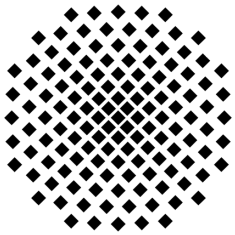
UNIVERSITÄT STUTTGART

FAKULTÄT FÜR INFORMATIK UND ELEKTROTECHNIK
Institut für Softwaretechnologie, Abteilung Software Engineering



Fachstudie

Lösungen für die Testfallverwaltung

Simon Brodtmann
Ralf Ebert
Tim Schmidt
13. Januar 2011

Betreuer
Markus Knauß
Holger Röder

Inhaltsverzeichnis

1 Einleitung

1.1 Zustandekommen

Diese Fachstudie wurde in Zusammenarbeit der Abteilung Softwareengineering des Instituts für Softwaretechnologie der Universität Stuttgart mit der Daimler Fleetboard GmbH initiiert. Die Aufgabenstellung wurde von Volker Werner, als Vertreter der Daimler FleetBoard GmbH und Holger Röder und Markus Knauß von der Abteilung Softwareengineering gemeinsam erarbeitet. Die Aufgabenstellung wurde uns am 23.06.2010 vorgelegt.

1.2 FleetBoard [1]

Die Daimler FleetBoard GmbH wurde im Jahr 2003 als 100%ige Tochter der Daimler AG gegründet und vereint langjährige Erfahrungen aus der LKW-Branche und Informationstechnologie-Know-how unter einem Dach. Das DEKRA-zertifizierte Unternehmen stattete seit Markteinführung der FleetBoard Dienste im Jahr 2000 über 55.000 Lkw bei mehr als 1.200 Speditionen aus (Stand: 12/2009).

Die FleetBoard Zentrale befindet sich in Stuttgart-Vaihingen. Für den Einbau der FleetBoard Hardware sorgt das Einbauteam in der unternehmenseigenen Halle auf dem Wörther Werksgelände oder beim Kunden vor Ort.

Weltweit beschäftigt FleetBoard derzeit mehr als 140 Mitarbeiter (Stand: 12/2009), deren oberste Priorität auf der Weiterentwicklung zukunftsträchtiger Lösungen für das Alltagsgeschäft von Transportunternehmen und Logistikern liegt.

1.3 Aufgabenstellung

Aufgabe der Fachstudie ist die Analyse und Bewertung der bestehenden Test-Werkzeuge und -Prozesse beim Industriepartner und die Konzeption eines Lösungsansatzes zur Optimierung der Testfallverwaltung. Dabei soll insbesondere die Umsetzbarkeit des Lösungsansatzes in einem etablierten, industriellen Umfeld berücksichtigt werden.

1.4 Aufbau dieses Berichts

Die Vorbereitung unseres Lösungsvorschlags besteht aus drei Arbeitsschritten: Analyse des Ist-Zustands, Ermittlung von Schwachstellen und Erhebung von Anforderungen. Die zu untersuchenden Bereiche sind die Organisation, der Testprozess und die eingesetzten Werkzeuge.

1. Quelle http://www.fleetboard.com/info/de/unternehmensportrait.html (19.11.2010)

Unsere Fachstudie hat sich an den Arbeitsschritten orientiert, den Endbericht haben wir zur Verbesserung der Übersicht aber in die untersuchten Bereiche unterteilt, welche jeweils die drei Arbeitsschritte enthalten. Zu dem Testprozess und den Werkzeugen haben wir jeweils noch die Meinung der befragten Mitarbeiter zusammengefasst.

Anhand der erhobenen Anforderungen formulieren wir anschließend eine Ideallösung, die einen Großteil der Anforderungen aber nicht die entstehenden Kosten beachtet. Zusammen mit einer subjektiven Gewichtung der Anforderungen bieten wir anschließend einen wirtschaftlich umsetzbaren Lösungsvorschlag.

Ein Begriffslexikon, der verwendete Fragenkatalog für die Interviews, Beispiele aus ClearQuest, sowie die vollständige Priorisierung der Anforderungen ergänzen dieses Dokument.

1.5 Vorgehensweise und Quellen

Wir hatten während der gesamten Bearbeitungszeit der Fachstudie uneingeschränkten Zugang zu den Mitarbeitern und den von ihnen genutzten und hier vorgestellten Werkzeugen mit Test- und Produktivdaten, sowie den firmeninternen Wikis erhalten, die zusammen die Grundlage unserer Recherchen bilden.

Für die Analyse haben wir einen Fragebogen entworfen und insgesamt zehn Mitarbeiter aus verschiedenen Abteilungen interviewt. Grundlage für die Wahl der verschiedenen Mitarbeiter sind die verschiedenen Testarten, die es bei FleetBoard gibt. So wurden beispielsweise Mitarbeiter aus den Teams für Hardwaretests, Betriebstests und Serverschnittstellentests befragt (siehe Abbildung 1.1).

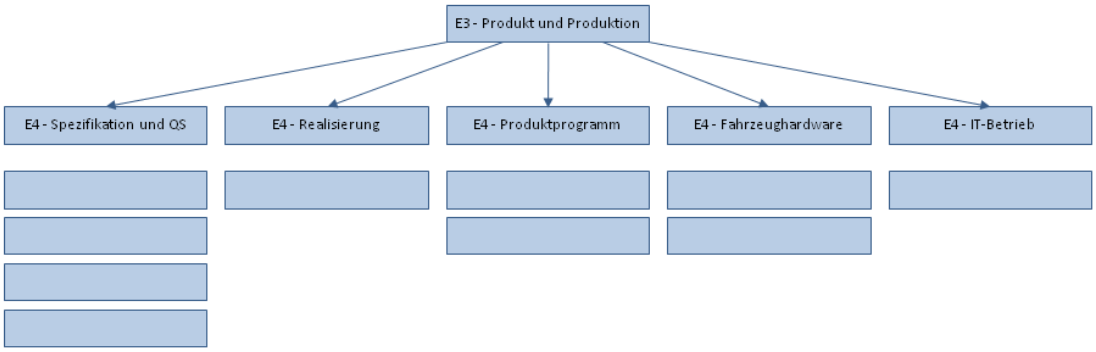


Abbildung 1.1: Interviewte Mitarbeiter (anonymisiert)

2 Organisation

2.1 Analyse

Bei FleetBoard findet sich in der Abteilung Produkt und Produktion eine klassische Matrixorganisation, die sich vertikal aus der Daimler-Linienorganisation zusammensetzt und horizontal aus mehreren Scrum-Teams.

