geringe Effektivität der QS-Maßnahmen bei seca 515/514 könnte auf die Schwachstellen des Testens zurückzuführen sein. Zusammenfassend ermittelte die Bestandsanalyse die folgenden drei Defizite im Zusammenhang mit secas QS-Maßnahmen:

1. Ein Systemtest, der nicht vollständig die Anforderungen abdeckt.
2. Mangelnde Modul-, Integrationstests und Reviews.
3. Das Fehlen von entwicklungsbegleitenden Regressionstests.

Die beschriebene Problematik bei seca 515/514 ist vermutlich hierauf zurückzuführen. Die repräsentativen Fehler, die nicht während der Softwareentwicklung erkannt wurden, zeigen, dass bei seca 515/514 eine unvollständige Abdeckung der Anforderungen durch die Testspezifikation dafür verantwortlich ist, dass diese Fehler durch den Systemtest nicht abgefangen werden konnten.

Die gute Effektivität des Systemtests bei seca 101 hat verschiedene Ursachen. Zum einen spielt sicherlich die Testspezifikation der seca 101, die eine bessere Abdeckung der Anforderungen als die der seca 515/514 aufweist, eine wichtige Rolle. Zum anderen wurden bei seca 101 verstärkt Modultests und Reviews eingesetzt, wodurch 35% aller Fehler schon vorab abgefangen werden konnten und somit der Systemtest einem geringeren „Fehlerdruck“ ausgesetzt war.

Das Fehlen von entwicklungsbegleitenden Regressionstests, die nach einer Softwareänderung das bestehende System überprüfen sollten, könnte ebenfalls für den hohen Anteil an „durchgerutschten Fehlern“ bei seca 515/514 mitverantwortlich sein. Denn speziell bei der Entwicklung der seca 515/514 wurden neben dem Systemtest und dem intuitiven Testen nur wenige entwicklungsbegleitende QS-Maßnahmen eingesetzt. So konnte der Fehlerdruck auf den Systemtest nicht reduziert werden.

3.4 Verbesserungsvorschläge auf Basis der Bestands- und Fehleranalyse

Die nähere Betrachtung der Fehler bei seca 515/514 hat bestätigt, dass eine der Ursachen für die nicht ausreichende Effektivität der QS-Maßnahmen die unvollständige Abdeckung der Anforderungen ist. Die bislang durchgeführten Testmaßnahmen haben gerade die Anforderungen nicht abgedeckt, die von den Entwicklern fehlerhaft umgesetzt wurden. So konnten Fehler während des Systemtests nicht abgefangen werden.

Die vollständige Abdeckung der Anforderungen war aufgrund der geringen Effizienz von manueller Testausführung nicht möglich. Denn ein manueller Test, der vollständig die Anforderungen abdeckt, wäre aus der Sicht des Zeitaufwandes und der Kosten nicht vertretbar. Das gerade beschriebene Effizienzproblem verhindert außerdem den Einsatz eines entwicklungsbegleitenden Regressionstests. Dieser Test könnte vor allem Fehler, die nach einer Änderung ins System eingebracht wurden, erkennen und den Systemtest damit ebenfalls „entlasten". Demnach kann die Steigerung der Effizienz als eine Voraussetzung für die Verbesserung der Effektivität betrachtet werden. So könnten sowohl ein entwicklungsbegleitender Regressionstest, als auch ein vollständiger Systemtest umgesetzt werden.

Andere Ursache für die mangelnde Effektivität ist der sporadische Einsatz von entwicklungsbegleitenden QS-Maßnahmen auf der niedrigen Entwicklungsebene (Modultest und Integrationstest), sowie statischer Tests (z.B. Review und statischer Analyse). Diese Maßnahmen könnten helfen, Fehler früh im Entwicklungsprozess zu erkennen und damit den „Fehlerdruck" auf den Systemtest erheblich zu reduzieren.

Die Software seca 515/514 wird derzeit vorwiegend extern entwickelt, so dass quellcodeorientierte QS-Maßnahmen nur eingeschränkt eingesetzt werden können. Da seca die Anforderungen an die Software formuliert, bietet sich der Einsatz von anforderungsorientierten QS-Maßnahmen, also von Black-Box Tests, an.

Zur Steigerung der Effektivität sollte folglich ein automatisiert ablaufender, entwicklungsbegleitender Regressionstest eingesetzt werden, der die risikorelevanten Bereiche der Software vollständig abdeckt. Da eine Systemtestspezifikation bereits vorliegt, liegt es nahe, auf dessen Basis die Testbereiche für den Regressionstest zu identifizieren, zu priorisieren und auszuwählen. Die ausgewählten Testbereiche müssen die Anforderungen vollständig abdecken. Das gilt es zu überprüfen und notfalls durch weitere Testfälle zu vervollständigen. Da der Regressionstest bestimmte Bereiche der Systemtestspezifikation abdecken soll, kann er später eingesetzt werden, um diese Bereiche des Systemtests zu entlasten.

Im folgenden Kapitel wird für den risikoorientierten und automatisierten Black-Box-Regressionstest ein Testkonzept erarbeitet und schließlich bei seca 515/514 implementiert.

4 Testkonzept

Das Ziel des Kapitels ist es ein Testkonzept für einen risikoorientierten und automatisierten Black-Box-Regressionstest auszuarbeiten. Mit dessen Hilfe werden Unternehmen, die sich ähnlichen Herausforderungen bei dem Softwaretesten wie seca stellen müssen, in der Lage sein, den Testbereich zu optimieren. Das Konzept beschränkt sich dabei auf operatives Testmanagement. Strategisches Testmanagement ist nicht Untersuchungsgegenstand dieser Arbeit.

In diesem Kapitel werden die Rahmenbedingungen des Testkonzepts festgelegt. Die Implementierung des Konzeptes bei seca 515/514 im nächsten Kapitel zeigt schließlich detailliert wie das Konzept umgesetzt werden kann.

Das Konzept besteht aus zwei Teilbereichen. Im ersten Bereich wird die Toolauswahl für die Automatisierung anhand bestimmter Kriterien vor dem Testprozess durchgeführt. Der zweite Bereich beinhaltet einen permanenten Testprozess, der aus mehreren Schritten besteht. Die folgenden Abschnitte stellen die beiden Teilbereiche vor.

4.1 Testtoolauswahl

Um das passende Testtool auswählen zu können, müssen Kriterien definiert werden, anhand welcher die verfügbaren Tools bewertet werden.

Es wurden folgende Kriterien ermittelt:

- Erkennung von Oberflächenobjekten
- OCR-Erkennung / Bilderkennung
- Betriebssystem
- Skriptsprachen
- Kosten

Für die Durchführung von GUI-Tests muss ein Testtool in der Lage sein Oberfläche erkennen und bedienen zu können. Das wichtigste Kriterium für GUI-Tests ist also die Erkennung von Oberflächenelementen. Dazu muss ein Testtool in der Lage sein sowohl bei der Bedienung, als auch bei der Überprüfung der GUI die Oberflächenobjekte, sowie deren Eigenschaften zu erkennen. Falls es nicht möglich ist Oberflächenobjekte zu erkennen, soll das Testtool über eine Bild bzw. Texterkennungsfunktion den Bildinhalt identifizieren können.

Ein anderes wichtiges Kriterium ist das Betriebssystem auf dem die zu testende Software ausgeführt wird. Das System sollte möglichst von dem Testtool unterstützt werden.

Die Tests können im „Capture and Play“-Modus und auch als Skripts erstellt werden. Je mehr Skriptsprachen von dem Tool unterstützt werden, desto besser ist es für den Tester. Denn dadurch hat er mehr Auswahl bei der Erstellung der automatisierten Tests.

Der wirtschaftliche Faktor spielt ebenfalls eine wichtige Rolle. Deshalb müssen auch Kosten für das Tool mitberücksichtigt werden. Falls geplant ist, auf mehreren Rechnern parallel zu testen, soll das Testtool auch über Ausführungslizenzen verfügen.

Nachdem ein Testtool ausgewählt wurde, sollten die wichtigsten Funktionen des Tools vorgestellt werden.

4.2 Testprozess

Der Testprozess besteht aus mehreren Schritten, die während der Entwicklung permanent durchlaufen werden (Abbildung 4.1). Zuerst werden die Testbereiche für den Regressionstest identifiziert und priorisiert. Anschließend werden die am höchsten priorisierten Testbereiche mit der vorhandenen Testspezifikation auf Vollständigkeit überprüft. Falls die Testspezifikation die Anforderungen unvollständig abdeckt, muss sie entsprechend erweitert werden. Wenn die Testspezifikation vollständig ist, können Referenzdaten und anschließend der Testskript für den automatisierten Test erstellt werden.

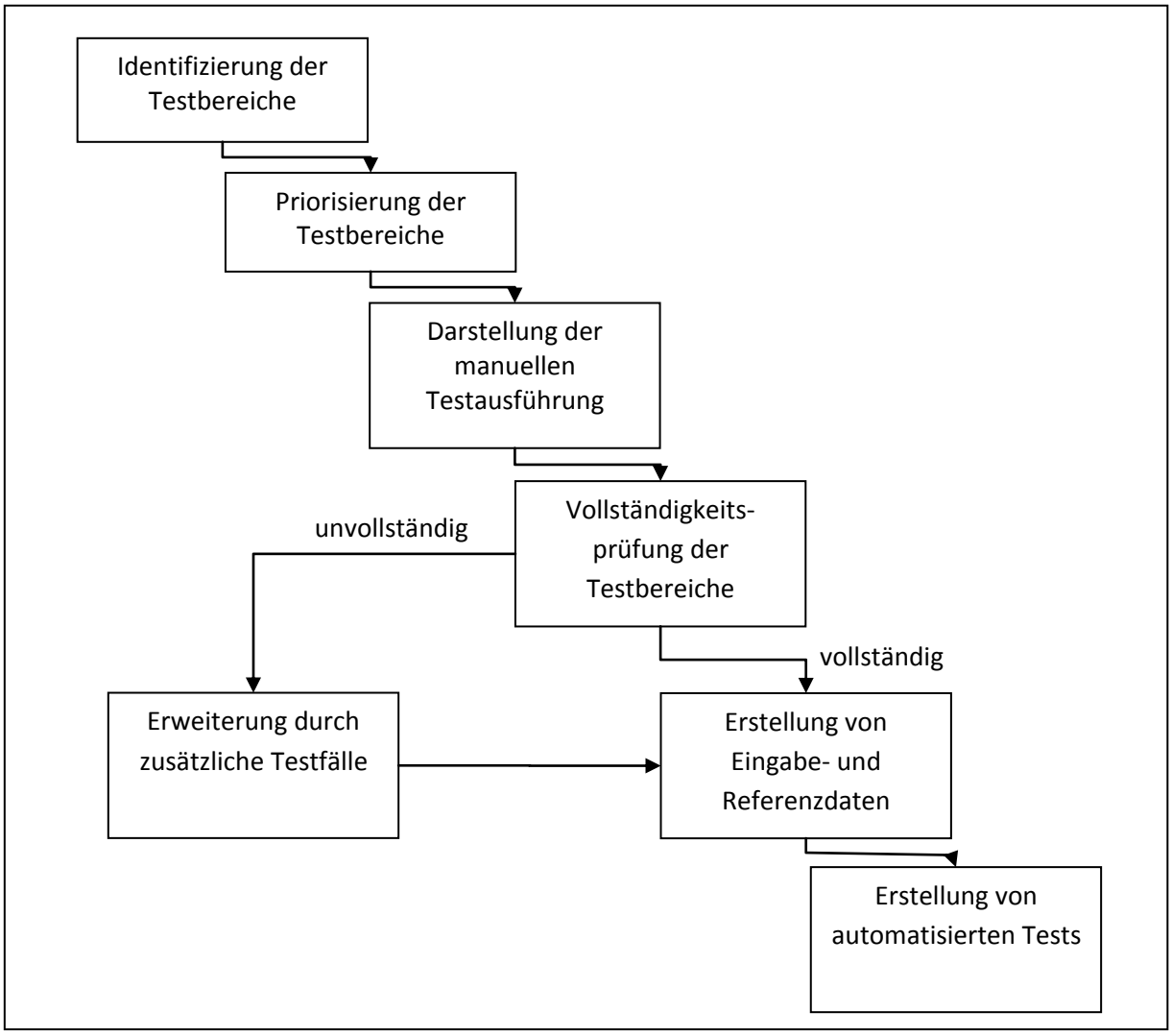


Abbildung 4.1: Testprozess, der permanent durchgeführt wird

4.2.1 Identifizierung und Priorisierung von Testbereichen

Ein Regressionstest wird nach einer Softwareänderung anhand von ausgewählten Testfällen ausgeführt. Die Testfälle definieren dabei Soll-Ergebnisse, die mit den Ist-Ergebnissen der Software zu vergleichen sind. Da Modifikationen in Form von Fehlerbehebungen oder Change Requirements in eine bestehende Software regelmäßig integriert werden, muss ein Regressionstest ebenso häufig durchgeführt werden. Der Test darf demnach, unabhängig davon ob er manuell oder automatisiert ausgeführt wird, einen gewissen Umfang nicht überschreiten. Auf der anderen Seite soll der Regressionstest die wichtigsten Bereiche der Software abdecken.

Zuerst müssen die Testbereiche der Software identifiziert werden. Es empfiehlt sich, dies in einem interdisziplinären Team von Domänenexperten anhand des Softwarepflichtenheftes oder, falls vorhanden, der Systemtestspezifikation durchzuführen.

Eine Priorisierung der Testbereiche soll dabei helfen Testkandidaten für den Regressionstest zu ermitteln und damit den Umfang des Tests zu begrenzen. Die höher priorisierten Bereiche werden in den Test integriert. Bei der Abstufung der Priorisierung wird vorgeschlagen zwischen „hoch“, „mittel“ und „niedrig“ zu unterscheiden. Eine andere Unterscheidung wäre grundsätzlich ebenfalls möglich. Die Priorisierung der Bereiche wird unter Berücksichtigung der Risiken, der Schwachstellen, sowie der Automatisierungseignung vorgenommen.

Die ermittelten Kernbereiche werden nun nach den Kriterien Risiko, Schwachstellen und Automatisierungseignung priorisiert.

Priorität nach Risiko

Auf der Basis der Risikoanalyse, die bei einer medizinischen Software vor dem Beginn der Entwicklung durchgeführt wird, werden die Risiken für die Patienten ermittelt. Falls die Risiken nicht akzeptabel sind, werden Risikokontrollmaßnahmen eingeführt, die das Risiko in den akzeptablen Bereich bringen sollen. Trotzdem bleibt meistens ein Restrisiko übrig, welches relativ hoch sein kann.

Die Risiken wurden bei der Risikoanalyse aus den Gefährdungen ermittelt, die wiederum auf bestimmte Bereiche der Software bzw. Hardware zurückzuführen sind. So können die Risiken zu den einzelnen Bereichen der Software zugeordnet werden, wofür ebenfalls Domänenexperten verantwortlich sind. Die höher bewerteten Risiken geben damit Rückschlüsse auf die kritischen Bereiche der Software. Diese Bereiche stellen ein höheres Risiko für den Patienten dar und sind somit für den Regressionstest höher zu priorisieren.

Priorität nach Schwachstellen

Da davon ausgegangen wird, dass die Software bereits getestet wurde, sollten Testbereiche, die in der Vergangenheit fehleranfällig waren, unabhängig von der Fehlerschwere, für den Regressionstest höher priorisiert werden. Denn dort liegen aus der Sicht der Fehleranfälligkeit die Schwachstellen der Software, die intensiver getestet werden sollten.

Dazu müssen die „tatsächlichen“ Fehler mit Hilfe einer Fehleranalyse ermittelt und anschließend den zuvor identifizierten Testbereichen zugeordnet werden. Die Zuordnung ist ebenfalls von Domänenexperten durchzuführen.

Priorität nach Automatisierungseignung

Die Priorisierung bei einem *automatisierten* Regressionstest wird durch die zu realisierende Automatisierung stark beeinflusst. Diese stellt an den Test spezielle Anforderungen. So ist eine vollständige Automatisierung aller Testaktivitäten weder möglich, noch sinnvoll. Denn es gibt immer Testbereiche, bei denen sich eine Automatisierung aus wirtschaftlicher Sicht nicht rentieren würde bzw. die aus technischer Sicht nicht zu realisierbar ist.

Deshalb sollte darüber entschieden werden, welche Testfälle zuerst und überhaupt automatisiert werden sollten und welche weiterhin manuell getestet werden sollten. Das hängt von verschiedenen Faktoren ab und muss letztendlich individuell entschieden werden. Folgende Kriterien helfen bei der Entscheidungsfindung:

Faktoren für eine Testautomatisierung

- Mehrfach zu wiederholende Tests, deren Eingaben sich nur minimal unterscheiden
- Tests, die über einen längeren Zeitraum laufen müssen
- Einfach strukturierte Tests
- Performancetest und Funktionstests, die überprüfen, ob die Software bei definierten Eingaben korrekte Ausgaben liefert

Faktoren für eine manuelle Testausführung

- Selten auszuführende Tests
- ReTests, die nach einer Fehlerbehebung unter Termindruck stattfinden müssen
- intuitive Tests, ohne vorhersehbare Ergebnisse
- nicht-funktionale Tests der Wartbarkeit, Testbarkeit, Benutzbarkeit

Die identifizierten Testbereiche werden unter Berücksichtigung der vorgestellten Faktoren priorisiert werden. Die höher priorisierten Bereiche werden bei der Auswahl zu automatisierender Testfälle priorisiert.

Zusammenfassend mit den beiden oberen Kriterien ergibt sich für die Priorisierung der Testbereiche die folgende Verteilung:

Testbereich	Risiko	Schwachstellen	Automatisierungseignung	Gesamtpriorität
1.	gering (1), mittel (2), hoch (3)	gering (1), mittel (2), hoch (3)	gering (1), mittel (2), hoch (3)	Von 1 bis 9 Punkte
2.	gering (1), mittel (2), hoch (3)	gering (1), mittel (2), hoch (3)	gering (1), mittel (2), hoch (3)	Von 1 bis 9 Punkte
…	gering (1), mittel (2), hoch (3)	gering (1), mittel (2), hoch (3)	gering (1), mittel (2), hoch (3)	Von 1 bis 9 Punkte

Tabelle 4.1: Priorisierung der Testbereiche nach ausgewählten Kriterien. In den Klammern sind die einzelnen Punkte für die jeweiligen Bewertungen dargestellt.