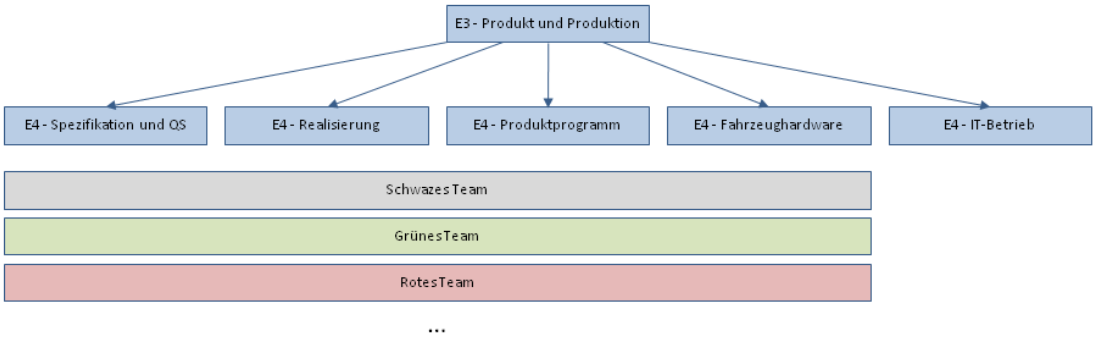


Abbildung 2.1: Organisationsstruktur

2.1.1 Linienorganisation

Die Linienorganisation untersteht dem Abteilungsleiter „Produkt und Produktion“ im Daimler-Rang „E3“. Unter ihm befinden sich fünf Teamleiter der Ebene „E4“, die die Teams „Spezifikation und Qualitätssicherung“, „IT-Betrieb“, „Produktprogramm“, „Fahrzeughardware“ und „Realisierung“ leiten.

Spezifikation und Qualitätssicherung

Das Team „Spezifikation und Qualitätssicherung“, das auch Aufgabensteller der Fachstudie ist, besteht aus sieben Mitgliedern. Es beschäftigt sich mit der Spezifikation von Anforderungen, die vom Produktteam geliefert werden, und mit der Qualitätssicherung der Ergebnisse der Realisierung. Dabei führen sie vor allem Integrationstests durch.

Produktprogramm

Das Team „Produktprogramm“ dient als Vermittler zwischen den Kunden, dem Vertrieb und der Entwicklung. Ihre Aufgabe ist dabei die Sammlung, Formalisierung und Bündelung von Anforderungen der Kunden. Zusätzlich werden – bisher noch nicht formalisiert oder mit einem Prozess – Anforderungen aus Sicht des Kunden getestet. Da das Produktteam untypischerweise

der Produktionsabteilung zugeordnet ist, sind einige Mitglieder auch als Projektleiter in der Entwicklung beschäftigt.

Betrieb

Das Team IT-Betrieb ist für die FleetBoard-Rechnersysteme zuständig. Dazu gehören nicht nur die Arbeitsplatzrechner der Mitarbeiter, sondern auch die Server und Datenbanken der Testsysteme und des Produktivsystems. Die Aufgabe der Mitarbeiter dieses Teams ist die Installation und Wartung dieser Systeme, wozu auch das Deployment neuer Releases mit anschließendem Test der Serverinstallation gehört.

Fahrzeughardware

Zum Gesamtprodukt FleetBoard gehören auch Geräte, die in die Fahrzeuge des Kunden verbaut werden. Das ist zum einen die „Telematikplattform“, ein Gerät, das an den CAN-Bus des Fahrzeuges angeschlossen ist und Fahrzeugdaten an die FleetBoard-Zentrale sendet, und zum anderen der „Dispo-Pilot“, ein Gerät für den Fahrer zur Abwicklung von Transportaufträgen, zur Navigation und Kommunikation. Da diese Geräte und die Software von Zulieferern entwickelt werden, ist das Hardware-Team für die Qualitätssicherung dieser Geräte zuständig, wozu beispielsweise auch Testfahrten gehören.

Realisierung

Das größte der E4-Teams der Produktabteilung besteht aus Softwareentwicklern, die die Anforderungen umsetzen und Server- und Clientsoftware entwickeln.

2.1.2 Projektorganisation

Quer zur Linienorganisation sind die Mitarbeiter in sechs Teams aufgeteilt, die verschiedene Aufgabenbereiche haben und sich aus verschieden vielen Mitgliedern unterschiedlicher Verteilung der Linienteams zusammensetzen. Diese Teams verwenden einen an Scrum angelehnten Prozess. Diese Teams sind nach den Farben Rot, Grün, Gelb, Orange, Blau und Schwarz benannt, wobei das schwarze Team für Integrationstests und Qualitätssicherung zuständig ist. Die „farbigen“ Teams spezifizieren und realisieren Anforderungen, die ihnen vom Team „Produktprogramm“ geliefert werden. Dazu gehört auch die Implementierung von automatisierten Testskripts. Dabei sind einzelnen Teams verschiedene Aufgabenbereiche der FleetBoard-Welt zugeteilt. So kümmert sich ein Team beispielsweise um die serverseitigen Themen, während sich andere Teams um die Cliententwicklung kümmern.
Das schwarze Team nimmt eine Sonderrolle ein. Es setzt sich größtenteils aus Mitgliedern des Linienteams „Spezifikation und Qualitätssicherung“ zusammen, während die farbigen Teams (sie werden meist einfach „Scrum-Teams“ genannt), großteils aus Mitgliedern des Teams „Realisierung“ bestehen. Das schwarze Team befasst sich mit Qualitätssicherung auf höherer Ebene. Das sind insbesondere Testplanung, Testaufwandsplanung (Zuweisen von Testfällen an Tester) und Integrationstests. Nebenher kümmert es sich ständig um die Weiterentwicklung und Verbesserung der Testinfrastruktur. Beispielsweise mit Hilfe dieser Fachstudie.
In den farbigen Entwicklungsteams arbeiten zusätzlich zu den Realisierern je ein Mitglied des Teams „Spezifikation und Qualitätssicherung“ als Spezifikateur und Tester, sowie ein Mitglied des Teams „Produktprogramm“ als Product-Owner und Projektleiter.

2.2 Schwachstellen

Anfangs machte die Organisationsstruktur einen soliden Eindruck, denn es gibt immerhin explizit ein Team für die Qualitätssicherung und delegierte Mitarbeiter in jedem Entwicklerteam.

Im Laufe der Analyse wurde aber klar, dass die Qualitätssicherung nicht die Stellung hat, die sie eigentlich braucht.

Es werden Prozesse erarbeitet und definiert. Tools werden ausgesucht und eingeführt. Dokumentation wird angefangen. Die Analyse ergab, dass aber nichts von all dem wirklich bis zum Ende durchdacht und umgesetzt wurde.

(SS 2.1) Prozesse sind nur implizit und teilweise weiß auch der betroffene Mitarbeiter nicht Bescheid. Tools werden zwar verwendet, aber nicht wie vorgesehen. Datenstrukturen entstehen zufällig und wachsen eher, statt definiert zu werden. Dokumentation fehlt oft komplett. Wenn es sie doch gibt, dann meist unvollständig oder mit einem falschen Fokus.

(SS 2.2) Es gab bereits eine Umstellung dahingehend, dass ein Entwickler nun nicht mehr selbst die Testfälle zu seinem Code schreiben soll. Sprich, es gibt full-time Tester, die den Code der Entwickler testen sollen. Allerdings reicht die Zahl der Tester bei weitem nicht aus.

2.3 Anforderungen

2.1 Probleme auf organisatorischer Ebene

- Ursache: Prozesse sind nicht gut genug dokumentiert. Richtlinien werden nicht eingehalten, oft gibt es aber auch keine.
- Anforderung: Es muss eine Person geben, die sich den Aufgaben gemäß LL07, Kap. 13.1.3 widmet. Die gibt es grundsätzlich zwar schon, aber die Zeit zur Umsetzung ist nicht festgelegt und die Aufgaben sind nicht klar definiert.

2.2 Zu wenig Tester

- Ursache: Das Wachstum bei FleetBoard beschränkt sich auf die Einstellung von Entwicklern. Die Zahl der Tester wächst nicht mit.
- Anforderung: Es müssen mehr Mitarbeiter für die Tests reserviert werden, damit die Entwickler nicht mehr testen müssen. Werden zusätzliche Entwickler eingestellt, so muss auch sichergestellt sein, dass es einen Tester gibt, der Zeit hat dessen Code zu testen.

2.4 Bestehendes

Die Matrixorganisation scheint gut zu funktionieren. Sie sollte daher so bleiben. Zwar ist es schwierig, die Strukturen bei FleetBoard zu verstehen, allerdings ist das nur ein Dokumentationsproblem.

Sehr gut ist, dass das Bewusstsein für Qualitätssicherung und Tests bei FleetBoard stark ausgeprägt ist. Das sollte erhalten bleiben. Die meiste Kritik von uns bewegt sich daher auf einem sehr hohen Niveau. Lobenswert ist, dass es zwei Mitarbeiter für die Entwicklung der Testinfrastruktur gibt.

3 Prozess

3.1 Analyse

Um einen Überblick über den Testprozess mit den eingesetzten Dokumenten, den entstehenden Artefakten und den involvierten Personen zu liefern, folgt ein Diagramm (siehe Abbildung 3.1).

Entwickelt wird iterativ mit einem „Release“ als Grundlage für einen Zyklus. Ein Release enthält mehrere Entwicklungssprints. Während ein Release ca. alle sechs Monate fertiggestellt wird, dauert ein Sprint genau einen Monat. Innerhalb der Entwicklungssprints wird meist nur die neue Funktionalität getestet. In zwei Sprints wird ausschließlich die komplette Funktionalität der FleetBoard Soft- und Hardware getestet.

Ein Sprint läuft wie folgt ab:

Nachdem die Testfälle spezifiziert sind, werden sie je nach Testtyp und Automatisierungsgrad mit unterschiedlichen Werkzeugen implementiert. Das Produktmanagement gibt die geplanten Features für das nächste Release mit dem Werkzeug Focal Point an die Entwicklerteams. Eine Anforderung in Focal Point heißt ebenfalls „Focal Point“.

Ein Mitarbeiter aus dem Entwicklerteam, welches für eine Anforderung zuständig ist, erstellt aus dem Focal Point (meist mit zusätzlichen Rückfragen) die Spezifikation und legt diese in RequisitePro ab. Diese Spezifikation dient als Grundlage für alle Entwickler sowie für das schwarze Team.