Die einzelnen Abstufungen der Priorität werden mit einem bis drei Punkten bewertet. So kann der Testbereich mit der insgesamt höchsten Priorität identifiziert werden. Die Gesamtpriorität wird dabei in Punkten als Summe der einzelnen Prioritäten angegeben.

Falls es mehrere Testbereiche mit der ähnlichen oder sogar gleichen Gesamtpriorität muss individuell entschieden werden, ob nur ein oder tatsächlich alle Bereiche in den Regressionstest übernommen werden. Im Regelfall soll ein Testbereich ausgewählt werden.

4.2.2 Darstellung der manuellen Testausführung

Bevor Testfälle aus dem ausgewählten Testbereich automatisieren werden, sollte deren manueller Testausführung genau analysiert werden. Dies dient dazu wichtige und verallgemeinerbare Schritte der jeweiligen Testfälle zu identifizieren und

Die Entscheidung über die eingesetzte Testtechnik kann ebenfalls durch die Analyse des manuellen Tests beeinflusst werden.

In diesem Konzept bedient man sich dem GUI-Testverfahren.

Bei dem Übergang von dem manuellen zum automatisierten Testdurchlauf existiert im Bezug auf die technische Realisierung der Testautomatisierung eine Reihe von kritischen Aspekten. Die Betrachtung des manuellen Testdurchlaufes dient dazu diese Punkte zu identifizieren. Wie die kritischen Stellen konkret aussehen hängt letztendlich von der zu testenden Anwendung ab.

4.2.3 Vollständigkeitsprüfung der Testbereiche

Der ausgewählte Testbereich (oder evtl. mehrere Bereiche), der am höchsten priorisiert wurde, sollte die entsprechenden Softwareanforderungen vollständig abdecken. Die Testfälle des Testbereiches können der vorhandenen Systemtestspezifikation entnommen werden.

In Fällen, wo sicher ist, dass der ausgewählte Testbereich der Testspezifikation die Anforderungen vollständig abdeckt, kann direkt mit der Umsetzung der Automatisierung begonnen werden.

Wenn die vorhandene Testspezifikation die Softwareanforderungen in dem ausgewählten Bereich nicht vollständig abdeckt, müssen sie zwingend erweitert werden.

Die Testfälle der vorhandenen Systemtestspezifikation wurden auf der Basis von Softwareanforderungen, als Black-Box Tests, erstellt. Die Black-Box Technik soll auch bei der Generierung von neuen Testfällen eingesetzt werden. Dazu können unterschiedliche Methoden der Black-Box Tests, die bereits im Kapitel 2.2.3 vorgestellt wurden, verwendet werden. Welche Methode für die Testfallgenerierung gewählt wird, hängt am Ende von der Anforderung ab. Bei Eingabedaten

Nach dem die vorhandenen Testfälle vervollständigt wurden, liegt nun ein Testbereich vor, der die ausgewählten Anforderungen vollständig abdeckt. Die Testfälle können nun automatisiert werden. Bevor die Ausgabedaten der Software mit den Referenzdaten (Soll-Ergebnissen) verglichen werden können, müssen die Referenzdaten erst einmal erstellt werden.

4.2.4 Erstellung von Eingabe- und Referenzdaten

Die Soll-Ergebnisse werden auf der Basis von Testfällen definiert und sind von den Eingabedaten abhängig. In Fällen, wo die Ergebnisse über Formeln berechnet werden müssen, kann es sehr zeitaufwändig und fehleranfällig sein, die Soll-Ergebnisse zu ermitteln. Deshalb bietet sich hierfür der Einsatz von Tools an, die dabei helfen die Berechnungen durchzuführen. Z.B. können Excel-Makros verwendet werden. Die Referenzdaten können anschließend in Tabellen abgelegt werden.

Wenn es sich bei den Soll-Ergebnissen um Graphiken oder Zeichnungen handelt, müssen die entsprechenden Bildaufnahmen erstellt und abgelegt werden.

Kritische Aspekte

Eine Herausforderung bei dem Testen einer medizinischen Software, die mit Messdaten arbeitet, sind die Eingabedaten und Referenzdaten (Soll-Ergebnisse) für den Testfall. Bei einem automatisierten Regressionstest sollten die Tests reproduzierbar und wiederholbar sein. D.h. die Ausgabedaten (Ist- Ergebnisse) der Software sollten jederzeit gegen die zuvor definierten Referenzdaten (Soll-Ergebnisse) geprüft werden können. Das ist aber nur dann möglich, wenn die Eingabedaten für einen Testfall gleich bleiben. Wenn es sich bei den Eingabetestdaten zumindest teilweise um Messdaten handelt, die zuerst von einer Mess-Hardware ermittelt werden müssen, kann es durch die Messtoleranzen der aufnehmenden Sensoren der Messtechnik zu nicht deterministischen Ergebnissen führen. Das führt zu unterschiedlichen Ausgabewerten für denselben Testfall. Somit müssten die Referenzdaten nach jeder Messung neu ermittelt werden, was bei einem Regressionstest ungünstig ist. Die Referenzdaten sollten schließlich gleich bleiben. Die Lösung für dieses Problem kann z.B. der Einsatz von Messnormalen sein, die standardisierte Ergebnisse liefern.

4.2.5 Erstellung von automatisierten Tests

Wenn sich die Ausgangssituation für einen Test unterscheidet, kann es dazu führen, dass die zu testende Software sich unterschiedlich verhält und die Ergebnisse des Tests unbrauchbar sind. Daher soll am Anfang des automatisierten Tests ein definierter Initialzustand hergestellt werden.

Nachdem alle relevanten Rahmenbedingungen, wie z.B. Programmeinstellungen, Datenbankzugriffe, für den Testfall erfüllt und ein Ausgangszustand festgelegt wurde, kann mit der Erstellung des Tests begonnen werden.

Testgrundgerüst

Die meisten Tools bieten einen Test Recorder an mit dem ein Grundgerüst des Tests aufgezeichnet werden kann. Es werden dabei die Oberflächenaktionen durchgeführt und aufgenommen, die auch bei der manuellen Bedienung der zu testenden Anwendung erfolgen.

Im nächsten Schritt muss der aufgenommene Test meistens in Abhängigkeit von den kritischen Stellen erweitert und überarbeitet werden, bevor es tatsächlich ausgeführt werden kann.

Anpassung des aufgezeichneten Tests

Das Grundgerüst, welches im vorherigen Schritt mit dem Test Recorder aufgezeichnet wird, soll alle Grundaktionen beinhalten. Allerdings müssen meistens zusätzlich Schritte, wie z.B. Fallunterscheidungen und Verzögerungen in den aufgezeichneten Testablauf eingebunden werden.

Ein anderer wichtiger Punkt ist zu beachten, wenn mehrere, ähnliche Testfälle ausgeführt werden, bei den sich zwar die Eingabedaten unterscheiden, die Eingabestruktur allerdings nahezu unverändert bleibt. Über sog. datengetriebenes Testen kann hier eine wesentliche Erleichterung bei der Eingabe von Daten erzielt werden. Dazu werden die Eingabedaten soweit wie möglich parametrisiert und die Datensätze in Tabellen abgelegt. Sie können dann dynamisch in den Test eingebunden werden.

Überprüfung mittels Checkpoints

Die Checkpoints sind bestimmte Stellen im automatisierten Test, bei denen eine Überprüfung der zu testenden Anwendung stattfindet. Sie können vom Tester bei der Testaufzeichnung oder im Nachhinein angelegt werden. Am Ende des automatisierten Überprüfung kann der Tester anhand der Checkpoints, die in einem Testlog angezeigt werden, feststellen ob der Test erfolgreich gelaufen ist oder nicht. Falls eine Prüfung an einem Checkpoint fehlerhaft war, wird es, z.B. durch eine Error-Message, entsprechend gekennzeichnet.

So können die Checkpoints einzelne oder mehrere Eigenschaften von Objekten überprüfen. Welche Eigenschaften konkret geprüft werden, hängt dabei wesentlich vom untersuchten Objekt und dem erwarteten Testergebnis ab. So kann z.B. untersucht werden, ob ein Eingabefeld oder ob ein Radio-Button zu einem bestimmten Zeitpunkt aktiv ist.

Soll bei dem Checkpoint eine Graphik untersucht werden, wird diese pixelweise überprüft und mit der zuvor gespeicherten „Soll-Graphik“ verglichen. Der Tester sollte festlegen, nach wie vielen Abweichungen bei dem Pixelvergleich ein Fehler ausgelöst wird.

Ein Checkpoint kann außerdem der Textinhalt in der zu testenden Anwendung sein. Das setzt allerdings voraus, dass entweder über die Eigenschaften der Objekte auf den Textinhalt zugegriffen werden kann oder, dass das Tool über eine Texterkennung verfügt, mit deren Hilfe der Text ausgelesen werden kann. An dieser Stelle soll darauf aufmerksam gemacht werden, dass die Textinhalte nicht nur als Checkpoints verwendet werden können. Sie können z.B. bei Fallunterscheidungen im Testdurchlauf eingesetzt werden, um eine differenzierte Weiterverarbeitung zu gewährleisten.

Kritische Aspekte

Tooleinsatz

Eine Voraussetzung dafür, dass das Tool auf die Oberflächeninhalte zugreifen kann, ist, dass sowohl das Tool als auch die zu testende Software auf demselben Betriebssystem, also auch demselben Rechner, ausgeführt werden. Bei Inkompatibilitäten zwischen den Anforderungen des zu testenden Systems und des Automatisierungstools ist es jedoch nicht möglich, beide auf einem System laufen zu lassen.

Als Lösung kann die zu testende Anwendung von einem lokalen PC, auf dem das Testtool ausgeführt werden kann, angesteuert werden. Der Remote-Zugriff kann z.B. mittels Virtual Network Computing (VNC) erfolgen und erlaubt es die Bildschirminhalte des Rechners, auf dem die Anwendung ausgeführt wird (Server), auf einem lokalen PC (Client) abbilden zu lassen. Die Benutzeraktionen (z.B. Tastatureingaben, Mausbewegungen und Mausklicks) werden im Gegenzug von dem lokalen PC zum Embedded Rechner übertragen. VNC arbeitet nach dem Client-Server Prinzip. Während sich das Server-Programm auf dem Rechner mit der zu testenden Software befindet, wird das Client-Programm auf dem lokalen PC ausgeführt. Beide Rechner müssen sich dazu einem gemeinsamen Netzwerk befinden. Das Netzwerkprotokoll Remote Framebuffer Protocol (RFB) dient dabei zur Übertragung von Bildschirminhalten und Benutzeraktionen. Der Nachteil hier wäre allerdings die fehlende Erkennung der Oberflächenobjekte.

Oberflächenbedienung

Während bei dem manuellen Test der Tester die Oberflächenelemente erkennt und anschließend bedient, muss bei einem automatisierten Testablauf das Tool diese Funktionen übernehmen. Wenn das Tool nicht in der Lage ist Oberflächenobjekte zu erkennen, muss bei der Bedienung mit festen Koordinaten gearbeitet werden, was allerdings nur dann sinnvoll ist, wenn sich die Objektverteilung und Objektfunktionen an der Oberfläche möglichst wenig verändern.

Fallunterscheidungen

Ein weiteres Problem entsteht durch Abfragen, die sich während eines Testdurchlaufs dynamisch verändern können. Dies können z.B. Sicherheitsabfragen oder Meldungen eines med. Alarmsystems sein, die durch bestimmte Aktionen des Testers bestätigt werden müssen, damit der Test fortgesetzt werden kann. Hierzu können if-then Anweisungen eingesetzt werden.

Synchronisation und Timing

Die meisten Aktionen des Testdurchlaufs können hintereinander ohne eine längere Verzögerung ausgeführt werden. Lediglich die Reaktionszeit des Systems, muss bei der Testausführung beachtet werden. Zwischen einigen Aktionen entstehen allerdings größere Verzögerungen, die bei dem Durchlauf berücksichtigt werden müssen. Diese Stellen sollten bei dem manuellen Test vorab identifiziert werden.

Während bei dem manuellen Test der Tester die Verzögerungen abwartet, müssen bei einem automatisierten Durchlauf die Verzögerungen („Delays“) entweder manuell vorgegeben oder durch das Testtool detektiert werden. Bei der letzten Variante muss das Tool in der Lage sein nach der Verzögerung die Veränderungen gegenüber dem vorherigen Zustand zu erfassen. Das könnte sich z.B. durch veränderte Eigenschaften bestehender der Oberflächenobjekte äußern. Falls keine Objekte erkannt werden können, müssen die Verzögerungen an den vorher beschriebenen Stellen manuell vorgegeben werden. Da die Reaktionszeit des Systems schwanken kann, sollen die Verzögerungen in diesem Fall etwas großzügiger gewählt werden.

4.3 Fazit

Mit diesem Testkonzept ist es möglich medizinische Software entwicklungsbegleitend zu testen. Dabei werden hochpriorisierte Systembereiche mit Hilfe des Regressionstests getestet. Diese Bereiche werden zuvor identifiziert und nach den Kriterien Risiko, Schwachstellen und Automatisierungseignung priorisiert. Der Testbereich mit der höchsten Priorität wird anschließend auf seine Vollständigkeit überprüft. Wenn der Testbereich die entsprechenden Anforderungen nicht vollständig abdeckt, ist er durch weitere Testfälle, die auf Basis der Black-Box Technik erstellt werden, zu ergänzen. Die vollständige Abdeckung der Anforderungen soll eine hohe Effektivität des Regressionstests bei der Fehlererkennung gewährleisten.

Anschließend wird die Automatisierung bei den Testfällen des ausgewählten Testbereiches umgesetzt. Die Automatisierung soll die mangelnde Effizienz, die bei einer manuellen Testausführung vorliegt, erhöhen.

5 Implementierung bei seca 515/514

Das im Rahmen dieser Arbeit erarbeitete Testkonzept wird am Beispiel der seca 515/514 Embedded-Software, die bereits im Kapitel 3.2.2 vorgestellt wurde, implementiert.

Dazu wird am Anfang ein Testautomatisierungstool ausgewählt und ein Testprozess bestehend aus folgenden 6 Schritten eingesetzt: (1) Identifizierung und (2) Priorisierung der Testbereiche, (3) Darstellung der manuellen Testausführung, (4) Vollständigkeitsprüfung der Testbereiche, (5) Erstellung von Eingabe- und Referenzdaten, sowie (6) Erstellung von automatisierten Tests.

Die folgenden Abschnitte stellen die einzelnen Schritte des Konzeptes vor.

5.1 Testtoolauswahl

Es existiert eine Reihe von Tools, mit denen automatisierte GUI-Tests durchgeführt werden können. Nach einer Marktuntersuchung kamen die Automatisierungs-Tools „Test Complete“, „Froglogic Squish“, sowie „HP QuickTest Professional“ in die engere Auswahl. Diese wurden nach den im Kapitel 4.1 aufgestellten Auswahlkriterien bewertet.

	Test Complete	**Squish**	**QuickTest**
Erkennung von Qt-Objekten	Ja	Ja	Nein
Erkennung von Windows-Objekten	Ja	Ja, aber in einer separaten Version	Ja
Skriptsprachen	Visual Basic Script, Java Script, Delphi Script, C++ Script, C# Script	Java Script, Tcl, Python	Visual Basic Script
Betriebssysteme	Windows, Eingebettete Systeme (Windows Embedded Standard)	Windows, Linux, Unix, Mac OS, Eingebettete Systeme	Windows
OCR-Erkennung	Ja	Nein	Nein
Bilderkennung	Ja	Ja	Nein

Kosten / Lizenz 1.Hauptlizenz 2.Ausführungslizenz	1. Basic-Version: 1000€; Advanced-Version: 2000€ 2. 400€ bzw. wird bei der „Advanced-Version“ mit der Hauptlizenz geliefert	1. 2400€ 2. 450€	1. 6000€ 2. Nicht vorhanden

5.1:Die Auswahlkriterien angewandt auf die Testautomatisierungstools.

Die Auswahlkriterien konnten am besten von dem Test Complete erfüllt werden. Test Complete unterstützt sowohl die Erkennung von Windows- als auch von Qt-Oberflächenobjekten. Es kann auf allen Windows- und einigen Eingebetteten-Betriebssystemen (Windows Embedded Standard) ausgeführt werden. Die Testskripte können in verschiedenen Programmiersprachen geschrieben werden. Außerdem wird die OCR-Texterkennung von Test Complete, dem einzigen der bewerteten Tools, angeboten. Auch der relativ niedrige Preis von 2000€ für die „Advanced-Version“ und die zusätzlich mitgelieferte Ausführungslizenz, waren dafür mitentscheidend, dass für die Automatisierung der Regressionstests die Wahl auf Test Complete gefallen ist. Im Anhang A6 liegt eine Beschreibung von Test Complete, die auf die wichtigsten Funktionalitäten des Tools näher eingegangen wird.

5.2 Identifizierung und Priorisierung von Testbereichen

Ein vollständiger Test, der alle Bereiche der Software mit allen Eingabewerten und Vorbedingungen überprüft, ist aus verschiedenen Gründen nicht durchführbar. Deshalb muss unter der Berücksichtigung ausgewählter Kriterien festgelegt werden, welche Bereiche der Software eine höhere Priorität für den Test haben.

Die folgenden Kapitel bestimmen zuerst die zu testenden Bereiche der Software anhand der vorhandenen Systemtestspezifikation bestimmt werden. Desweiteren werden anhand der Kriterien Risiko, Schwachstellen, und Automatisierungseignung Prioritäten der einzelnen Bereiche ermittelt. Der Bereich mit der höchsten Priorität ist ein Kandidat für den automatisierten Regressionstest.