Die Spezifikation in RequisitePro dient der Entwicklung selbst und der Spezifikation der Testfälle in ClearQuest als Grundlage. Das wird teilweise von den Entwicklern selbst gemacht (so ist es vom schwarzen Team auch gedacht), teilweise schreibt aber auch das schwarze Team die Testfallspezifikationen. Manchmal werden die Testfallspezifikationen nur „Spezifikation“ genannt, wodurch die Bezeichnungen in der Praxis schwammig werden.

Am Ende eines Sprints werden die Tests vom schwarzen Team ausgeführt. Im letzten Sprint eines Releases wird ausschließlich getestet. Einen Überblick für die Planung und Durchführung der Tests soll die Restaufwandsschätzung, eine selbst geschriebene Weboberfläche (siehe 4.2.2 ITT - Testmanagement), bieten. Diese arbeitet mit den Daten von ClearQuest und bereitet diese grafisch auf.

Gefundene Fehler landen in einem Bugzilla und werden am Ende des Tests von einem Buggremium besprochen und priorisiert. Anhand dieser Informationen wird ein Release freigegeben oder nicht.

Grundsätzlich wird bei FleetBoard ein Test als „erfolgreich“ bezeichnet, wenn er durchgelaufen ist ohne einen Fehler zu finden. Es wird sehr viel Wert auf „grüne“ Testergebnisse gelegt.

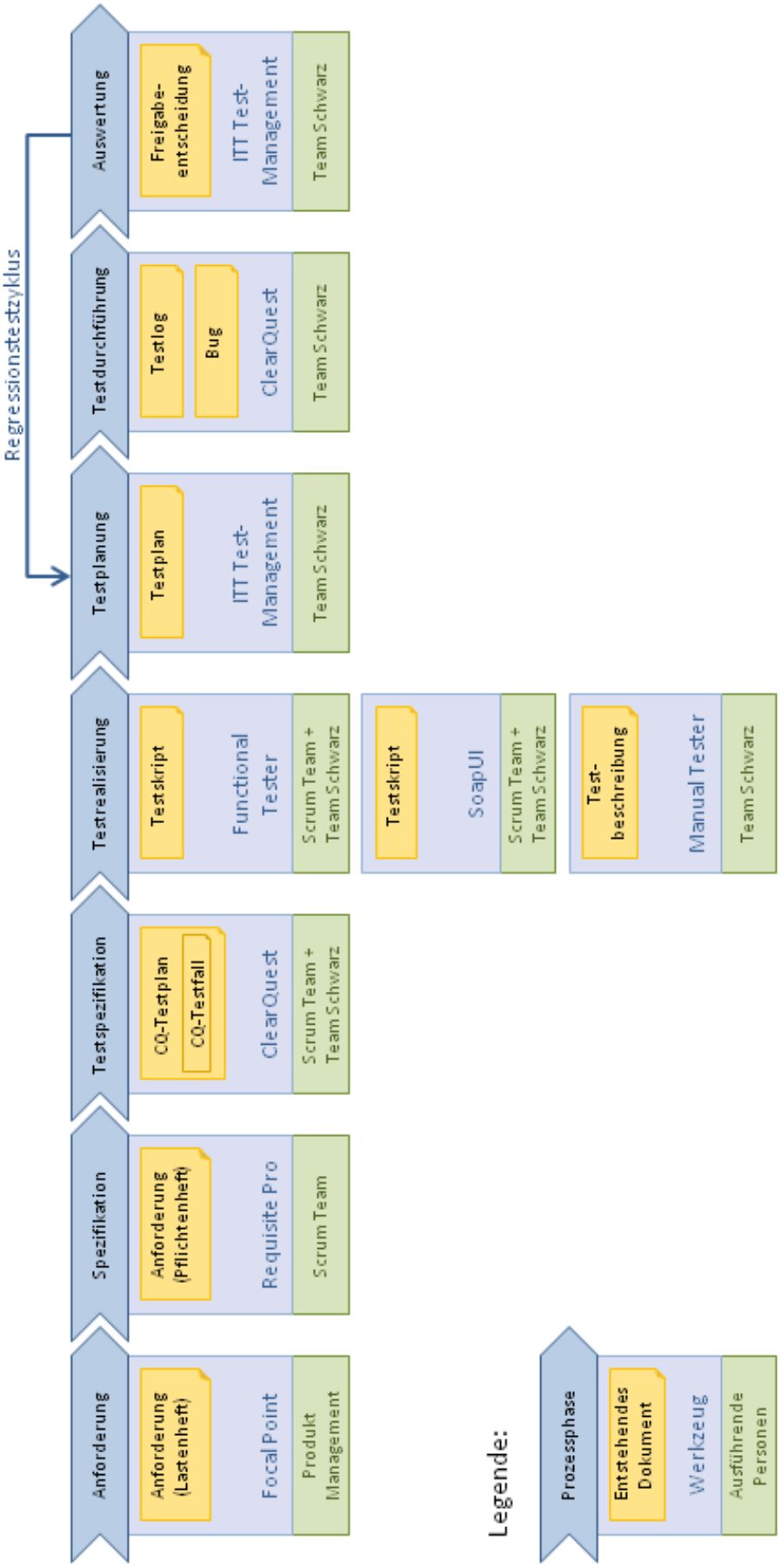


Abbildung 3.1: Der Testprozess bei FleetBoard

Anforderungsverfolgung

Um verfolgen zu können, welche Testergebnisse die Implementierung einer Anforderung hat, wurde bei FleetBoard eine Anforderungsverfolgung eingeführt. Diese besteht aus Verweisen der verschiedenen Artefakte aus verschiedenen Tools im Spezifikations- und Testprozess untereinander. So werden durchgehende Verbindungen von der Anforderung in RequisitePro über Testfall und Testskript zum Testergebnis in der ClearQuest-Datenbank hergestellt. Diese Verweise sind teilweise programmatisch nachverfolgbar realisiert, also über Referenzen in Datenbankfeldern, oder sie sind nur über textuelle Verweise vorhanden, die sich nicht automatisch auswerten lassen. Abbildung 3.2 stellt diese durchaus komplexen Zusammenhänge dar. Eine gestrichelte Linie in der Grafik stellt eine „weiche“, nicht automatisch auswertbare Verbindung dar, während eine durchgehende Linie eine auswertbare Verbindung darstellt. Eine gepunktete Linie repräsentiert keine tatsächliche Verbindung, sondern zeigt nur, wie die verschiedenen Artefakte zusammengehören.

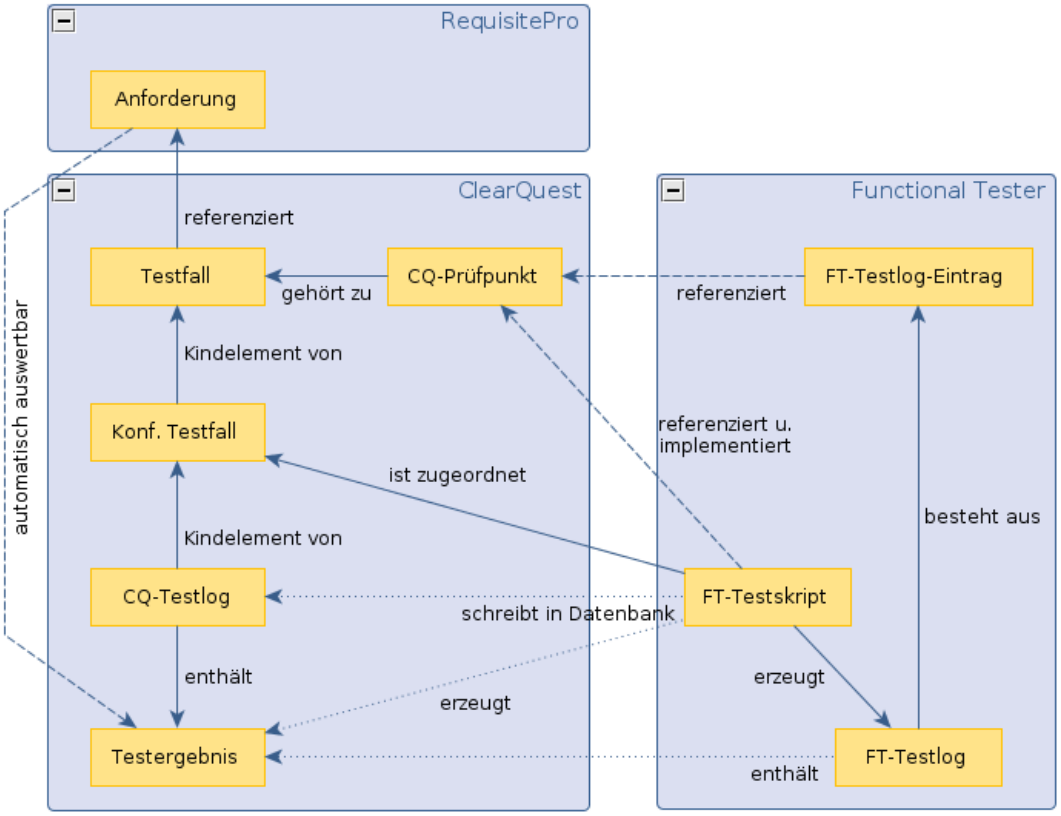


Abbildung 3.2: Anforderungsverfolgung bei FleetBoard

3.1.1 Dokumentation

Die beste Dokumentation des Testprozesses bei FleetBoard konnten wir im Wiki des schwarzen Teams finden. Dort wird schrittweise das Vorgehen von der Testplanung bis zum Testabschluss beschrieben. Allerdings stimmt der dokumentierte Ablauf nicht mit dem realen überein.

Der Prozess laut Wiki sieht wie folgt aus:

Die zeitliche und personelle Einplanung der Testaktivitäten wird im Testplan von IBM Rational ClearQuest festgehalten.
Nach der Testplanung folgt die Testanalyse und das Testdesign. Die Umsetzung erfolgt mit IBM Rational ClearQuest durch das Anlegen und Spezifizieren von Testfällen. Die Realisie-

rung erfolgt durch Umsetzung der spezifizierten Testfälle in Testskripte mit Hilfe von Testtools. Die GUI Tests werden mit Functional Tester umgesetzt.

Bei der Durchführung werden die geplanten Testfälle in definierter Reihenfolge ausgeführt und protokolliert. Die gefundenen Fehler werden in das Fehlermanagementsystem eingetragen und an den zuständigen Entwickler weitergereicht.

Die Testauswertung soll den aktuellen Qualitätsstatus sichtbar machen. Testprotokolle und Meldungen der gefundenen Fehler werden dabei ausgewertet. Dieser Qualitätsstatus ermöglicht die Steuerung im Sprint. Der Testabschluss wird am Ende eines Sprints im Sprint Review Meeting vorgestellt, wo der aktuelle Qualitätsstatus ausgewertet und berichtet wird.

Ein Problem ist, dass den Mitarbeitern nicht bekannt ist, dass dieser Prozess nachlesbar ist. Dazu kommt, dass Fehler in der Dokumentation enthalten sind. So werden unserer Analyse zufolge die zeitliche und personelle Planung nicht im Testplan, sondern im konfigurierten Testfall vorgenommen (siehe 4.1.1 ClearQuest).