5.2.1 Testbereiche

Die Testfälle aus der Systemtestspezifikation für seca 515/514 wurden auf der Basis der Softwareanforderungen, als Black-Box-Tests entworfen. Diese können in drei Kernbereiche unterteilt werden (Tabelle 5.2):

Testbereiche	Beschreibung
Testbereich 1	Testfälle, welche die BIA-Messung, sowie die Ergebnisse der Auswertung auf ihre Richtigkeit überprüfen.
Testbereich 2	Testfälle, welche administrative Funktionalitäten, wie Kommunikation, User Management, Installation, Update und Systemeinstellungen auf ihre Korrektheit überprüfen.
Testbereich 3	Alle anderen Testfälle, welche die Korrektheit von Systemmeldungen, Graphiken, Labels etc. überprüfen.

Tabelle 5.2: Dargestellt sind die Testbereiche und die Beschreibungen, die identifiziert wurden.

Nun werden die ermittelten Testbereiche anhand ausgewählter Kriterien priorisiert.

5.2.2 Priorität nach Risiko

Basierend auf der zugehörigen Risikoanalyse hat die Software seca 515/514 die Sicherheitsklasse A. Die einzige Gefährdung, die zu einem nicht akzeptablen Risiko führt, ist der "begründet vorhersehbarer Missbrauch". Demnach liegt für den Patienten ein Risiko vor, wenn der Arzt eine falsche Therapieentscheidung (z. B. Medikamentenvergabe) trifft, ohne sich eine weitere Bestätigung der Diagnose zu holen. Als Folge der Therapieentscheidung könnte es bei dem Patienten zur Beeinträchtigung des Gesundheitszustandes kommen. Eine falsche Therapieentscheidung des Arztes wird besonders dann begünstigt, wenn die angezeigten Kenngrößen und Normbereiche, z.B. aufgrund nicht korrekt umgesetzter Formeln, fehlerhaft sind. Gegen dieses Risiko wurde eine Risikokontrollmaßnahme eingeführt. Es handelt sich dabei um einen Hinweis in der Bedienungsanleitung und der Zweckbestimmung, der besagt:

„*Die Software seca 515/514 ist keine Diagnosesoftware. Zur Erstellung einer genauen Diagnose müssen neben den Ergebnissen der seca 515/514 gezielte Untersuchungen durch den Arzt veranlasst und deren Ergebnisse berücksichtigt werden*“.

Trotz dieser Risikokontrollmaßnahme liegt für die Gefährdung ein Restrisiko der Stufe 6 vor. Es handelt sich um ein hohes Risiko, welches an den nicht-akzeptablen Bereich grenzt. Alle anderen Gefährdungen wurden mit einem geringen Risiko versehen, so dass das Gesamtrisiko von seca akzeptiert wird.

Nach der Risikoanalyse bleibt festzuhalten, dass falsch angezeigte Kenngröße und Normbereiche das Eintreten der Gefährdung „begründet vorhersehbarer Missbrauch“ begünstigen können. Genau aus diesem Grund werden die Testfälle, welche die Ergebnisse der Auswertung überprüfen sollen, bezogen auf das Risiko höher priorisiert. Die anderen beiden Testbereiche haben laut der Risikoanalyse nur geringe potentielle Folgen für den Patienten und werden deshalb mit geringer Priorität eingestuft.

5.2.3 Priorität nach Schwachstellen

Die Testbereiche, die in der Vergangenheit besonders fehleranfällig waren, werden unabhängig von der Fehlerschwere höher priorisiert. Denn dort liegen aus der Sicht der Fehleranfälligkeit die Schwachstellen der Software, die intensiver getestet werden sollten.

Die bei der FSM von seca 515/514 ermittelten Fehler wurden, soweit es möglich war, zu einem der drei Testbereiche zugeordnet (Anhang A4). Demnach stammen die meisten Fehler, jeweils 43% und 35%, aus den Testbereichen 2 und 3. Diese Testbereiche sind deshalb höher zu priorisieren. Die Fehler, die aufgrund von falschen Ergebnissen in der Auswertung zustande gekommen sind, waren mit 22% seltener und sind somit niedriger zu priorisieren.

5.2.4 Priorität nach Automatisierungseignung

Unter der Berücksichtigung der im Kapitel 4.2.1 vorgestellten Faktoren wurde festgestellt, dass die Testfälle aus dem Testbereich 1 am ehesten für eine Automatisierung in Frage kommen. Die BIA-Messungen müssen mehrmals in dem gleichen Testablauf ausgeführt werden, wobei sich nur wenige Eingabedaten verändern. Die anschließende Überprüfung der Auswertung läuft ebenfalls nach der gleichen Struktur ab. Die Ausgaben der Software, die sich in Abhängigkeit von den Eingaben unterschieden, werden dabei mit den zuvor definierten Ergebnissen verglichen. Sowohl der Eingabeprozess, als auch der Vergleichsprozess von Softwareausgaben und den Soll-Ergebnissen nehmen bei manueller Durchführung viel Zeit in Anspruch.

Die Tests aus dem Bereich 3 kommen ebenfalls für die Automatisierung in Frage. Bei den Tests werden bestimmte Ausgabedaten mit den zuvor definierten Soll-Ergebnissen verglichen, was ebenfalls zeitaufwändig ist. Allerdings liegen nur wenige gleich strukturierte Testabläufe vor. Der größte Teil der Tests wird dabei nicht mehrfach, sondern einzeln ausgeführt und ist somit aufwändiger zu automatisieren, als die Testfälle aus dem Bereich 1.

Die Tests aus dem Bereich 2 sind am wenigsten für die Automatisierung geeignet. Bei der Überprüfung geht es weniger um Eingabedaten, die bestimmte Ausgabedaten hervorrufen, als um das Prüfen administrativer Anforderungen. Dabei werden die Tests, wie z.B. Update bzw. Installation, meist nur einmalig innerhalb eines Systemtests ausgeführt und nehmen relativ wenig Zeit in Anspruch.

Zusammenfassend mit den beiden oberen Kriterien ergibt sich für die drei Priorisierungen die folgende Verteilung:

Testbereich	Risiko	Schwachstellen	Automatisierungseignung	Gesamtpriorität
1.	hoch	mittel	hoch	8
2.	gering	hoch	gering	5
3.	gering	hoch	mittel	6

Tabelle 5.3: Priorisierung der Testbereiche bei seca 515/514.

Die Ergebnisse der durchgeführten Testfallpriorisierung zeigen, dass die Black-Box-Tests, die zur Prüfung von BIA-Messung und der Auswertung eingesetzt werden, am höchsten priorisiert werden.

Der folgende Abschnitt überprüft in welchem Maße die Testfälle die zugehörigen Anforderungen abdecken.

5.3 Darstellung der manuellen Testausführung

Die bisher manuell durchgeführte Überprüfung der Auswertungen (Testbereich 1) besteht aus folgenden Abschnitten:

1. Initialzustand herstellen
2. Gewicht und Größe ermitteln
3. Bioelektrische Impedanzanalyse durchfuhren
4. Messungen einem Patienten zuordnen
5. Messergebnisse überprüfen
6. Soll-Ergebnisse ermitteln

1. Initialzustand herstellen

Um im weiteren Verlauf die BIA-Messung einem Patienten zuordnen zu können, muss sichergestellt sein, dass der mBCA auf eine seca-Datenbank zugreifen kann. Dazu gibt es zwei Möglichkeiten:

1. Die vorkonfigurierte seca-Datenbank befindet sich auf einem Server und kann vom mBCA aus über Funk oder Ethernet angesteuert werden.

2. Die vorkonfigurierte seca-Datenbank befindet sich auf einem USB-Speicherstick, der an die USB-Schnittstelle des mBCA's angeschlossen ist.

Für die Testdurchführung, wird die erste Variante eingesetzt, da sie auch im Betrieb am häufigsten verwendet wird. Die Funkübertragung ist weniger robuster und auch langsamer, als die Ethernet-Verbindung, so dass der PC (Server) mit der seca-Datenbank und der mBCA über Ethernet verbunden werden. Die Kommunikation zwischen dem PC und dem mBCA wird dabei durch den CLS-Server, der zuvor auf dem PC mit der seca-Datenbank zu installieren ist, festgelegt.

In der seca-Datenbank muss mindestens ein User vorliegen. Beim ersten Anmelden des mBCA's an die Datenbank wird der Anwender von dem System aufgefordert sich mit einer User-PIN zu authentifizieren. Dabei erscheint ein Dialogfenster in dem über einen Ziffernblock die User-PIN eingetragen und mit dem Klick auf die Enter-Taste bestätigt werden kann. Am rechten Rand der GUI leuchtet nun das Login-Symbol auf. Der mBCA ist nun an der seca-Datenbank angemeldet und es kann mit dem eigentlichen Testdurchlauf begonnen werden.

In der seca-Datenbank können mit Hilfe der PC Software seca 115 bereits vor der Testdurchführung Patienten mit ihren patientenspezifischen Parametern angelegt werden. Diese Patienten werden dann direkt der BIA-Messung zugeordnet und müssen nicht erst am mBCA erstellt werden. Das spart Zeit und unterbricht nicht den Testfluss.

2. Gewicht und Größe messen

Zu Beginn der Messung werden Gewicht, Größe und BMI (Body Mass Index) eingetragen bzw. ermittelt. Das Gewicht wird von dem mBCA ermittelt, indem sich eine Testperson auf

die Wiegefläche des eingeschalteten Gerätes stellt und abwartet bis der Gewichtswert in dem Feld „Weight“ angezeigt und in das Feld „Hold“ übernommen wird (Abbildung 5.1). Anschließend wird die Größe (in cm) in dem Feld „Height“ manuell über Tastatureingabe eingetragen. Der BMI wird von der Software automatisch ermittelt und erscheint in dem Feld „BMI“.

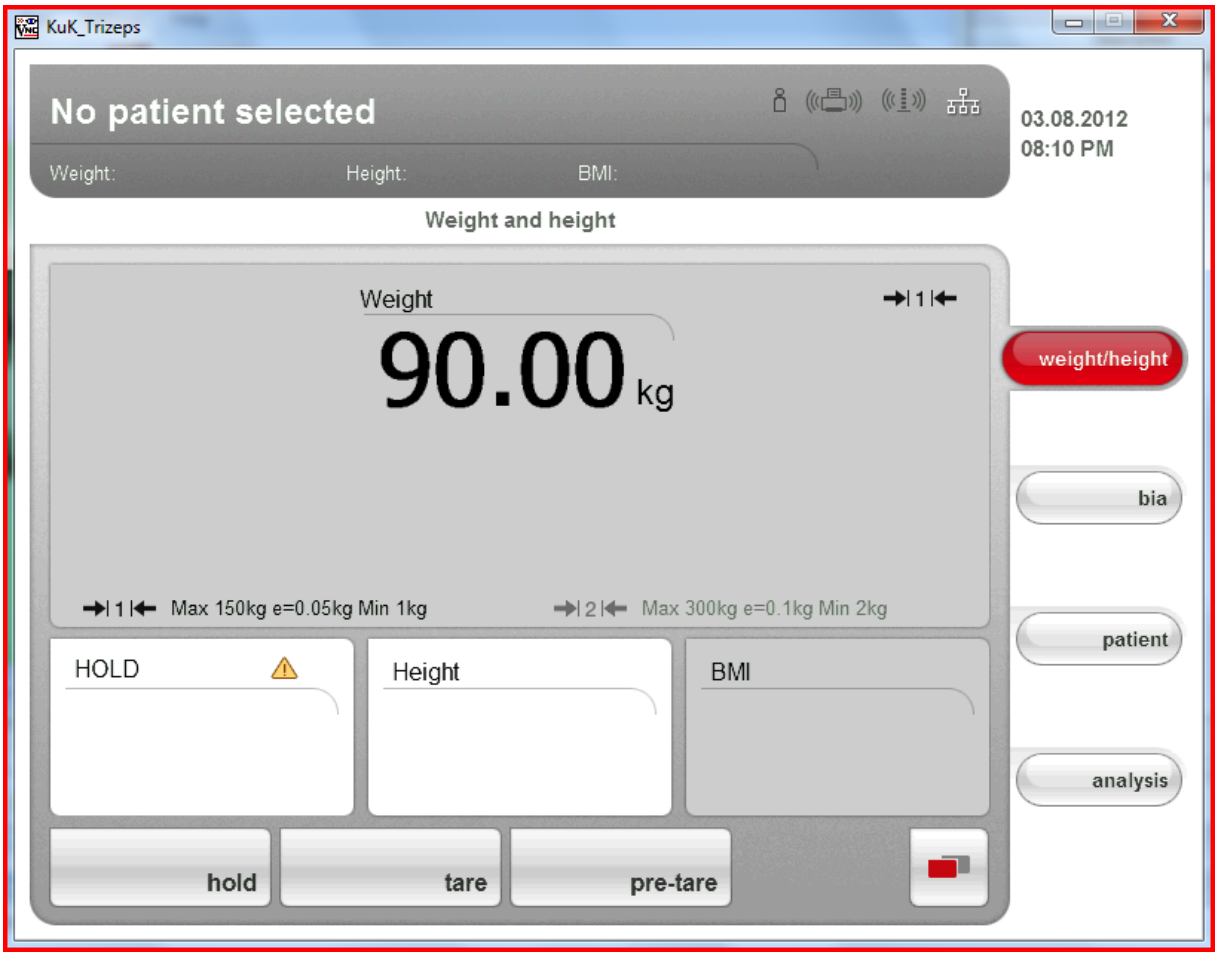


Abbildung 5.1: Das Anzeigefeld des Reiters "weight/height". Im Feld „Weight“ wird das momentan gemessene Gewicht von 90kg angezeigt.

3. Bioelektrische Impedanzanalyse durchfuhren

Als nächstes wird der Reiter „bia“ aktiviert und die BIA-Messung durchgeführt. Es erscheint ein Dialogfenster, in dem eine Modulauswahl vorgeschlagen wird. Die dort vorgeschlagene Standardauswahl soll nicht verändert werden. Durch das Klicken der Taste „continue“ gelangt der Anwender in das nächste Dialogfenster, wo eine Sicherheitsabfrage mit „yes“ oder „no“ durchgeführt wird (Abbildung 5.2).

Abbildung 5.2: Die Sicherheitsabfrage, die im Reiter „bia“ vor einer BIA-Messung durchgeführt wird. Bei gefährdeten Patienten darf die Messung nicht gestartet werden.

Die Bezeichnung der beiden Tasten wechselt dabei in zufälliger Reihenfolge zwischen „yes“ und „no“. Beim Klick auf „yes“ wird die Messung abgebrochen. Die Software aktiviert wieder den Reiter „weight/height“. Beim Klick auf „no“ wird der Messvorgang fortgesetzt und es erscheint ein neues Dialogfenster, wo die korrekte Positionierung des Patienten angezeigt wird. An dieser Stelle muss sich die Testperson entsprechend der eingeblendeten Zeichnung auf dem Gerät positionieren und den Kontakt zu den Hand- und Fußelektroden herstellen. Sobald der Kontakt zwischen der Testperson und den Elektroden hergestellt wurde, startet die Software einen Countdown (3s). Die anschließende Messung startet automatisch und dauert ca. 20 Sekunden. Dabei wird die Restdauer in einem neuen Dialogfenster angezeigt. Sobald die Messung abgeschlossen wurde, erscheint ein neues Dialogfenster mit der Meldung „End of measurement“. Mit der Taste „continue“, gelangt man schließlich in ein weiteres Dialogfenster, in dem der PAL-Wert (Physical Activity Level) eingetragen und mit der Taste „confirm“ bestätigt wird. Für den Testdurchlauf wird hier ein durchschnittlicher Wert von 1,6 gewählt.

4. Messungen einem Patienten zuordnen

Grundsätzlich sind in der seca 515/514 nach einer BIA-Messung drei Szenarien möglich.

- Wenn das Gewicht und die Größe (Abschnitt 1) noch nicht bestimmt bzw. eingetragen wurden, aktiviert das System den Reiter „weight/height“, um die beiden Werte festzulegen.
- Wenn noch kein Patient zugeordnet wurde, aktiviert das System den Reiter „patient“.
- Wenn sowohl ein Patient zugeordnet, als auch das Gewicht und die Größe ermittelt wurden, aktiviert das System den Reiter „analysis“.

Da sowohl Gewicht, als auch Größe bereits definiert sind, wird der Reiter „patient“ aktiviert. Hier kann die BIA-Messung entweder einem bereits angelegten Patienten aus der Patientendatenbank oder einem neu angelegten Patienten zugeordnet werden. Für den Testdurchlauf liegt in der Datenbank ein vordefinierter Patient vor, der in der Suchmaske „Patient search“ über den Vornamen „Test“ gesucht werden kann.