Zur Veranschaulichung des Prozesses ist im Wiki Abbildung 3.3 zu finden. Diese Grafik ist von sehr geringer Qualität, ist nicht mehr aktuell und belegt den aktuellen Zustand der Dokumentation.

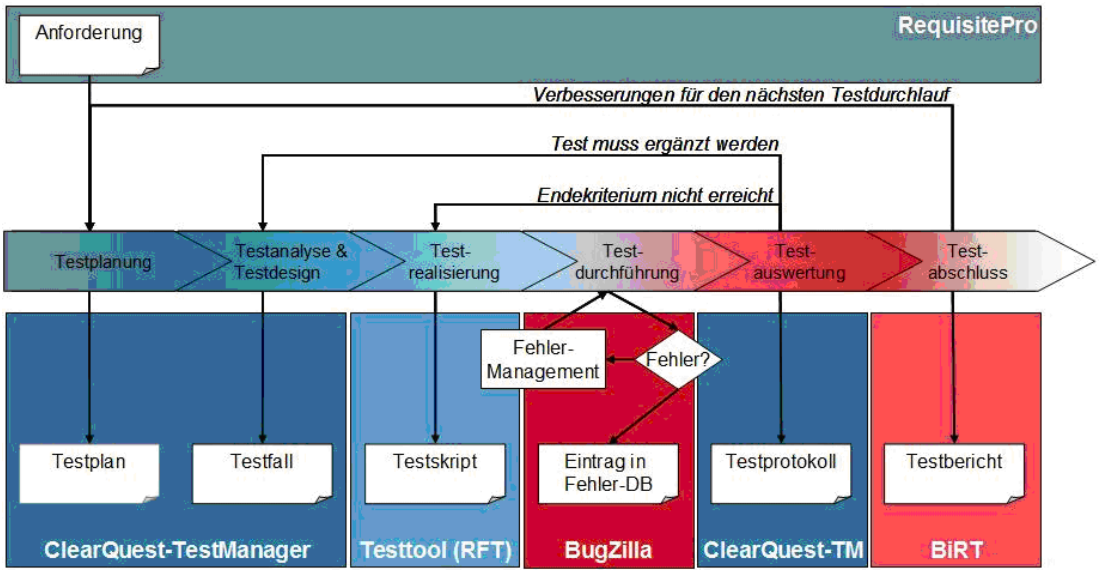


Abbildung 3.3: Grafik zum Testprozess im Wiki des Integrationstestteams

3.2 Bewertung durch die Mitarbeiter

Einen kompletten Überblick über den Entwicklungsprozess von den Anforderungen bis zum Releasetest hat von allen Befragten, wenn überhaupt, nur Volker Werner. Es gibt ein paar Beschreibungen in diversen Wikis, diese sind aber schlecht auffindbar und bieten keine Übersicht über das ganze Konzept. Wenigstens die Mitglieder des Integrationstestteams müssten den Prozess sehr gut kennen. Bei den meisten Mitarbeitern ist der Prozess nur implizit durch Einlernen und Nachfragen angekommen.

Der Testprozess ist nicht allen Mitarbeitern klar. Sie wissen teilweise nicht, an welcher Stelle sie sich im Prozess befinden und wirken teilweise unmotiviert, da sie nicht wissen, was mit ihrer Arbeit geschieht. Ihnen ist nicht klar, wer die Ergebnisse des Testens erhält und ob diese überhaupt angesehen werden.

3.3 Schwachstellen

Jedes der eingesetzten Werkzeuge bei FleetBoard deckt einen Bereich des Testprozesses ab. Um die Werkzeuge in einen Kontext zu bringen und die Verknüpfungen untereinander verstehen zu können, brauchen wir einen Überblick über den Testprozess. Viele der Probleme, die für den einzelnen Entwickler augenscheinlich bei einem konkreten Werkzeug liegen, sind aber eher im Prozess begründet, was erst durch unsere ganzheitliche Analyse aufgedeckt werden konnte.

3.3.1 Testprozess allgemein

(SS 3.1) Der Prozess wird aufgrund der Tatsache, dass die Mitarbeiter ihn nur mündlich überliefert bekommen, nicht reproduzierbar durchgeführt. Zuständigkeiten sind nicht klar. So werden manchmal Testfallspezifikationen von den Entwicklern selbst geschrieben, in anderen Fällen werden diese jedoch vom schwarzen Team geschrieben. Eine schriftlich festgehaltene Linie wer was spezifiziert existiert nicht.

(SS 3.2) Es gibt pro Entwicklungsteam nur einen Spezifizierer und Tester. Da dieser nicht alle Tests alleine durchführen kann, helfen die Entwickler bei der Testdurchführung aus. Das bedeutet, dass Entwickler ihren eigenen Code testen. Ihnen fehlt aber nicht nur Objektivität, sondern auch die nötige Motivation.

(SS 3.3) Die Kommunikation zwischen dem schwarzen Team und den Entwicklungsteams wird durch verschiedene Versionsbezeichnungen der FleetBoard-Software erschwert. Entwickler wissen nicht, auf welche Version sich eine Bug-Meldung bezieht und verschwenden so Zeit bei der Suche nach dem Fehler.

(SS 3.4) Erwähnt werden sollte noch, dass mehrere Mitarbeiter, insbesondere bei den Hardwaretests, sagten, zur Testautomatisierung hätten sie keine Zeit. Man hat also keine Zeit, Zeit zu sparen. Es sollte also überlegt werden, ob nicht ein fest eingeplantes Zeitbudget zur Automatisierung von Testfällen angebracht wäre, was jeden weiteren Sprint zu mehr Einsparungen führen würde. Dass dem so ist, weiß man von der Automatisierung bei der Software.