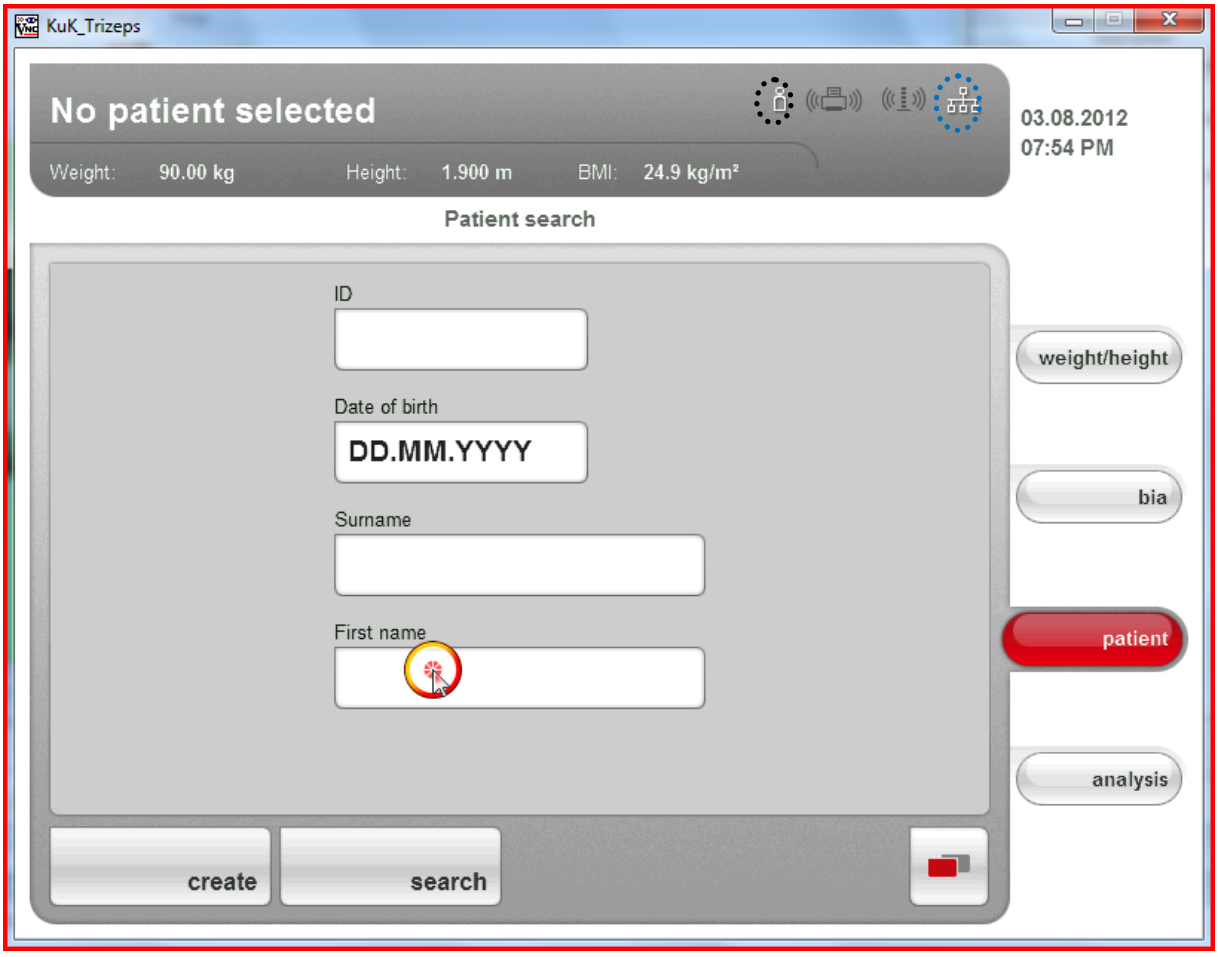


Abbildung 5.3: Dargestellt ist die Suchmaske „Patient search“ im Reiter "patient". Das Symbol im blauen Kreis zeigt, dass zwischen dem mBCA und der seca-Datenbank eine Verbindung über Ethernet hergestellt wurde. Das Symbol im schwarzen Kreis zeigt, dass der Anwender über einen User angemeldet ist.

In der Maske kann außerdem nach der Patienten ID, dem Geburtsdatum, und dem Vornamen gesucht werden (Abbildung 5.3). Nach dem auf die Taste „search“ geklickt wird, startet die Suche in der seca-Datenbank. Dabei greift der mBCA auf die seca-Datenbank zu, was einige

Sekunden andauern kann. Der Eintrag für den gefundenen Patienten „Test“ wird in einer Patientenliste angezeigt. Nach dem Klick auf den Eintrag, wird die Patientenakte mit den patientenspezifischen Daten angezeigt. Durch den Klick auf die Taste „confirm“ wird schließlich die aktuelle BIA-Messung dem ausgewählten Patienten zugeordnet.

5. Messergebnisse überprüfen

Die Überprüfung der Messergebnisse im Reiter „analysis“ ist der sicherheitsrelevanteste Aspekt der Software. Denn die Auswertungen unterstützen den Arzt bei seiner Therapie- und Diagnoseentscheidung. Nach dem die BIA-Messung dem Patienten zugeordnet wurde, aktiviert das System den Reiter „analysis“. Es erscheint ein Dialogfenster mit dem ersten Auswertemodul „Development/growth“ (Abbildung 5.4).

An dieser Stelle wird die BIA-Messung durch den Klick auf die Taste „save“ in der seca-Datenbank gespeichert. Die Auswertungen können somit später über die PC Software seca 115 eingesehen werden.

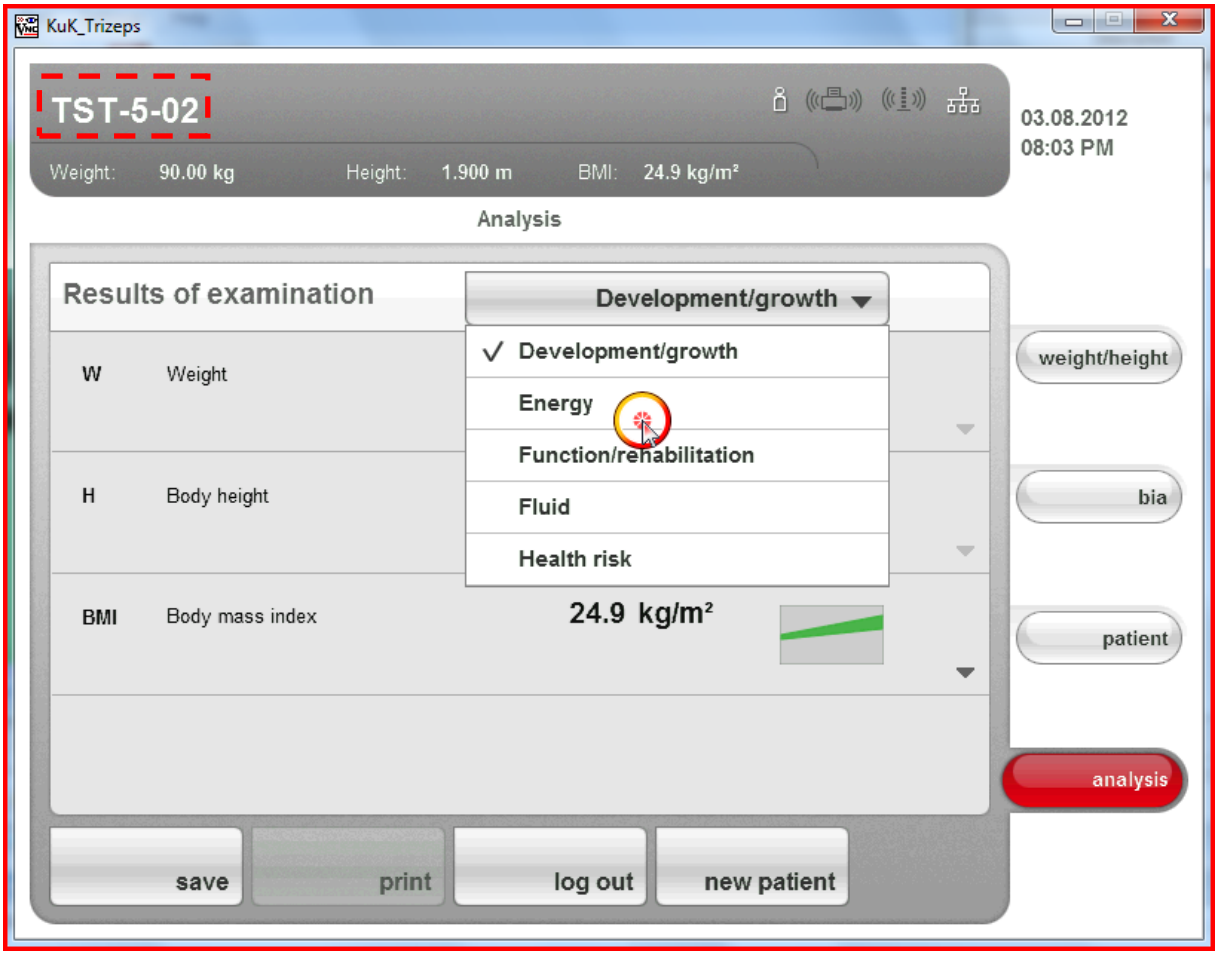


Abbildung 5.4: Dargestellt ist der Reiter „analysis" mit den fünf verschiedenen Auswertungsmodulen. Im der oberen Leiste (roter Rechteck) ist der Patient zu sehen, dem die aktuelle BIA-Messung zugeordnet wurde.

Die anderen fünf Auswertmodule können über das Dropdownmenü angewählt und geöffnet werden. In den fünf Modulen werden insgesamt 18 Kenngrößen und 8 Normbereiche angezeigt. Diese werden mit den separat zu ermittelten Referenzdaten (Soll-Ergebnisse) manuell verglichen.

6. Soll-Ergebnisse ermitteln

Der mBCA ermittelt während der Messung mehrere Körperwiderstandswerte des Patienten, die dem Patienten zugeordnet und beim Speichern in die seca-Datenbank übertragen wurden. Auf der Basis dieser Werte und der patientenspezifischen Daten bestimmt seca 515/514 die Kenngrößen und Normbereiche.

Die Referenzdaten werden ebenfalls in Abhängigkeit von Widerstandswerten und den patientenspezifischen Daten ermittelt. Dazu müssen die notwendigen Informationen ein Excel-Berechnungs-Sheet manuell eingetragen (kopiert) werden. Das Excel-Sheet erlaubt es alle Kenngrößen und Normbereiche zu bestimmen und wurde bei seca interdisziplinär erstellt und überprüft.

Die Soll-Ergebnisse werden anschließend mit den von der seca 515/514 ermittelten Ist-Ergebnissen, die auf dem mBCA angezeigt werden, manuell verglichen.

5.4 Vollständigkeitsprüfung der Testbereiche

Wie im Kapitel „Analyse“ erwähnt, deckt die Systemtestspezifikation der seca 515/514 nicht alle Anforderungen vollständig ab. Vor allem betrifft es die Überprüfung der Auswertungen, also den Testbereich 1 der Testspezifikation, welcher für einen automatisierten Regressionstest am ehesten in Frage kommt. Der Aufwand für eine vollständige Abdeckung der Anforderungen wäre bei der manuellen Testdurchführung viel zu hoch, so dass nur ein Teil der Anforderungen tatsächlich durch die Testfälle abgedeckt wird. Da nun eine automatisierte Testdurchführung eingesetzt wird, sollten die Testfälle aus dem Testbereich 1 vervollständigt werden.

Für die Erstellung weiterer Testfälle wird weiterhin die Black-Box-Technik verwendet. Hierzu können unterschiedliche Vorgehensweisen eingesetzt werden (Kapitel 2.2.3). Für die Anforderungen, die vollständig durch den Testbereich 1 abgedeckt werden sollten, wird die Methode der Äquivalenzklassenbildung gewählt. Diese eignet sich, wie in Kapitel 3 bereits erwähnt, gerade wenn mit Werten und Wertebereichen gearbeitet, die sich in Klassen einteilen lassen.

Sowohl für die Kenngrößen, als auch für die Normbereiche können Äquivalenzklassen gebildet werden. Laut den Anforderungen von seca 515/514 werden insgesamt 18 Kenngrößen definiert, die ermittelt und dargestellt werden (Anhang A2). Acht dieser Kenngrößen werden nicht als reine Werte angezeigt, sondern in Graphiken dargestellt und dabei in bestimmte Normbereiche eingeordnet.

In diesem Kapitel soll das Vorgehen bei der Erstellung von Äquivalenzklassen anhand der Fettmasse-Normbereiche gezeigt werden. Die Erstellung von Äquivalenzklassen für alle Kenngrößen und Normbereiche würde den Umfang dieser Arbeit sprengen. Das vorgestellte Prinzip lässt sich jedoch auf alle anderen Kenngrößen und Normbereiche übertragen.

Fettmasse-Normbereiche

Mit Hilfe der Fettmasse können die Energiereserven bei übergewichtigen oder stark untergewichtigen Patienten bestimmt und die Fettmasseänderung während einer medizinischen Behandlung bzw. eines Krankheitsverlaufes verfolgt werden.

Die Normbereiche helfen dabei die bei der BIA-Messung ermittelte Fettmasse (in %) in einzuordnen (Abbildung 5.5).

Abbildung 5.5: Der Bargraph zeigt den Normbereich der Fettmasse (in %) für einen kaukasischen, männlichen, Patienten, Altersgruppe „30 Jahre“. Bei dem angezeigten Patienten beträgt die Fettmasse 16% und liegt somit im „grünen“ Bereich.

Um nun die Äquivalenzklassen ermitteln zu können, müssen die Anforderungen für die Fettmasse-Normbereiche analysiert werden. Die Normbereiche werden nach den patientenspezifischen Daten Ethnie, Geschlechtstyp und Altersgruppe eingeteilt. Die angezeigten Normbereiche sind unabhängig von der ermittelten Fettmasse.

	Weiblich			Männlich		
	Ethnie			Ethnie		
Alter	**AA (%)**	**AS (%)**	**WT (%)**	**AA (%)**	**AS (%)**	**WT (%)**
30	20-32	25-35	21-33	8-20	13-23	8-21
50	21-34	25-36	23-35	9-22	13-24	11-23
70	23-35	26-36	25-38	11-23	14-24	13-25
Alter		**MSA (%)**	**AN (%)**		**MSA (%)**	**AN (%)**
20-79		24-37	19-38		4-21	4-25

Tabelle 5.4: Die Normbereiche für die Fettmasse kategorisiert nach dem Geschlecht, der Ethnie und den Altersgruppen. AA = afroamerikanisch, AS = asiatisch, WT = kaukasisch, MSA = mittel- und Südamerikanisch, AN = andere.

Die Tabelle 5.4 zeigt die Wertebereiche, die in Abhängigkeit von den patientenspezifischen Daten gebildet werden. Demnach existieren 22 Normbereiche in welche die Kenngröße Fettmasse (in %) eingeordnet werden kann.

Wenn die Altersbereiche „< 20-Jahre“ und „> 79-Jahre“ berücksichtigt werden, bei den kein Normbereiche angezeigt werden dürfen, kommen noch 20 *ungültige* Äquivalenzklassen hinzu.

Insgesamt werden also für die Fettmasse-Normbereiche 42 Äquivalenzklassen (22 gültige und 20 ungültige) gebildet. Auf der Basis dieser Klassen entstehen also mindestens 42 Testfälle, welche die Anforderungen an die Normbereiche der Fettmasse vollständig abdecken. Bisher hat die vorhandene Testspezifikation für die Normbereiche der Fettmasse nur 4 Testfälle beinhaltet, wonach lediglich 9% dieser Anforderungen abgedeckt wurden. Die vorhandene Testspezifikation ist also entsprechend zu vervollständigen.

Test case - ID	**Patient data**	**Expected result**
Ethnic: caucasian		
TST_1_01	age: 30 years gender: male ethnic group: caucasian weight: 90 kg height: 190 cm	The measurements results of seca embedded software and the reference-data (see Excel-Data table “Berechnungen”) should be equal. FM-Bargraph-normal ranges: **8**% und **21**%

Tabelle 5.5: Ein Testfall zur Prüfung des Normbereichs der Fettmasse für einen kaukasischen, männlichen, Patienten, aus der Altersgruppe „30 Jahre“.

Der dargestellte Testfall (Tabelle 5.5) überprüft eine Äquivalenzklasse der Fettmasse-Normbereiche, welche durch die patientenspezifischen Daten „kaukasisch“, „männlich“ und dem Alter von 30 Jahren (Altersgruppe „30 Jahre“) charakterisiert wird. Das Gewicht, die Größe und die Impedanz des Patienten beeinflussen die Normbereiche nicht und können somit beliebig gewählt werden. Alle 42 Testfälle finden sich im Anhang A7.

Fazit

Nach dem gerade vorgestellten Prinzip können bei allen restlichen Kenngrößen und Normbereichen Äquivalenzklassen und entsprechende Testfälle gebildet werden.

Wie das Beispiel des Fettmasse-Normbereiches gezeigt, kann die Anzahl der Testfälle, vergleichsweise hoch werden. Unter Berücksichtigung der Tatsache, dass insgesamt 18 Kenngrößen und 8 Normbereiche in seca 515/514 angezeigt werden können, ist anzunehmen, dass aus den Äquivalenzklassen eine hohe Anzahl zu automatisierender Testfälle abgeleitet werden kann.

Die große Anzahl von Testfällen bedeutet aber nicht, dass für jeden dieser Testfälle eine BIA-Messung mit vordefinierten Eingaben gestartet werden muss. Denn obwohl die Testfälle verschiedene Kenngrößen und Normbereiche überprüfen, sind die Eingabedaten vieler Testfälle identisch. Dies wird dadurch begründet, dass bei den patientenspezifischen Daten, wie Ethnie, Geschlecht und Altersbereiche nur eine begrenzte Anzahl von Kombinationen existiert. Somit können innerhalb einer BIA-Messung mit gleichen Eingabedaten mehrere Kenngrößen und Normbereiche, also mehrere Testfälle, überprüft werden. Diese Tatsache reduziert die Anzahl der zu wiederholenden BIA-Messungen und somit auch den zeitlichen Aufwand für die Testdurchführung erheblich.

Die folgenden Kapitel gehen demnach davon aus, dass die Eingabedaten, die bei den Testfällen für die Fettmasse-Normbereiche eingesetzt wurden, als Eingabedaten für die Tests aller Kenngrößen und Normbereiche eingesetzt werden können. Deshalb werden nicht nur die Fettmasse-Normbereiche überprüft, sondern alle Kenngrößen und Normbereiche, die auf der Basis der Eingabedaten ermittelt wurden.

5.5 Erstellung von Eingabe- und Referenzdaten

Kritische Aspekte

Während der manuellen Testausführung werden das aktuelle Gewicht und der aktuelle Körperwiderstand der Testperson gemessen. Die Bestimmung der Kenngrößen und Normbereiche erfolgt, neben den patientenspezifischen Angaben, auf der Basis dieser Größen. Sogar bei der gleichen Testperson können sich sowohl das Gewicht als auch der Körperwiderstand innerhalb kürzester Zeit verändern. Die ermittelten Referenzdaten müssten deshalb nach jeder BIA-Messung manuell neu ermittelt werden. Das ist sehr zeitaufwändig und fehleranfällig.

Für die spezifischen Patientenangaben lässt sich das Problem lösen, indem Patienten mit den jeweiligen Angaben in einer seca-Datenbank angelegt und schließlich den BIA-Messungen zugeordnet werden. Das konstante Gewicht des Testpatienten kann ebenfalls durch ein festgelegtes und genormtes Eisengewicht ersetzt werden. Um einen gleichbleibenden Körperwiderstand einer Testperson simulieren zu können, wird ein „Normpatient“ eingesetzt. Der Normpatient besteht aus hochwertigen, konstant bleibenden Widerständen und wird über Klemm- und Klebeelektroden mit dem mBCA verbunden.

Nun können auf der Basis von definierten Eingabedaten zu erwartende Referenzdaten ermittelt und schließlich für den automatisierten Testdurchlauf eingesetzt werden.
Die zeitaufwändige und fehleranfällige Ermittlung der Referenzdaten, nach jeder BIA-Messung, ist somit nicht mehr notwendig.

Erstellung von Eingabe- und Referenzdaten

Die Referenzdaten, gegen welche die angezeigten Kenngrößen und Normbereiche überprüft werden sollen, werden auf der Basis von patientenspezifischen Daten, sowie den Messdaten, Gewicht, Größe und Körperwiderstand ermittelt.

Für den automatisierten Testdurchlauf wurden ein konstantes Gewicht und ein Normpatient mit konstanten Widerstandwerten verwendet. Da die Größe ebenfalls konstant gehalten wird, hängen die Referenzdaten nur noch von den patientenspezifischen Angaben (Ethnie, Geschlecht und Alter) ab.

Die patientenspezifischen Angaben dienen als Eingabedaten für die Testfälle der Fettmasse-Normbereiche, die zuvor auf der Basis von 42 Äquivalenzklassen gebildet wurden. Für die insgesamt 42 Testfälle liegen demnach 42 Eingabedatensätze vor. Wie im Kapitels 5.4 bereits erwähnt, sind die Eingabedaten von Testfällen verschiedener Kenngrößen und Normbereiche identisch. So bietet es sich an, nach einer BIA-Messung nicht nur Fettmasse-Normbereiche zu überprüfen, sondern alle Kenngrößen und Normbereiche, die auf der Basis der Eingabedaten berechnet wurden. Damit werden weitaus mehr als die 42 Testfälle abgedeckt, die für die Fettmasse-Normbereiche definiert wurden. Da es sich hier um einen Pilottestdurchlauf handelt, soll auf die Ermittlung der genauen Abdeckung an dieser Stelle verzichtet werden.

Auf Basis von 42 patientenspezifischen Daten und konstanten Messwerten (Gewicht, Größe und Widerstandwert) werden die Referenzdaten für alle Kenngrößen und Normbereiche mit Hilfe des Excel-Berechnungssheets bestimmt und in einer Excel-Tabelle *Berechnungen* erfasst (Anhang A8). Die Reihenfolge in der die einzelnen Referenzdaten während des Testdurchlaufs aus der Excel-Tabelle aufgerufen werden, wird durch den Data-Driven Loop (Datengetriebenes Testen) bestimmt.

5.6 Erstellung von automatisierten Tests

Der Embedded Rechner des mBCA's, auf dem die zu testende Software seca 515/514 ausgeführt wird, verfügt über eigeschränkte Systemressourcen, die für den Einsatz von Test Complete nicht ausreichend sind. So muss, wie bereits in Kapitel 4.2.5 beschrieben, auf die VNC-Lösung zurückgegriffen werden. Die GUI der seca 515/514 wird somit über eine VNC-Verbindung von einem Client-Rechner aus angesteuert und bedient. Test Complete kann in dem eingesetzten VNC-Viewer keine Objekte der seca 515/514 GUI erkennen. So beziehen sich Aktionen lediglich auf die Mauskoordinaten und nicht auf Objekte. Da sich die Position der Tasten und Eingabefelder innerhalb der Maske nicht verändert, sollten bei der koordinatenorientierten Testausführung keine Probleme entstehen. Auch bei der Überprüfung der Auswertungen (z.B. Kenngrößen) oder Fallunterscheidungen (z.B. Sicherheitsabfragen) muss auf die Erkennung der Objekten und Objekteigenschaften verzichtet werden. Hier muss auf die Texterkennung von Test Complete zurückgegriffen.

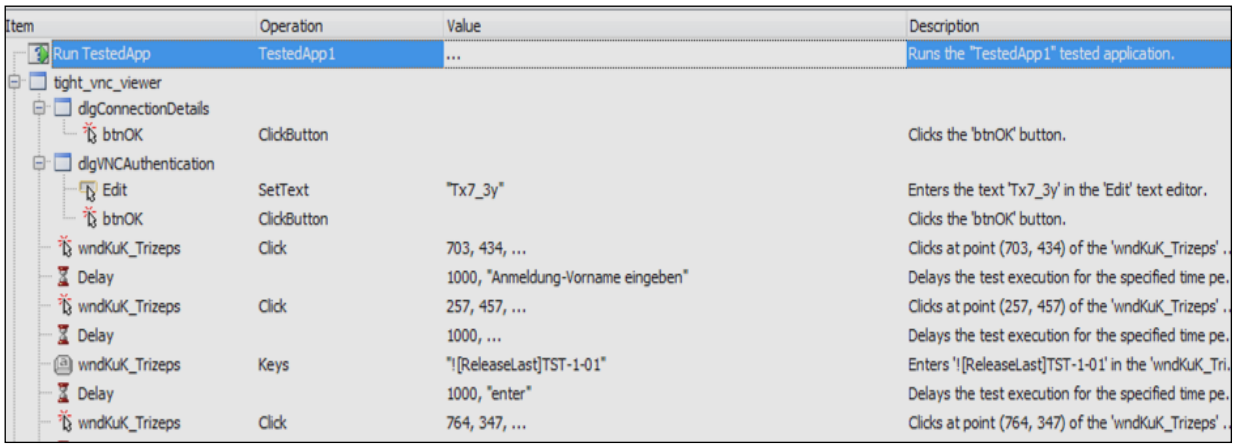
Hier zusammengefasst die Ausgangssituation:

- Es soll überprüft werden, ob die Ergebnisse der Auswertung und der definierten Referenzdaten gleich sind.

- Eine zuvor erstellte seca-Datenbank mit 42 Patienten liegt auf einem Rechner (Windows 7) vor, der über Ethernet mit dem mBCA verbunden ist. TestComplete und der VNC-Server befinden sich ebenfalls auf dem gleichen Rechner.
- Da mit einem VNC-Programm (VNC-Viewer) gearbeitet wird, existiert nur ein Fenster, innerhalb dessen seca 515/514 läuft.
- Das Gewicht von 90kg liegt auf der Wiegefläche und der Normpatient ist ebenfalls an die Elektroden des mBCA's angeschlossen.
- Umfang dieser Implementierung beschränkt sich auf 42 Testfälle, bei denen nur die Werte überprüft werden. Die Messpunkte in den Graphiken werden bei dieser Implementierung nicht überprüft.
- Wie auch bei der manuellen Testausführung kann der automatisierte Test in mehrere Abschnitte unterteilt werden. Hier wurden folgende Abschnitte gewählt: „Testgrundgerüst aufzeichnen", „Anpassen des Keyword Tests", „Überprüfung der Auswertung".

5.6.1 Testgrundgerüst

Nachdem der Initialzustand hergestellt und sowohl Test Complete, als auch der VNC-Viewer gestartet wurden, kann der eigentliche Test mit Hilfe eines Test Complete Recorder aufgezeichnet werden. Alle Aktionen, die bei dem manuellen Test bis zu der Überprüfung der Messergebnisse beschrieben sind, werden von dem Tester ausgeführt und anschließend von Test Complete in einem Keyword Test abgelegt (Abbildung 5.6).



Item	Operation	Value	Description
Run TestedApp	TestedApp1	...	Runs the "TestedApp1" tested application.
tight_vnc_viewer			
dlgConnectionDetails			
btnOK	ClickButton		Clicks the 'btnOK' button.
dlgVNCAuthentication			
Edit	SetText	"Tx7_3y"	Enters the text 'Tx7_3y' in the 'Edit' text editor.
btnOK	ClickButton		Clicks the 'btnOK' button.
wndKuK_Trizeps	Click	703, 434, ...	Clicks at point (703, 434) of the 'wndKuK_Trizeps' ..
Delay		1000, "Anmeldung-Vorname eingeben"	Delays the test execution for the specified time pe.
wndKuK_Trizeps	Click	257, 457, ...	Clicks at point (257, 457) of the 'wndKuK_Trizeps' ..
Delay		1000, ...	Delays the test execution for the specified time pe.
wndKuK_Trizeps	Keys	"![ReleaseLast]TST-1-01"	Enters '![ReleaseLast]TST-1-01' in the 'wndKuK_Tri.
Delay		1000, "enter"	Delays the test execution for the specified time pe.
wndKuK_Trizeps	Click	764, 347, ...	Clicks at point (764, 347) of the 'wndKuK_Trizeps' ..

Abbildung 5.6: Ein Ausschnitt des aufgezeichneten Keyword-Tests in dem ausgewählte Elemente, Operationen, sowie Testdaten und Beschreibungen dargestellt sind.

Der Keyword-Test dargestellt im Advanced Modus (als Visual Basic Script):

```
'Führe "TestedApp1" aus.
Call TestedApps.TestedApp1.Run
'Klicke auf die Taste 'btnOK'.
Call Aliases.tight_vnc_viewer.dlgConnectionDetails.btnOK.ClickButton
```

```
'Gebe den Text 'Tx7_3y' in das 'Edit' Objekt ein.
Call Aliases.tight_vnc_viewer.dlgVNCAuthentication.Edit.SetText("Tx7_3y")
'Klicke auf die Taste 'btnOK'.
Call Aliases.tight_vnc_viewer.dlgVNCAuthentication.btnOK.ClickButton
'Klicke auf die Koordinate (703, 434) des 'wndKuK_Trizeps' Objektes.
Call Aliases.tight_vnc_viewer.wndKuK_Trizeps.Click(703, 434)
'Verzögere die Testausführung um ein bestimmtes Zeitintervall.
Call Delay(1000, "Anmeldung-Vorname eingeben")
'Klicke auf die Koordinate (257, 457) des 'wndKuK_Trizeps' Objektes.
Call Aliases.tight_vnc_viewer.wndKuK_Trizeps.Click(257, 457)
'Verzögere die Testausführung um ein bestimmtes Zeitintervall.
Call Delay(1000)
'Tippe 'TST-1-01' in das 'wndKuK_Trizeps' Objekt ein.
Call Aliases.tight_vnc_viewer.wndKuK_Trizeps.Keys("TST-1-01")
'Verzögere die Testausführung um ein bestimmtes Zeitintervall.
Call Delay(1000, "enter")
'Klicke auf die Koordinate (764, 347) des 'wndKuK_Trizeps' Objektes.
Call Aliases.tight_vnc_viewer.wndKuK_Trizeps.Click(764, 347)
```

5.6.2 Anpassen des Keyword-Tests

Mit dem Test Recorder wurde ein Keyword-Test aufgezeichnet, der alle Grundaktionen beinhaltet. Allerdings werden dort die kritischen Aspekte nicht berücksichtigt und müssen zusätzlich in den aufgezeichneten Testablauf eingebunden werden. Zwei wesentliche Aspekte sind Synchronisation und Timing, sowie die „yes/no“-Sicherheitsabfrage.

Synchronisation und Timing

Während des Testdurchlaufes existieren mehrere Stellen, wo es zu längeren Verzögerungen zwischen den einzelnen Testaktionen kommt. Beim automatisierten Testdurchlauf von seca 515/514 wird die Software über einen Remote-Zugriff (VNC-Viewer) getestet, so dass Test Complete keine Objekte innerhalb von seca 515/514 erkennen kann. Das Tool baut zwischen den Aktionen, die koordinatenorientiert ausgeführt werden, Standartverzögerungen (*Delays*) von 0,5 Sekunden ein. Deshalb müssen an den jeweiligen Stellen diese manuell angepasst werden.

Die Verzögerung verursacht durch die BIA-Messung liegt bei ca. 35 Sekunden. Diese Verzögerung wird durch die Kontaktherstellung zwischen der Testperson und den Elektroden, dem Countdown kurz vor der BIA-Messung und der Messung selbst verursacht.

Bei dem Zugriff des mBCA’s auf die seca-Datenbank kommt es ebenfalls zu mehreren Verzögerungen. So entsteht durch die Patientensuche eine Verzögerung von ca. 3 Sekunden. Die Auswertung der Ergebnisse, nach dem der BIA-Messung ein Patienten zugeordnet wurde, dauert ca. 2 Sekunden und das Speichern der Auswertung in der Datenbank kann sogar bis zu 10 Sekunden. Das muss ebenfalls in dem Test berücksichtigt werden.

„yes/no“-Sicherheitsabfrage

Bevor die BIA-Messung gestartet werden kann, wird eine Sicherheitsabfrage durchgeführt, bei der abgefragt wird, ob der Patient zu einer gefährdeten Kategorie gehört. Mit einem Klick auf die Taste „no“ wird der Messvorgang fortgesetzt und auf die Taste „yes“ abgebrochen. Die Schwierigkeit hierbei ist, dass vor jeder Messung die Belegung der Taste zufällig zwischen „no“ und „yes“ wechselt. Um die aktuelle Belegung der Taste zu bestimmen, könnte Test Complete entweder die Eigenschaften des Steuerelementes „Taste“ auslesen oder die eingebaute Texterkennung den Text bestimmen. Da, wie bereits erwähnt, die Erkennung von Steuerelementen nicht möglich ist, muss die Texterkennung eingesetzt werden. Mit Hilfe der if-then Anweisung wird schließlich die „no“-Taste erkannt und bestätigt. Die BIA-Messung kann daraufhin gestartet werden.

Die Texterkennung im TestComplete kann mit Hilfe der Methode OCRObj.GetText ( ) umgesetzt werden. Die Methode erkennt den Text in einem vordefinierten Bildschirmbereich und gibt diesen zurück. Zuvor wird mit Hilfe der Methode OCR.Create.Object ( ) ein OCRObj Objekt aus dem Bildschirmbereich erzeugt. Das *OCRObj* Objekt wendet OCR (Optical Character Recognition) auf den jeweiligen Bildschirmbereich an.

So lautet das VB-Skript für die Texterkennung:

```
sub YesNo (x,y,v,w,n)


  '  Aktiviere das VNC Fenster.
  Set vnc = Sys.Process("tight-vnc-viewer")
  Set Window = p.Window("VNCviewer","KuK_Trizeps",1)
  '  Erfasse den betroffenen Bildausschnitt im VNC Fensters und poste
     diesen im Log. Die Variablen x,y,v,w definieren Koordinaten, deren
     Werte in einer Tabelle abgelegt sind.
  Set Rect = u.Picture(x, y, v, w)
  Call Log.Picture(Rect, "Ausschnitt mit dem zu erkennenden Text.")
  '  Erzeuge ein OCR Objekt.
  Set OCRObj = OCR.CreateObject(Rect)
  '  Erkenne den Text und poste im test-log.
  ist = OCRObj.GetText()
  Call Log.Message(ist, "Erkannte Text.")
```

Der VB-Skript für die Texterkennung wird als „Script Routine“ in den Keyword-Test eingebunden. Falls es sich bei dem erkannten Text um die Taste „no“ handelt, soll eine vordefinierte Stelle innerhalb der Taste angeklickt werden. Falls es sich nicht um die Taste „no“ handelt, soll der Mausklick an die vordefinierte Stelle im Bereich der anderen Taste erfolgen. D.h. es muss an dieser Stelle eine Fallunterscheidung eingebaut werden:

```
  'Prüfe ob der erkannte Text mit der Variablen n, die als Text „no"
   definiert ist, übereinstimmt.
If (ist = n) then
  'Klicke die Position (216, 497) des VNC Fensters an.
   Call Aliases.tight_vnc_viewer.wndKuK_Trizeps.Click(216, 497)
Else
  'Klicke die Position (454, 491) des VNC Fensters an.
   Call Aliases.tight_vnc_viewer.wndKuK_Trizeps.Click(454, 491)
End If
```

Hier ist anzumerken, dass die Variablen *x,y,c,w,n* auch als Konstanten hätten definiert werden können. Denn sowohl die Koordinaten des Bildausschnitts, als auch der Soll-Text der Taste bleiben gleich. Damit allerdings das Testskript robuster und wartbarer ist, werden dennoch Variablen eingesetzt.

Datengetriebener Test

Bei jedem Testdurchlauf müssen sowohl verschiedene Patienten mit ihren patientenspezifischen Daten, als auch die Größe des Patienten in den Test eingebunden werden.

Da der Testablauf gleich bleibt und sich lediglich die Eingabedaten gering unterschieden, eignet sich die Data-Driven Loop Funktion von Test Complete. Dazu werden die Eingaben für den Patientennamen und die Größe parametrisiert und in der Excel-Tabelle *Testfaelle* abgelegt. Die Excel-Tabelle *Testfaelle* enthält die Spalten „Name“ und „Groesse“ in denen Patientennamen und Größenwerte (in cm) abgelegt sind (Tabelle 5.6).

Name	Groesse
TST-1-01	190
TST-1-02	190
TST-1-03	190
…	…
…	…

Tabelle 5.6: Ausschnitt der Excel-Tabelle *Testfaelle*

Der Data-Driven Loop ermöglicht es dann diese Daten bei jedem Testdurchlauf dynamisch aufzurufen und in den das entsprechende Eingabefeld einzusetzen. So werden die Patientennamen über die Patientensuche in das Feld „First name“ eingetragen. Der Patient mit seinen spezifischen Daten wird dann aus der seca-Datenbank geladen der BIA-Messung zugeordnet. Die Größenwerte werden entsprechend in das Feld „Größe“ eingetragen und von seca 515/514 übernommen.

Die modifizierten Befehle sehen im VB-Skript folgender Maßen aus:

```
    'Füge den Inhalt der Tabelle „Testfaelle", Spalte „Groesse" an die
    zuvor ausgewählte Stelle des VNC-Fensters ein.
Call Aliases.tight_vnc_viewer.wndKuK_Trizeps.Keys
(Project.Variables.Testfaelle.Value("Groesse"))
    'Füge den Inhalt der Tabelle „Testfaelle", Spalte „Name" in das VNC
    Fenster ein.
Call Aliases.tight_vnc_viewer.wndKuK_Trizeps.Keys
(Project.Variables.Testfaelle.Value("Name"))
```

Der Aufruf Project.Variables.Testfaelle.Value ( ) gibt die Werte der Excel-Tabelle *Testfaelle* zurück. Die Zeile wird dabei durch den Data-Driven Loop vorgegeben. In der Standardeinstellung werden alle Zeilen der jeweiligen Spalte der Reihe nach durchlaufen.