(SS 3.5) Vor allem die Übergänge zwischen den Phasen des Testprozesses bereiten Probleme. Sie sind oft unzureichend definiert und Artefakte haben nicht die Qualität, die für den nächsten Arbeitsschritt erforderlich ist. Viel mündliches Nachfragen ist notwendig.

(SS 3.6) Generell sollte Redundanz so gut wie möglich vermieden werden. Uns ist aufgefallen, dass viele Daten bei FleetBoard redundant sind. Das wird meist technisch begründet. Bei Dokumentation ist es meist fehlende Zusammenarbeit und fehlende Korrekturen.

(SS 3.7) FleetBoard hat eine umfassende, für das Unternehmen spezifische, Terminologie. Das bereitet oft Probleme wenn unterschiedliche Teams zusammenarbeiten müssen. Abkürzungen werden selbstverständlich verwendet, die das andere Team nicht kennt. Begriffe werden eingeführt aber nirgends definiert.

(SS 3.8) Tester können sich oft nicht voll auf ihre Arbeit konzentrieren, sondern müssen sich mit architektonischen Besonderheiten rumärgern. Das Starten von SSH-Verbindungen oder Testservern wie z.B. bei soapUI sollte so gut es geht durch die Infrastruktur abgenommen oder wenigstens vereinfacht werden.

3.3.2 Dokumentation

(SS 3.1) Ein Problem des Prozesses besteht darin, dass er im Wesentlichen mündlich verbreitet wird. Als Folge daraus ergibt sich, dass die Motivation mancher Mitarbeiter etwas gedämpft ist, da sie nicht wissen an welcher Stelle im Prozess sie sich befinden und wo die Ergebnisse ihrer Arbeit landen. Sie würden es begrüßen, wenn der Prozess schriftlich festgehalten wäre.

Eine ausführliche Dokumentation zum Entwicklungsprozess ist bei FleetBoard nirgendwo zu finden. Es existiert lediglich eine Wikiseite (im IT Realisierungs-Wiki), auf der erwähnt wird, dass bei FleetBoard nach dem Scrum Prozess entwickelt wird. Dort wird auf externe Seiten zu Scrum verlinkt. Die Mitarbeiter wissen zwar ungefähr über die Abläufe, die sie direkt betreffen, Bescheid, jedoch fehlt ihnen der Überblick.

(SS 3.9) Zum Testprozess gibt es zwar eine Dokumentation im Wiki des schwarzen Teams, jedoch ist diese weder vollständig, noch entspricht sie durchgehend der Realität. Es gibt eine Schritt-für-Schritt-Anleitung von der zeitlichen und personellen Einplanung bis zur Realisierung mit Functional Tester. Diese ist jedoch sehr grob und nicht ausreichend für die Dokumentation eines Prozesses.

(SS 3.10) An der Dokumentation der einzelnen Teams ist zu bemängeln, dass diese nicht immer für alle zugänglich ist. Außerdem ist das Wiki-Konzept fleetboardweit weder einheitlich noch überhaupt geregelt. Verschiedene Teams verwenden verschiedene (insgesamt vier) Wikis, die verschiedene Zugangsdaten benötigen und verschiedene Wiki-Software einsetzen.

3.3.3 Kundenakzeptanztests

(SS 3.11) Eine Sonderstellung in unserer Fachstudie haben die Kundenakzeptanztests, denn es gibt sie bisher noch nicht explizit. Sie wurden bereits vor geraumer Zeit vom schwarzen Team angedacht, sind also geplant, umgesetzt wurde der Plan allerdings nie. Der momentane Stand ist, dass das Produktmanagement komplett manuell und undokumentiert die fertige Anwendung, oder schon vor Releaseende fertige Teilergebnisse, grob kontrolliert. Testfälle und Einschätzung des Ergebnisses liegen im Ermessen des jeweiligen Produktmanagers.

3.3.4 Anforderungsverfolgung (siehe Abbildung 3.2)

(SS 3.12) Die Nachverfolgbarkeit der Ergebnisse von Prüfpunkten ist automatisiert nicht möglich. Prüfpunkte existieren zwar in der ClearQuest-Datenbank und werden in den Logs von Functional Tester referenziert, allerdings lässt sich keine programmatische Verbindung einem Prüfpunkt und seinen Testergebnissen herstellen. Auch ist in ClearQuest nicht vorgesehen, dass ein Prüfpunkt überhaupt ein Testergebnis haben kann. Die Ergebnisverfolgbarkeit ist somit nur auf Testfall-Ebene – und damit nur für eine Gruppe von Lehrbuch-Testfällen – gegeben.

Nach der Testausführung wird zwar das Testergebnis in der ClearQuest-Datenbank gespeichert, nicht aber das gesamte Testlog. Wenn ein Fehler aufgetreten ist, dann ist es im Nachhinein nicht mehr möglich, mit Hilfe der Datenbank den genauen Fehler zu ermitteln.

3.3.5 Manuelle Tests

(SS 3.13) Wie sich in den Interviews herausgestellt hat, gibt es für die manuellen Tests in dem Team für die Telematikplattform mehrere Schichten von Testfällen. In ClearQuest gibt es jeweils nur eine grobe Beschreibung des Testfalls ohne genauere Angaben und ohne Erwähnung von Teilautomatisierungen durch ausführbare Testprogramme. Manual Tester enthält alle Einzelschritte ohne Semantik und ebenfalls ohne Erläuterung der ausführbaren Testprogramme. Als Drittes gibt es die Testprogramme, die erst im Quellcode ihre Vorgehensweise und den Nutzen preisgeben. Dadurch werden die Testfälle sehr unübersichtlich. Niemand blickt komplett durch.