Es werden 42 BIA-Messungen durchgeführt bei den 42 verschiedenen Patienten aus der seca-Datenbank geladen werden. Insgesamt werden also 42 Zeilen der Tabelle *Testfaelle* durchlaufen.

5.6.3 Überprüfung von Kenngrößen und Normbereichen

Die Automatisierung aller Aktivitäten, die bis zur Auswertung durchgeführt werden ist abgeschlossen, und es können nun die von seca 515/514 angezeigten Auswertungen im Reiter „analysis“ mit den Referenzdaten verglichen werden.

Für die Überprüfung einzelner Kenngrößen und Normbereiche muss, ähnlich wie bei der Sicherheitsabfrage, auf die Texterkennung von Test Complete zurückgegriffen werden. Das VB-Skript zur Erkennung von Kenngrößen und Normbereichen sieht wie folgt aus:

```
sub Werteueberpruefung (x,y,v,w,soll,medPar)

  '  Aktiviere das VNC Fenster.
  Set p = Sys.Process("tight-vnc-viewer")
  Set u = p.Window("VNCviewer","KuK_Trizeps",1)
  '  Erfasse den betroffenen Bildausschnitt im VNC Fensters und poste
     diesen im Log. Die Variablen x,y,v,w definieren Koordinaten, deren
     Werte in einer Tabelle abgelegt sind.
  Set Rect = u.Picture(x, y, v, w)
  Call Log.Picture(Rect, "Ausschnitt mit dem zu erkennenden Text.")
  '  Erzeuge ein OCR Objekt.
  Set OCRObj = OCR.CreateObject(Rect)
  '  Erkenne den Text.
  ist = OCRObj.GetText()
```

Die Bildausschnittskoordinaten *x,y,v,w* aller Kenngrößen und Normbereiche wurden einzeln mit Hilfe von Test Complete bestimmt und in der Excel-Tabelle *Berechnungen* erfasst.

Um zu prüfen, ob die Texterkennung überhaupt Zeichen erkennen konnte, wird die folgende if-then Anweisung in das Skript eingebaut:

```
  'Prüfe, ob überhaupt Zeichen erkannt wurden.
  If (Len(ist) = 0) then
    'Im test-log notieren, dass keine Zeichen erkannt wurden.
    log.Error("Kein Wert erkannt.")
  Else
   'Im test-log den Ist-, Soll-Ergebnis und den medizinischen Parameter
   sind.
    Call Log.Message(ist, "Ist-Ergebnis")
    Call Log.Message(soll, "Soll-Ergebnis")
    Call Log.Message(medPar, "Medizinischer Parameter")
```

Diese Anweisung erhöht die Robustheit des Testskripts. Falls kein Inhalt erkannt werden konnte, wird es mit einem *Error* im Testlog signalisiert. Falls doch ein Inhalt erkannt wurde, soll dieser mit den Referenzdaten aus der Excel-Tabelle *Berechnungen* verglichen werden. Das wird mit der folgenden if-then Anweisung umgesetzt:

```
      'Prüfe, ob der erkannte „ist"-Wert mit dem Parameter der Variable
       „soll", also den Referenzdaten, übereinstimmt.
    If (soll = ist) then
       'Falls „soll" und „ist" übereinstimmen, poste einen Checkpoint im
        Testlog
      log.Checkpoint("Korrekte Berechnung")
     'Falls „soll" und „ist" nicht übereinstimmen, poste einen Error
      im Testlog.
    Else
      log.Error("Falscher Wert erkannt")
    End If
  End If
end sub
```

Die Variablen *x,y,v,w,* sowie *soll* und *medPar* greifen auf die jeweiligen Zeilen aus der zuvor angelegten Excel-Tabelle *Berechnungen* zu. Der Data-Driven Loop steuert diesen Prozess und ermöglicht es, dass bei jedem Testdurchlauf die Zeileninhalte der gewünschten Spalte für den Abgleich in den Testablauf eingebunden werden.

Nun sind die gewünschten Testskripte einsatzbereit. Das Skript erlaubt es bei jedem Testdurchlauf die definierten Eingabe- und die Referenzdaten aus den Excel-Tabellen in den Test einzubinden und mit den Ist-Ergebnissen der Auswertung zu vergleichen

5.7 Fazit

Bei dem automatisierten Test wurden 42 BIA-Messungen mit 42 verschiedenen Patienten ausgeführt. Während eines Testdurchlaufs wurden im Reiter Auswertung 18 Kenngrößen, sowie 8 Normbereiche überprüft. In der Gesamtzahl wurden 1458 Ausgaben der Software mit den Referenzdaten verglichen.

Der Gesamttest dauerte 2 Stunden und 56 Minuten, im Durchschnitt also 4 Minuten und 12 Sekunden pro Testdurchlauf.
Zum Vergleich: Ein manueller Testdurchlauf dauert ca. 30 Minuten. Denn neben der manuellen Bedienung des Gerätes, müssen auch die Referenzdaten nach jeder BIA-Messung auf der Basis der aktuelle gemessenen Körperwiderstandwerte und Gewichtswerte neu bestimmt und gegen die Softwareausgaben verglichen werden. Für eine manuelle Testausführung von 42 BIA-Messungen und die anschließende Überprüfung der Auswertungen müsste ein Tester demnach 21 Stunden investieren. Das wäre aus der Sicht des Kosten- und Zeitaufwandes nicht zu vertreten.

Der automatisierte Test zeigte auch im Bezug auf Fehlererkennung erste Erfolge. So konnte in der getesteten seca 515/514 (Version 1.0.17.3509) ein Fehler bei der oberen Grenze des Fettmasse-Normbereichs detektiert werden. Die obere Grenze wurde bei einem weiblichen Patienten der Ethnie „andere“ und einem Alter im Bereich von 20-79 Jahren (Testfall-ID: TST-5-05) mit 19% falsch implementiert. Der durch die Anforderung vorgeschriebene Wert liegt bei 38%.

Neben den eben beschriebenen Erfolgen konnten während des automatisierten Tests auch einige Probleme beobachtet werden. So hat die Texterkennung bei jedem Test an zwei bis fünf Stellen das angezeigte Ergebnis falsch interpretiert. Am häufigsten wurden die Zahlen „3“ und „5“ nicht erkannt, so dass nach dem Abgleich mit den Referenzdaten ein Error in dem Testlog angezeigt wurde. Außerdem zeigten einige Ergebnisse kleinere Abweichungen, die bedingt durch die Messtoleranzen des mBCA’s zustande kamen. Diese Abweichungen wurden allerdings bei dem Vergleich von Ist- und Sollergebnissen nicht berücksichtigt, so dass Test Complete an diesen Stellen Errors gemeldet hat. Nach den 42 Testdurchläufen wurden somit zwischen 180 und 200 „falsche“ Errors angezeigt. Diese „falschen“ Fehler konnten allerdings bei der anschließenden Durchsicht des Testlogs relativ einfach erkannt und übersprungen wurden. Denn nach jeder Texterkennung (Ist-Ergebnis) und dem anschließendem Abgleich mit dem Soll-Ergebnis wurden sowohl das Ist-Ergebnis, als auch das Soll-Ergebnis in dem Testlog gepostet. So konnte relativ schnell nachvollzogen werden, ob es sich bei dem Error tatsächlich um einen Fehler handelt oder nur um eine fehlerhafte Texterkennung seitens Test Complete.

In dem folgenden, abschließenden Kapitel werden die Ergebnisse der Analyse, der Konzepterarbeitung, sowie der Konzeptumsetzung nochmal zusammengefasst.

6 Zusammenfassung und Ausblick

Ziel dieser Arbeit war es ein Testkonzept zu erarbeiten, welches effizient, effektiv und entwicklungsbegleitend eingesetzt werden kann.

Als Beispiel für dieses Thema wählte diese Arbeit das Unternehmen seca. In der Ausgangssituation setzte seca als maßgebliche QS-Maßnahme einen entwicklungsabschließenden Systemtest ein. Entwicklungsbegleitende QS-Maßnahmen in früheren Stadien der Entwicklung setzte seca nur sporadisch und nicht konsequent bei allen Produkten ein. Dies führte bislang zu höherem "Fehlerdruck" auf den Systemtest und schließlich insgesamt zu einer verringerten Fehlerentdeckungsrate durch den Systemtest. Bei seca 515/514 war die Effektivität des Systemtests mit 59% am geringsten. So wurden zum Teil sicherheitskritische Fehler in der Auswertung erst nach Entwicklungsabschluss entdeckt.

Diese Arbeit identifizierte zwei grundlegende Probleme: unvollständige Testspezifikationen, sowie unvollständige Software-Qualitätssicherung bei der Entwicklung einer medizinischen Software der Sicherheitsklasse A.

Die fehlende Abdeckung von sicherheitskritischen, funktionalen Anforderungen, speziell bei seca 515/514, führt zu einer Reduktion der Effektivität des Systemtests. Denn insbesondere bei seca 515/514 waren die Anforderungen, welche nicht durch die Testspezifikation abgedeckt wurden, anfällig für eine Fehlinterpretation durch die Entwickler. Als Konsequenz konnte der Systemtest diese Fehler jedoch ebenfalls nicht identifizieren. Die mangelnde Vollständigkeit der Testspezifikation war durch ein Effizienzproblem der manuellen Testdurchführung verursacht. In der Ausganssituation war eine vollständige Abdeckung aus Zeit- und Kostengründen folglich nicht rentabel.

QS-Maßnahmen für Software der Sicherheitsklasse A sind laut DIN EN 62304 bei der Entwicklung nicht zwingend nötig. So können Unternehmen selber entscheiden, ob und welche QS-Maßnahmen sie tatsächlich anwenden. Dieser Ansatz hat sich aufgrund der erhöhten Fehleranfälligkeit als nicht sinnvoll herausgestellt. Entwicklungsbegleitende QS-Maßnahmen sollten zum Standardrepertoire jedes Unternehmens im medizinischen Umfeld gehören.

Um diese Probleme zu lösen empfiehlt die vorliegende Arbeit den verstärkten Einsatz entwicklungsbegleitender Tests. Da seca bestimmte Produkte in Kooperation mit externen Dienstleistern entwickelt, steht der zugrunde liegende Quellcode mitunter nicht zur Verfügung. Deshalb empfiehlt diese Arbeit, risikoorientierte Black-Box Regressionstests zu verwenden. Das Effizienzproblem der manuellen Testdurchführung wird durch den Einsatz von Testautomatisierung und entsprechenden Werkzeugen gelöst.

Das hierfür erarbeitete Testkonzept für den risikoorientierten und automatisierten Black-Box-Regressionstest besteht aus der vorgelagerten Auswahl eines Testautomatisierungstools und

einem kontinuierlich durchgeführten Testprozess. Für die Auswahl des Testautomatisierungstools stellt diese Arbeit transparente Kriterien vor, welche Unternehmen anwenden können, um Anforderungen an die Testautomatisierung wirtschaftlich effizient umzusetzen. Der Testprozess identifiziert zunächst Testbereiche innerhalb der Systemtestspezifikation. Im Falle einer zu geringen Testabdeckung wird diese durch Anwendung der Black-Box-Technik erweitert. Dieser Ansatz löst auch das oben beschriebene Problem der unvollständigen Testspezifikation. Im Anschluss an diesen Schritt priorisiert man die Testbereiche gemäß der Kriterien Risiko, Schwachstellen und Automatisierungseignung. Dies resultiert in der Wahl eines geeigneten Testbereiches für die Automatisierung.

Im Rahmen der Erstellung dieser Arbeit wurde dieses Konzept bei seca 515/514 implementiert, weil dort die eingesetzten QS-Maßnahmen, vor allem der Systemtest, die geringste Effektivität gezeigt haben und das Produkt hauptsächlich von einem externen Dienstleister entwickelt wird.

Der Testbereich 1, in dem die BIA-Messung, sowie die Ergebnisse der Auswertung überprüft werden, hat sich als der Bereich mit der höchsten Priorität herausgestellt. Er zeigte das höchste Risiko und hatte prozentual die meisten durch den Systemtest unentdeckten Fehler. Da dieser Testbereich sich auch gut für die Automatisierung eignete, war es möglich, die Testautomatisierung am praktischen Beispiel zu demonstrieren. In diesem Zusammenhang wurden die kritischen technischen Aspekte bei der Erstellung von definierten Eingabe- und Referenzdaten durch den Einsatz eines Normpatienten und eines konstanten Gewichtes gelöst. An den kritischen Stellen bei der Umsetzung von automatisierten Tests wurde hingegen mit dem VNC-Viewer, der if-then Anweisungen, Delays, sowie Data-Driven Loops gearbeitet.

Ein Nachteil des Testkonzeptes, bedingt durch die Verwendung der GUI-Prüftechnik ergibt sich, wenn die Oberflächenobjekte der zu testenden GUI nicht erkannt werden. Dann muss mit Bild- und Texterkennung gearbeitet werden. Die Texterkennung hat sich in dieser Arbeit als fehleranfällig erwiesen, deshalb sollte in der Zukunft darauf weitestgehend verzichtet werden. Das ist allerdings nur dann möglich, wenn entweder die GUI-Prüftechnik nicht mehr eingesetzt wird oder wenn auf die Oberflächenobjekte zugegriffen werden kann. Das letztere wird allerdings durch den Einsatz von VNC verhindert. Eine Möglichkeit wäre es die seca 515/514 auf einem stationären PC, z.B. als native Windows-Anwendung, welche auch auf das Embedded-Gerät portierbar ist, auszuführen. Die Anwendung wäre dann über ein Netzwerk mit dem Gerät verbunden und könnte dieses ansteuern.

Zudem ist es ineffizient, für jeden Test der Auswertungsergebnisse zunächst eine BIA-Messung durchzuführen. Zwar werden auf diese Weise auch alle Aktionen vor der Auswertung getestet, im Rahmen der Tests, welche die Ergebnisse der Auswertung betreffen, ist dies jedoch nicht notwendig und zeitaufwändig.

Als problematisch stellt sich auch heraus, dass das System größtenteils keine definierte Veränderung einzelner Messwerte zulässt. Werte müssen vornehmlich tatsächlich gemessen werden, bevor sie in die Auswertung übernommen werden.

Beide Probleme können gelöst werden, wenn die in der seca-Datenbank hinterlegten Messwerte eines Patienten bei Bedarf auf den mBCA geladen werden. Dies ermöglicht der Soft-

ware auf Basis dieser Werte die Auswertung durchzuführen. Deren Ergebnisse können im nächsten Schritt deterministisch und automatisiert überprüft werden. Mit diesem Feature kann bei dem Test der Auswertungen auf eine reale BIA-Messung verzichtet werden. Hierdurch ist eine erhebliche Reduktion der Testausführungszeit möglich. Da es möglich ist, sämtliche Messwerte in der seca-Datenbank zu definieren, können zudem beliebige Testdaten generiert werden. Dies führt zu einer weiteren Vereinfachung der automatisierten Testausführung.

Ein weiteres Problem sind die Messtoleranzen des mBCA's. Diese führen dazu, dass die Ergebnisse der Auswertung bei manchen Messungen variieren, was bei dem Abgleich zwischen den konstanten Referenzwerte zu Errors im Testlog führt. Hier können bei dem Abgleich mit den Referenzwerten entsprechende Toleranzen eingebaut werden, um die Messabweichungen mitberücksichtigen.

Das Testkonzept geht nicht auf die strategischen Fragen des Testmanagements ein. Hier sind einige Fragen aufgelistet, die zukünftig durch das Unternehmen beantwortet werden sollten:

1. Wie kann der Regressionstest in dem Entwicklungsprozess umgesetzt werden? D.h. nach welchen Änderungen in der Software soll der Regressionstest durchgeführt werden? Und wann soll der Regressionstest ausgeführt werden?
2. Soll der Regressionstest durch weitere Testbereiche ergänzt werden?
3. Stehen menschliche bzw. technische Ressourcen zur Verfügung?
4. Wie kann die Wartung und Pflege der Testfälle realisiert werden?
5. Wie kann der Regressionstest sinnvoll verbessert werden?

Literatur

[BORL 06] Borland GmbH: Requirements Based Testing: Engpässe im Testprozess vermeiden, 2006

[BRCA 03] Brcan, Gregor & Hänsel, Jochen; Automatisierung von GUI-Tests, Seminar Softwaretechnik, TU Berlin, 2003

[BUGZ 12] The Bugzilla Guide: How do I use Bugzilla?, http://www.bugzilla.org/docs/2.16/html/how.html, Zugriff: 30.11.2012

[DENG 07] Denger, Christian: Studie zum Stand der Softwareentwicklung in der Medizintechnik - Management Zusammenfassung, 2007

[DIN EN 13485] DIN EN 13485: Medizinprodukte - Qualitätsmanagementsysteme - Anforderungen für regulatorische Zwecke

[DIN EN 14971] DIN EN 14971: Medizinprodukte - Anwendung des Risikomanagements auf Medizinprodukte

[DIN EN 62304] DIN EN 62304: Medizingeräte-Software, Software-Lebenszyklus-Prozesse

[DIN EN 9001] DIN EN 9001: Qualitätsmanagementsysteme - Anforderungen

[DIN EN 9126] DIN ISO 9126: Softwarequalitätsmerkmale

[EMAM 98] Emam, K. El: Benchmarking Kappa for Software Process Assessment Reliability Studies, ISERN Report, 1998

[FREI 05] Freimut; Denger; Ketterer: An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management, Proceedings of the 11th. International Symposium on Software Metrics, 2005

[HOFF 08] Hoffmann, Dirk W. : Software-Qualität, 1. Auflage, Springer, 2008

[IEEE 03] Investigating the defect detection effectiveness and cost benefit of nominal inspection teams, IEEE Transactions on Software Engineering, Volume: 29, Issue: 5, 2003

[IEEE 90] The Institute of Electrical and Electronics Engineers: IEEE Standard Glossary of Software Engineering Terminology, 1990

[LIGG 09] Liggesmeyer, Peter: Software-Qualität: Testen, Analysieren und Verifizieren von Software, 2.Auflage, Spektrum Akademischer Verlag, 2009

[ROSE 06] Rosenthal, Klaus-Peter; Siwon, Peter: "Leseprobe Trend Guide „Embedded Test“, 2006

[SPILL 08] Spillner, Andreas: Das W-Modell-Vorteile der agilen Prozesse in einem konservativen Umfeld nutzen, Hochschule Bremen - Zentrum für Informatik und Medientechnologien, 2008

http://www.gi-hb-ol.de/uta/gi-rg/WModellSpillner.pdf Zugriff: 28.10.2012

[SPILL 10] Spillner, Andreas: Basiswissen Softwaretest, 4. Auflage, dpunkt.verlag GmbH, 2010

[THALL 02] Thaller, Georg Edwin: Softwaretest, Verifikation und Validation, 2. Auflage, Heise Medien, 2002

[WINT 01] Winter, Mario; Sneed, Harry M.: Testen objektorientierter Software, Hanser Fachbuch, 2001

Anhang

A1 Funktionsprinzip der bioelektrischen Impedanzanalyse (BIA)

Problem

Der derzeitige sogenannte Goldstandard zur Analyse der Körperzusammensetzung besteht aus einer Kombination von technisch teilweise sehr aufwändigen und zeitintensiven Methoden zur Ermittlung von einzelnen Kenngrößen. Durch den hohen technischen und finanziellen Aufwand gepaart mit hohem Zeit- und Raumbedarf ist der sogenannte Goldstandard für den Alltag in Kliniken und Arztpraxen ungeeignet.

Lösung

Die bioelektrische Impedanzanalyse (BIA) ist eine Methode zur schnellen, einfachen und nicht-invasiven Abschatzung der Körperzusammensetzung. Es ist nur ein einzelner Messvorgang an einem einzelnen Gerät notwendig. seca hat in aufwändigen Studien nachgewiesen, dass der mBCA mit einem durchschnittlichen Fehler von 0,98% mit dem Goldstandard vergleichbar ist. Das kann kein anderer Hersteller.

Bei der BIA-Methode wird der menschliche Körper als elektrischer Leiter in einem Wechselstromkreis betrachtet und sein Wechselstromwiderstand (Impedanz) gemessen.

Wird ein geringer Wechselstrom über Elektroden an Armen und Beinen in den Körper geleitet und über jeweils ein zweites Elektrodenpaar der Spannungsabfall gemessen, können dabei verschiedene Komponenten der Körperimpedanz ermittelt werden.

Aus den gemessenen Größen kann in Verbindung mit Gewicht, Größe, Alter, Ethnie und Geschlecht eines Patienten die Körperzusammensetzung des Patienten berechnet und anschließend beurteilt werden. Voraussetzung dafür sind nach wissenschaftlichen Gesichtspunkten ermittelte Formeln.

Deshalb hat seca in Zusammenarbeit mit verschiedenen Instituten und Kliniken populationsspezifische Prädikationsformeln zur Ermittlung von Kenngrößen entwickelt.

Die seca-eigenen Formeln sind in seca mBCA-Geräten und der seca PC-Software implementiert. Damit ist seca Pionier in der wissenschaftlich fundierten, medizinisch aussagekräftigen Bestimmung der Körperzusammensetzung per bioelektrischer Impedanzanalyse.

A2 Auswertmodule

Entwicklung/Wachstum

Ziel dieses Modules ist die Überwachung von Wachstumsverläufen und Gewichtsveränderungen im Zuge eines Krankenhausaufenthalts oder einer medizinischen Behandlung. Speziell bei Kindern unterstutzt dieses Modul regelmäßige Kontrolluntersuchungen zur Beurteilung der Wachstumsentwicklung. Es werden folgende Kenngrößen dargestellt:

• Gewicht

• Größe

• Body Mass Index (BMI)

Energie

Ziel dieses Modules ist die quantitative Bestimmung des Energieverbrauchs und der Energiereserven des Körpers zur Beurteilung von Gewichtsveränderungen, Krankheitsverläufen und des allgemeinen Ernährungszustands eines Patienten. Es stellt folgende Kenngrößen dar:

• Fettmasse (FM) in kg

• Fettmasse (FM) in %

• Im Körper gespeicherte Energie (Körper)

• Ruheenergieverbrauch (REE)

• Gesamtenergieverbrauch (TEE)

Funktion/Rehabilitation

Dieses Modul dient der Bestimmung des Fitnesszustandes sowie der Beurteilung der Stoffwechselaktivität und des Trainingserfolgs z. B. im Rahmen einer Rehabilitations- oder Physiotherapie. Folgende Kenngrößen werden dargestellt:

• Fettfreie Masse (FFM)

• Fettmasse (FM) in kg

• Fettmasse (FM) in %

• Fettmasse-Index (FMI)

• Fettfreie-Masse-Index (FMMI)

• Skelettmuskelmasse (SMM)

• Weichteilmagermasse (LST)

Flüssigkeit

Das Flüssigkeitsmodul ermöglicht die Bestimmung des Flüssigkeitsstatus eines Patienten, sowie die Beobachtung von Flüssigkeitsveränderungen in Folge einer medizinischen Behandlung. Es werden folgende Kenngrößen dargestellt:

• Gesamtkörperwasser (TBW)

• Extracellulares Wasser (ECW)

• Hydration (HYD)

• Bioelektrische Impedanzvektoranalyse (BIVA)

Gesundheitsrisiko

Ziel ist die Bestimmung des allgemeinen Gesundheitszustands oder bei bereits bekannter Erkrankung die Beurteilung des Schweregrades. Es werden folgende Kenngrößen dargestellt:

• Phasenwinkel φ bei 50Hz

• Hydratation (HYD)

• Bioelektrische Impedanzvektoranalyse (BIVA)

• Fettmasse-Index (FMI)

• Fettfreie-Masse-Index (FMMI)

A3 Checkliste aus der DIN EN 62304, Punkt 5.2.2

a. Anforderungen an die Funktionalität und die Leistungsfähigkeit
b. Ein- und Ausgaben des Software-Systems
c. Schnittstellen zwischen dem Software-System und anderen Systemen
d. Software gesteuerte Alarme, Warnungen und Anwender-Meldungen
e. Anforderungen für die Datensicherheit
f. Anforderungen an die Gebrauchstauglichkeit, die maßgeblich sind hinsichtlich menschlichem Versagen und Ausbildung
g. Datendefinition und Datenbank Anforderungen
h. Anforderungen für die Installation und Abnahme der gelieferten Medizinprodukte Software am Standort / an den Standorten der Anwendung und Wartung
i. Anforderungen hinsichtlich Methoden des Betriebs und der Wartung
j. Anwender-Dokumentation, die entwickelt werden soll
k. Anforderungen hinsichtlich der Wartung durch den Betreiber und
l. regulatorische Anforderungen

A4 Fehlerverteilungen für seca 115, seca 101 und seca 515/514

seca 115

Fehlerverteilung der Fehlerquellen des Experten 1

Klassifizierung	Fehler-Quelle	absolute Fehleranzahl	relative Fehleranzahl
1	Implementierung	65	0,77
2	Anforderungen	17	0,20
3	Andere	2	0,02

Fehlerverteilung der Fehlersenken des Experten 1

Klassifizierung	Fehler-Senke	absolute Fehleranzahl	relative Fehleranzahl
1	Systemtest	55	0,65
2	Release	15	0,18
3	Andere	14	0,17

Fehlerquellen und –senken für einzelne Fehler

ID	Status	Resolution	Fehler-Quelle, Experte 1	Fehler-Senke, Experte 1	Fehler-Quelle, Experte 2	Fehler-Senke, Experte 2
258	RESOLVED	FIXED	1	2	2	2
260	RESOLVED	FIXED	1	2	1	2
302	RESOLVED	FIXED	2	2	2	2
313	RESOLVED	FIXED	1	2	1	2
322	RESOLVED	FIXED	2	2	2	2
349	CLOSED	FIXED	1	2	1	2
357	CLOSED	FIXED	1	2	1	2
358	CLOSED	FIXED	1	1	1	1

359	CLOSED	FIXED	1	1	1	1
360	CLOSED	FIXED	1	1	1	1
361	CLOSED	FIXED	1	1	1	1
362	RESOLVED	FIXED	1	1	1	1
363	RESOLVED	FIXED	1	1	1	1
370	CLOSED	FIXED	2	3	2	1
372	RESOLVED	FIXED	2	3	2	3
373	RESOLVED	FIXED	2	1	2	1
376	RESOLVED	FIXED	1	1	1	1
378	RESOLVED	FIXED	1	1	1	1
379	RESOLVED	FIXED	1	3	1	3
409	RESOLVED	FIXED	1	3	1	3
410	RESOLVED	FIXED	3	1	1	1
423	RESOLVED	FIXED	1	1	1	1
429	CLOSED	FIXED	2	3	2	2
435	RESOLVED	FIXED	1	1	1	1
436	CLOSED	FIXED	1	1	1	1
447	CLOSED	FIXED	1	1	1	1
454	RESOLVED	FIXED	1	1	1	1
455	RESOLVED	FIXED	1	2	1	2
465	CLOSED	FIXED	1	2	1	2
492	CLOSED	FIXED	1	1	1	1
493	CLOSED	FIXED	1	1	1	1
494	RESOLVED	FIXED	1	1	1	1
495	RESOLVED	FIXED	1	1	1	1
496	RESOLVED	FIXED	1	1	1	1

497	RESOLVED	FIXED	2	1	2	1
499	RESOLVED	FIXED	2	1	2	1
501	RESOLVED	FIXED	1	3	1	1
502	CLOSED	FIXED	1	1	1	1
503	RESOLVED	FIXED	2	1	2	1
508	RESOLVED	FIXED	3	1	2	1
509	CLOSED	FIXED	1	1	1	1
516	CLOSED	FIXED	1	1	1	1
520	CLOSED	FIXED	1	2	1	2
523	CLOSED	FIXED	2	3	2	3
530	CLOSED	FIXED	2	3	2	3
533	CLOSED	FIXED	1	1	1	1
538	CLOSED	FIXED	1	1	1	1
540	CLOSED	FIXED	1	1	1	1
543	CLOSED	FIXED	1	1	1	1
544	CLOSED	FIXED	1	1	1	1
552	CLOSED	FIXED	1	1	1	1
555	CLOSED	FIXED	1	1	1	1
556	CLOSED	FIXED	1	1	1	1
557	CLOSED	FIXED	1	1	1	1
559	CLOSED	FIXED	1	1	1	1
563	CLOSED	FIXED	2	3	2	3
565	CLOSED	FIXED	1	1	1	1
567	CLOSED	FIXED	1	1	1	1
568	CLOSED	FIXED	1	1	1	1
572	CLOSED	FIXED	1	1	1	1

573	CLOSED	FIXED	1	3	1	3
576	CLOSED	FIXED	1	1	1	1
579	CLOSED	FIXED	1	1	1	1
584	CLOSED	FIXED	1	1	1	1
586	CLOSED	FIXED	2	3	2	3
588	CLOSED	FIXED	1	1	1	1
589	CLOSED	FIXED	1	1	1	1
595	CLOSED	FIXED	2	1	2	1
602	CLOSED	FIXED	2	2	2	2
617	CLOSED	FIXED	1	1	1	1
669	CLOSED	FIXED	1	2	1	2
671	CLOSED	FIXED	1	2	1	2
682	CLOSED	FIXED	1	2	1	2
689	RESOLVED	FIXED	1	1	1	1
736	CLOSED	FIXED	1	2	1	2
737	CLOSED	FIXED	1	1	1	1
741	CLOSED	FIXED	1	3	1	3
762	CLOSED	FIXED	2	1	2	1
767	RESOLVED	FIXED	1	3	1	3
770	CLOSED	FIXED	1	1	1	1
772	RESOLVED	FIXED	1	1	1	1
774	RESOLVED	FIXED	1	1	1	1
775	RESOLVED	FIXED	2	3	2	3
779	RESOLVED	FIXED	1	1	1	1

seca 101

Fehlerverteilung der Fehlerquellen des Experten 1

Klassifizierung	Fehler-Quelle	absolute Fehleranzahl	relative Fehleranzahl
1	Implementierung	89	0,65
2	Anforderungen	34	0,25
3	Andere	14	0,10

Fehlerverteilung der Fehlersenken des Experten 1

Klassifizierung	Fehler-Senke	absolute Fehleranzahl	relative Fehleranzahl
1	Systemtest	83	0,61
2	Release	6	0,04
3	Andere	48	0,35

Fehlerquellen und –senken für einzelne Fehler

ID	Status	Resolution	Fehler-Quelle, Experte 1	Fehler-Senke, Experte 1	Fehler-Quelle, Experte 2	Fehler-Senke, Experte 2
44	RESOLVED	FIXED	1	3	1	3
45	RESOLVED	FIXED	2	3	2	3
46	RESOLVED	FIXED	1	3	1	3
47	RESOLVED	FIXED	1	1	1	1
58	RESOLVED	FIXED	1	1	1	1
59	RESOLVED	FIXED	2	3	2	3
70	RESOLVED	FIXED	1	1	1	1
71	RESOLVED	FIXED	3	1	1	1
72	RESOLVED	FIXED	2	3	2	3

73	RESOLVED	FIXED	1	1	1	1
74	RESOLVED	FIXED	2	3	2	2
76	RESOLVED	FIXED	3	1	3	1
81	RESOLVED	FIXED	1	1	1	1
82	RESOLVED	FIXED	1	1	1	1
83	RESOLVED	FIXED	1	3	1	3
92	RESOLVED	FIXED	2	3	2	3
95	RESOLVED	FIXED	2	3	2	3
103	RESOLVED	FIXED	1	1	1	1
104	CLOSED	FIXED	1	1	1	1
105	RESOLVED	FIXED	1	1	1	1
106	RESOLVED	FIXED	2	3	2	1
122	RESOLVED	FIXED	1	1	1	1
123	RESOLVED	FIXED	2	1	2	1
124	RESOLVED	FIXED	1	1	1	1
129	RESOLVED	FIXED	2	1	2	1
130	VERIFIED	FIXED	2	1	2	1
131	VERIFIED	FIXED	1	1	1	1
132	VERIFIED	FIXED	1	1	1	1
134	VERIFIED	FIXED	1	1	1	1
135	VERIFIED	FIXED	1	1	1	1
138	VERIFIED	FIXED	1	1	1	1
139	VERIFIED	FIXED	1	1	1	1
141	VERIFIED	FIXED	2	3	2	3
143	RESOLVED	FIXED	1	3	1	3
149	RESOLVED	FIXED	1	1	1	1

151	RESOLVED	FIXED	1	1	1	1
155	RESOLVED	FIXED	1	1	1	1
156	RESOLVED	FIXED	1	1	1	1
160	RESOLVED	FIXED	1	1	1	1
170	RESOLVED	FIXED	1	1	3	1
171	RESOLVED	FIXED	1	1	1	1
174	RESOLVED	FIXED	1	1	1	1
175	RESOLVED	FIXED	1	1	1	1
176	RESOLVED	FIXED	1	1	1	1
178	RESOLVED	FIXED	1	1	1	1
179	RESOLVED	FIXED	3	1	1	1
182	RESOLVED	FIXED	1	1	1	1
183	RESOLVED	FIXED	1	1	1	1
184	RESOLVED	FIXED	2	3	2	3
185	RESOLVED	FIXED	1	1	1	1
186	RESOLVED	FIXED	3	1	3	1
187	VERIFIED	FIXED	3	3	3	3
188	RESOLVED	FIXED	1	1	1	1
189	RESOLVED	FIXED	3	1	3	1
190	RESOLVED	FIXED	1	3	1	3
191	RESOLVED	FIXED	1	3	1	3
192	RESOLVED	FIXED	1	1	1	1
194	VERIFIED	FIXED	1	1	1	1
195	VERIFIED	FIXED	1	1	1	1
196	VERIFIED	FIXED	2	3	2	3
199	VERIFIED	FIXED	1	1	1	1

200	VERIFIED	FIXED	2	3	2	3
201	VERIFIED	FIXED	1	1	1	1
202	VERIFIED	FIXED	1	1	1	1
203	RESOLVED	FIXED	1	1	1	1
206	RESOLVED	FIXED	3	3	3	1
208	VERIFIED	FIXED	1	1	1	1
209	RESOLVED	FIXED	1	1	1	1
211	RESOLVED	FIXED	2	3	2	3
213	VERIFIED	FIXED	1	3	1	3
217	VERIFIED	FIXED	2	3	2	3
219	RESOLVED	FIXED	1	1	1	1
221	RESOLVED	FIXED	1	3	1	3
223	RESOLVED	FIXED	2	1	2	1
225	RESOLVED	FIXED	2	3	2	3
227	RESOLVED	FIXED	1	3	1	3
228	RESOLVED	FIXED	2	3	2	3
229	RESOLVED	FIXED	3	3	3	3
231	RESOLVED	FIXED	1	1	1	1
234	RESOLVED	FIXED	2	3	2	3
255	RESOLVED	FIXED	3	3	3	3
259	RESOLVED	FIXED	1	3	1	3
266	RESOLVED	FIXED	1	3	1	3
267	RESOLVED	FIXED	2	3	3	3
270	RESOLVED	FIXED	2	3	2	3
273	RESOLVED	FIXED	1	3	1	3
279	RESOLVED	FIXED	1	1	1	1

280	RESOLVED	FIXED	1	1	1	1
281	RESOLVED	FIXED	1	1	1	1
282	RESOLVED	FIXED	2	2	2	2
288	RESOLVED	FIXED	3	3	3	3
289	RESOLVED	FIXED	1	1	1	1
292	RESOLVED	FIXED	2	2	2	2
297	RESOLVED	FIXED	1	1	1	1
298	RESOLVED	FIXED	1	1	1	1
299	RESOLVED	FIXED	1	1	1	1
308	RESOLVED	FIXED	1	2	1	2
312	RESOLVED	FIXED	1	2	1	2
334	RESOLVED	FIXED	1	1	1	1
335	RESOLVED	FIXED	2	3	2	3
337	RESOLVED	FIXED	2	3	2	3
338	RESOLVED	FIXED	1	1	1	1
339	RESOLVED	FIXED	1	1	1	1
340	RESOLVED	FIXED	2	3	2	3
343	RESOLVED	FIXED	1	2	1	2
344	RESOLVED	FIXED	1	1	1	1
345	RESOLVED	FIXED	3	3	3	3
346	RESOLVED	FIXED	1	1	1	1
347	RESOLVED	FIXED	1	1	1	1
348	RESOLVED	FIXED	2	1	2	1
365	RESOLVED	FIXED	2	2	2	2
367	RESOLVED	FIXED	2	3	2	3
371	RESOLVED	FIXED	1	1	1	1

384	RESOLVED	FIXED	2	3	2	3
390	RESOLVED	FIXED	3	1	3	1
400	RESOLVED	FIXED	1	1	1	1
403	RESOLVED	FIXED	1	1	1	1
404	RESOLVED	FIXED	3	1	3	1
406	RESOLVED	FIXED	1	1	1	1
407	RESOLVED	FIXED	1	3	1	3
420	RESOLVED	FIXED	1	1	1	1
438	RESOLVED	FIXED	1	1	1	1
452	RESOLVED	FIXED	2	1	2	1
479	RESOLVED	FIXED	1	1	1	1
486	RESOLVED	FIXED	1	1	1	1
487	RESOLVED	FIXED	3	3	3	3
490	RESOLVED	FIXED	1	3	1	3
504	RESOLVED	FIXED	1	3	1	3
512	RESOLVED	FIXED	1	1	1	1
527	RESOLVED	FIXED	2	3	2	3
534	RESOLVED	FIXED	2	3	2	3
536	RESOLVED	FIXED	1	1	1	1
569	RESOLVED	FIXED	3	3	3	3
570	RESOLVED	FIXED	1	1	1	1
571	RESOLVED	FIXED	1	1	1	1
581	RESOLVED	FIXED	1	1	1	1
606	RESOLVED	FIXED	1	3	1	3

seca 515/514

Fehlerverteilung der Fehlerquellen des Experten 1

Klassifizierung	Fehler-Quelle	absolute Fehleranzahl	relative Fehleranzahl
1	Implementierung	15	0,65
2	Anforderungen	6	0,26
3	Andere	2	0,09

Fehlerverteilung der Fehlersenken des Experten 1

Klassifizierung	Fehler-Senke	absolute Fehleranzahl	relative Fehleranzahl
1	Systemtest	10	0,44
2	Release	7	0,30
3	Andere	6	0,26

Fehlerquellen und -senken für einzelne Fehler, sowie Fehlerzuordnung zu einzelnen Testbereichen (Kapitel 5).

ID	Status	Resolution	Fehler-Quelle, Experte 1	Fehler-Senke 1, Experte 1	Fehler-Quelle 2, Experte 2	Fehler-Senke 2, Experte 2	Fehler-Zuordnung zu Testbereichen
285	CLOSED	FIXED	6	0	6	0	2
333	CLOSED	FIXED	6	0	6	0	2
428	RESOLVED	FIXED	6	0	6	0	2
521	CLOSED	FIXED	5	0	5	0	2
522	CLOSED	FIXED	4	3	4	3	3
553	CLOSED	FIXED	4	3	4	3	3
577	CLOSED	FIXED	4	3	4	3	3
598	CLOSED	FIXED	4	3	4	3	3
605	CLOSED	FIXED	4	3	4	3	3
615	RESOLVED	FIXED	4	3	4	3	2

654	CLOSED	FIXED	4	3	4	3	1
655	CLOSED	FIXED	6	3	6	3	2
657	CLOSED	FIXED	4	3	4	3	2
659	CLOSED	FIXED	4	4	4	4	1
672	CLOSED	FIXED	4	4	4	4	1
681	CLOSED	FIXED	4	4	4	4	1
728	CLOSED	FIXED	12	0	12	0	3
730	CLOSED	FIXED	4	4	4	4	3
751	CLOSED	FIXED	4	4	4	4	2
753	RESOLVED	FIXED	4	4	4	4	2
759	CLOSED	FIXED	6	0	6	0	2
764	CLOSED	FIXED	4	3	4	3	1
833	CLOSED	FIXED	6	4	6	4	3

Verteilung der Fehlerzuordnung, bei seca 515/514 (Kapitel 5)

Testbereiche	absolute Fehleranzahl	relative Fehleranzahl
1	5	0,22
2	10	0,43
3	8	0,35

A5 DDE-Berechnungen

seca 101:

DDE (system)=61% (system) / (4% (release)+61%) = 94%

DDE (QS)=61% (system)+35% (Andere) / 100% = 96%

seca 115:

DDE (system)=65% (system) / (18% (release)+65%) = 78%

DDE (QS)=61% (system)+17% (Andere) / 100% = 78%

seca 515/514:

DDE (system)=44% (system) / (30% (release)+44%) = 59%

DDE (QS)=44% (system)+26% (Andere) / 100% = 70%

A6 Test Complete

TestComplete ist ein Testautomatisierungstool, welches von SmartBear zur Unterstützung des Softwaretests entwickelt wurde. Mit dem Tool können Testfälle auf verschiedene Art und Weisen aufgezeichnet bzw. erstellt und später automatisch ausgeführt werden. Die Ergebnisse werden in einem Log-File protokolliert.

Das Tool unterstützt eine Vielzahl von Anwendungen (z.B. Qt- und .NET-basierte) und kann auf verschiedenen Betriebssystemen ausgeführt werden. Außerdem können mit Hilfe von Test Complete verschiedene Testarten, wie z.B. GUI-Tests und Regressionstests, umgesetzt werden.

In diesem Kapitel sollen die wichtigsten Features von Test Complete, sowie deren Vorteile bzw. Nachteile erläutert werden.

Testaufzeichnung

Test Complete verfügt über einen Test Recorder mit dem alle Aktionen, die ein Tester während eines manuellen Tests ausführt, aufgezeichnet werden können. Die Aktionen können in einem Keyword-Test- oder einem Advanced-Modus erfasst werden. Keyword-Test bietet eine übersichtliche Darstellung aller Aktionen, die intuitiv nachvollzogen werden kann.

Die aufgenommenen Aktionen können im Advanced Modus auch direkt in einer Skriptsprache (z.B. Visual Basic oder C++) dargestellt werden. Für einen Tester mit Programmiererfahrung kann diese Darstellung übersichtlicher sein.

Die Tests, die ausschließlich mit dem Test Recorder aufgezeichnet werden sind für eine anspruchsvolle automatisierte Testausführung meistens nicht geeignet. Sie sind linear aufgebaut und erlauben deshalb keine Fallunterscheidungen. Außerdem beinhalten sie lediglich konstante Eingabewerte, was eine mehrfache Testausführung des Testsablaufs mit verschiedenen Eingabewerten unmöglich macht. Die aufgezeichneten Tests lediglich ein Grundgerüst fest und müssen in nachfolgenden Schritten erweitert und überarbeitet werden.

Hierzu stellt Test Complete zahlreiche Funktionen (z.B. if-Anweisungen) zur Verfügung, die an den gewünschten Stellen in den Testverlauf eingebaut werden können. Einige der Funktionen, wie z.B. die Texterkennung, können nur in dem Advanced Modus, also in der Skriptform, aufgerufen werden. Es ist aber möglich diese Funktionen in Form einer „Script Routine“ zum Keyword-Test hinzuzufügen.

Grundsätzlich bietet Test Complete auch die Möglichkeit auf den Test Recorder ganz zu verzichten und alle Aktionen in dem Advanced Modus, also direkt in einem Testskript, manuell zu erstellen. Das ist vor allem dann interessant, wenn die zu testende Anwendung noch nicht lauffähig ist und somit der Test Recorder nicht eingesetzt werden kann.

Aufzeichnungsarten

Bei einem manuellen Test erkennt und bedient der Tester die GUI-Elemente. Bei einem automatisierten Durchlauf müssen diese Aufgaben vom Tool übernommen werden. Test Complete bietet hierfür zwei unterschiedliche Aufzeichnungsmodi an. Die Aktionen können koordinaten- und/oder objektorientiert aufgenommen und ausgeführt werden.

Bei der objektorientierten Aufzeichnung beziehen sich alle Aktionen (z.B. Tastatureingaben und Mausklicks) auf das aktive Fenster und die darin befindlichen Objekte (z.B. Buttons). Die Position der Objekte relativ zum Fenster oder zum Bildschirm spielt dabei keine Rolle. Allerdings kann Test Complete diesen Aufzeichnungsmodi nur dann einsetzen, wenn die Windows- bzw. Qt-Objekte innerhalb des aktiven Fensters erkannt werden. Die Objekte können anhand ihrer individueller Eigenschaften erkannt werden. Einige dieser Eigenschaften verändern sich zur Laufzeit und können deshalb nicht verwendet werden. Andere wiederum sind unveränderliche Eigenschaften, wie z.B. der Beschriftungstext, dessen Verwendung meistens ausreichend ist, um die Objekte eindeutig zu identifizieren.

Falls die Objekte nicht erkannt werden, können die Aktionen von Test Complete bezogen auf die Koordinaten des aktiven Fensters aufgezeichnet werden. Der koordinatenorientierte Aufzeichnungsmodi kann auch explizit ausgewählt werden.

Beide Aufzeichnungsmodi haben ihre bevorzugten Einsatzgebiete. Die koordinatenorientierte Aufzeichnung eignet sich besonders dann, wenn nicht auf die Steuerelemente bzw. Objekte der GUI zugegriffen werden kann. Damit lässt sich im Prinzip jede Applikation, unabhängig von der Ausführungsplattform bzw. der eingesetzten Entwicklungstechnologie testen. Die einzige Voraussetzung hierfür ist eine vorhandene GUI. Der große Nachteil ist allerdings die mangelnde Robustheit und Wartbarkeit dieser Tests. Da sich die Position von Objekten innerhalb der GUI in Abhängigkeit von dem Betriebssystem, Auflösung oder den durchgeführten Modifikationen (z.B. im Layout) verändern kann, sind die Tests nach einer Veränderung häufig unbrauchbar und müssen neu aufgenommen werden. Bei objektorientiert aufgezeichneten Tests hat eine Verschiebung oder Veränderung von Objekten weniger gravierende Folgen, denn diese sind von der Position der Objekte unabhängig.

Überprüfung der Ausgaben

Bei der Testautomatisierung sollen nicht nur die einzelnen Testschritte automatisch ausgeführt werden. Die erzeugten Ausgaben sollen ebenfalls automatisch überprüft werden. Hierfür bietet Test Complete die Möglichkeit an der gewünschten Stelle im Test sogenannte Checkpoints zu setzen. Die Checkpoints können verschiedene Eigenschaften der Ausgabe mit dem

gewünschten Ergebnis vergleichen. Am häufigsten werden die *Region Checkpoints* und die *Property Checkpoints* bzw. *Object Checkpoints* eingesetzt.

Der Region Checkpoint erlaubt es GUI-Bereiche innerhalb der Anwendung zu prüfen, die nicht als Objekte identifiziert werden konnten. Dazu zählen z.B. Diagramme und Schaubilder. Die Property Checkpoints bzw. Object Checkpoints greifen auf die GUI-Objekte zu und überprüfen entweder einzelne oder mehrere Eigenschaften des zu testenden GUI-Objektes. Auf diese Weise können z.B. Textinhalte überprüft werden, sofern diese in den Eigenschaften angezeigt werden. Test Complete bietet außerdem die Möglichkeit benutzerdefinierte Checkpoints zu setzen, die z.B. eine if-Anweisung überprüfen können.

Wenn die erzeugten Ausgaben von Soll-Ergebnisse, die in dem Checkpoint definiert sind, abweichen, wird es im Testlog protokoliert. Falls keine Abweichung festgestellt werden konnten, gilt der Test als bestanden.

Texterkennung

Wenn auf die Objekte nicht zugegriffen werden kann, der Textinnhalt (z.B. Werte oder Meldungen) dieser allerdings überprüft werden soll, können Region Checkpoints zur Bilderkennung eingesetzt werden. Ein großer Nachteil dabei ist, dass bei mehreren Checkpoints, wie sie z.B. bei wiederholter Testausführung benötigt werden, auch die Anzahl der Region Checkpoints, also der Bildausschnitte, steigt. Das bietet eine geringe Robustheit und benötigt viel Speicher, sowie einen hohen Wartungsaufwand, vor allem wenn sich die GUI verändert.

Test Complete bietet die Möglichkeit mit einer Texterkennung zu arbeiten. Die Texte werden dabei aus dem vordefinierten Bildbereich ausgelesen und überprüft. So kann auf die Bilderkennung verzichtet werden, was vor allem bei wiederholter Testausführung vom Vorteil ist. Außerdem ist es möglich auf der Basis von ausgelesenen Texten Fallunterscheidungen in den Testskript einzubauen und somit diese flexibel zu gestalten.

Datengetriebener Test

Test Complete unterstützt datengetriebenes Testen und bietet damit die Möglichkeit denselben Test mit verschiedenen Datensätzen mehrmals ablaufen zu lassen. Dazu werden die Testschritte, die wiederholt werden sollen, in einem *Data-Driven Loop* erfasst. Diese Funktion kann sowohl bei wiederholten Eingaben von leicht abgeänderten Testdaten, z.B. Tastatureingaben, als auch bei der wiederholten Überprüfung der Ausgaben eingesetzt werden.

Bei dem Data-Driven Loop liegen die Testdaten und der Testablauf getrennt vor. Der große Vorteil dabei ist, dass bei Modifikationen an der Software der Testablauf und die separat angelegten Testdaten leichter gepflegt bzw. angepasst werden können.

Die Eingabe- und Referenzdaten, die sich in Abhängigkeit von den Eingabedaten verändern, werden für den Data-Driven Loop parametrisiert und in einer Tabellenform intern oder extern (z.B. in Excel) abgelegt. Diese Daten werden erst während des Durchlaufs dynamisch in den Test eingebunden. Die Spaltennamen in der Tabelle repräsentieren die Namen der Parameter, die in dem Data-Driven Loop verwendet werden. Die einzelnen Zeilen der jeweiligen Spalte beinhalten Daten, die bei jedem Testdurchlauf als Eingabe- bzw. Referenzdaten eingebunden werden.

A7 Testfälle für Fettmasse-Normbereich

Test case - ID	**Patient data**	**Expected Result**
Ethnic: caucasian		
TST_1_01	age: 30 years gender: male ethnic group: caucasian weight: 90 kg height: 190 cm	The measurements results of seca embedded software and the reference-data (see Excel-Data table "Berechnungen") should be equal. FM-Bargraph-normal ranges: **8**% und **21**%
TST_1_02	age: 50 years gender: male ethnic group: caucasian weight: 90 kg height: 190 cm	The measurements results of seca embedded software and the reference-data (see Excel-Data table "Berechnungen") should be equal. FM-Bargraph-normal ranges: **11**% und **23**%
TST_1_03	age: 70 years gender: male ethnic group: caucasian weight: 90 kg height: 190 cm	The measurements results of seca embedded software and the reference-data (see Excel-Data table "Berechnungen") should be equal. FM-Bargraph-normal ranges: **13**% und **25**%
TST_1_04	age: 19 years gender: male ethnic group: caucasian weight: 90 kg height: 190 cm	The measurements results of seca embedded software and the reference-data (see Excel-Data table "Berechnungen") should be equal. FM-Bargraph-normal ranges: **No bargraph shown**
TST_1_05	age: 80 years gender: male ethnic group: caucasian weight: 90 kg height: 190 cm	The measurements results of seca embedded software and the reference-data (see Excel-Data table "Berechnungen") should be equal. FM-Bargraph-normal ranges: **No bargraph shown**
TST_1_06	age: 30 years gender: female ethnic group: caucasian weight: 90 kg height: 190 cm	The measurements results of seca embedded software and the reference-data (see Excel-Data table "Berechnungen") should be equal. FM-Bargraph-normal ranges: **21**% und **33**%
TST_1_07	age: 50 years gender: female ethnic group: caucasian weight: 90 kg height: 190 cm	The measurements results of seca embedded software and the reference-data (see Excel-Data table "Berechnungen") should be equal. FM-Bargraph-normal ranges: **23**% und **35**%

TST_5_06	age: 80 years gender: female ethnic group: others weight: 90 kg height: 190 cm	The measurements results of seca embedded software and the reference-data (see Excel-Data table "Berechnungen") should be equal. FM-Bargraph-normal ranges: **No bargraph shown**

A8 Referenzdaten aus der Excel-Tabelle „Berechnungen“

x	y	v	w	medPar	TST-1-01	TST-1-02	TST-1-03	TST-1-04
Development								
367	233	54	27	Gewicht	90,00	90,00	90,00	90,00
366	317	56	20	Größe	1,900	1,900	1,900	1,900
377	399	43	20	BMI	24,9	24,9	24,9	24,9
Energy								
368	235	61	21	FM-kg	18,75	20,13	21,51	17,99
393	260	26	20	FM-%	21	22	24	20
341	319	98	21	E-MJ	1.043,11	1.091,45	1.139,84	1.016,69
372	346	118	17	E-kcal	249.308	260.864	272.429	242.996
378	401	42	22	REE-MJ/D	7,78	7,54	7,30	7,91
386	428	81	17	REE-kcal/D	1.860	1.803	1.746	1.891
599	655	79	21	TEE-MJ/D	11,67	11,10	10,96	11,87
386	510	82	17	TEE-kcal/D	2.790	2.704	2.619	2.837
180	371	21	18	FM-Klein	8	11	13	
361	372	29	16	FM_Gross	21	23	25	
FunkReha								
368	237	80	22	FFM-kg	71,25	69,87	68,49	72,01
368	319	82	20	FM-kg	18,75	20,13	21,51	19,99
393	343	29	21	FM-%	21	22	24	20
388	399	34	22	FMI	5,2	5,6	6,0	5,0
380	424	62	23	FFMI	19,7	19,4	19,0	19,9
368	483	81	22	SMM	36,09	35,41	34,76	36,44
378	495	42	22	LST llA	2,83	2,70	2,58	2,89
368	515	42	22	LST reA	3,18	3,07	2,96	3,24
371	542	78	22	LST llB	13,20	13,19	13,20	13,19
390	570	80	20	LST reB	11,86	11,86	11,86	11,86
Fluid								