3.4 Anforderungen

3.4.1 Allgemein

3.1 Fehlender Überblick über den Testprozess

- Ursache: Es gibt keine aktuelle und gute Dokumentation dazu
- Anforderung: Es sollte eine Dokumentation geben, die den neuen und bestehenden Mitarbeitern einen Überblick über den Testprozess vermittelt. Rollen und Artefakte müssen dabei bereits eingeführt werden.

3.2 Entwickler testen ihren eigenen Code

- Ursache: Die Zeit zum Testen ist vorhanden, weil es gemacht werden muss, aber sie wird auf die Entwickler aufgeteilt, anstatt full-time Tester dafür einzusetzen.
- Anforderung: Die benötigte Zeit abschätzen und ausreichend Mitarbeiter für die Testspezifikation und -automatisierung zur Verfügung stellen.

3.3 Kommunikationsprobleme bei verschiedenen Versionen der eigenen Software

- Ursache: Es gibt keine einheitliche Versionsbezeichnung der Software- und Hardwarekomponenten bei FleetBoard zwischen den verschiedenen Teams.
- Anforderung: Versionsbezeichnungen aller Produkte aus dem Hause FleetBoard müssen vereinheitlicht sein.
- Verweis: Anf. 2.1

3.4 Zu wenig Zeit zur Automatisierung von Tests

- Ursache: Obwohl offensichtlich ist, dass dies ein Henne-Ei-Problem ist (würde man automatisieren, dann hätte man anschließend auch Zeit dafür), fehlt es an der Zeit zur Implementierung der Automatisierung.
- Anforderung: Es muss zusätzliche Zeit zur Automatisierung von Tests zur Verfügung gestellt werden. Am besten je Sprint eine fest definierte Menge.

3.5 Problematische Übergänge zwischen den Phasen des Testprozesses

- Ursache: Es fehlt an Dokumentation und Richtlinien für Prozessübergänge. Eine Kontrolle gibt es nicht.
- Anforderung: Es muss eine Dokumentation und Richtlinien für Prozessübergänge geben. Die Richtlinien müssen kontrolliert werden.

3.6 Vermeidung von Redundanz

- Ursache: Redundanz ist bei FleetBoard recht präsent, verursacht durch technische Gegebenheiten, fehlende Zusammenarbeit und fehlende Korrekturen von Dokumentation.
- Anforderung: Allgemein sollte dieser Aspekt bei architektonischen Veränderungen stärker beachtet werden. Im Nachhinnein ist es schwierig bis unmöglich dieses Problem zu beheben.

3.7 Terminologie

- Ursache: Nicht definierte Begriffe und Abkürzungen führen zu Kommunikationsschwierigkeiten.
- Anforderung: Es sollte ein FleetBoard-weites Begriffslexikon angelegt werden. Auf die Verwendung von Abkürzungen sollte möglichst verzichtet werden.

3.8 Verbesserung der Testinfrastruktur

- Ursache: Obwohl die Testinfrastruktur bereits sehr gut ist, so lässt sich im Detail noch Ärger bei der Arbeit vermeiden.
- Anforderung: Umständliches Starten von SSH-Verbindungen und Testservern sollte so gut wie möglich vermieden werden, sodass Tester sich auf die eigentliche Arbeit konzentrieren können.

3.4.2 Dokumentation

3.9 Unzureichende Qualität der Dokumentation

- Ursache: Die Dokumentation zum Testprozess ist unfertig, ungepflegt und hält keinen einheitlichen Qualitätsstandard ein.
- Anforderung: Neben der Übersicht über den Testprozess (siehe Abbildung 3.1), bedarf es einer qualitativ hochwertigen Dokumentation, die ausreichend ins Detail geht um die tägliche Arbeit erledigen zu können, sich aber trotzdem auf das Wesentliche konzentriert.

3.10 Nicht einheitliche Verwaltung der verschiedenen Dokumentationen

- Ursache: Jedes Team hat sein eigenes Wiki, teilweise nicht einsehbar für die anderen Teams. Das führte bisher schon einmal zu Reibung und machte es schwierig sich in anderen Wikis zurecht zu finden.
- Anforderung: Teamübergreifende Dokumentationsregeln. Eine Sorte Wiki, einheitlich aufgebaut, zugänglich für alle Mitarbeiter.

3.4.3 Kundenakzeptanztests

3.11 Einführung der Kundenakzeptanztests

- Ursache: Kundenakzeptanztests sind angedacht, wurden aber nie eingeführt.
- Anforderung: Dies ist eine Anforderung von FleetBoard selbst. Unsere Anforderung dazu ist, dass die Kundenakzeptanztests erfolgreich eingeführt werden.

3.4.4 Anforderungsverfolgung

3.12 Mangelhafte Verknüpfung von Testergebnissen mit den Tests

- Ursache: Es wird lediglich ein Ergebnis zu einem Testfall gespeichert. Auf Prüfpunktebene gibt es keine Zuordnung. Da das erzeugte Testlog nicht gespeichert wird, ist ein manuelles Nachlesen auch nicht möglich.
- Anforderung: Die Testergebnisse müssen exakter werden, indem sie nicht an einen Testfall, sondern an einen Prüfpunkt gehängt werden. Mindestens für fehlgeschlagene (automatisierte) Tests muss das Testlog gespeichert werden um die genaue Ursache nachschauen zu können.

3.4.5 Manuelle Tests

3.13 Manuelle Tests sind zu vielschichtig

- Ursache: Durch die drei Ebenen, in denen die manuellen Tests angelegt werden, werden diese sehr unübersichtlich und die Wartbarkeit leidet.
- Anforderung: Zusammenführung der Ebenen auf eine. ClearQuest ist der sinnvollste Ort um manuelle Tests umfassend und mit allen notwendigen Daten abzuspeichern. Eine Durchführung sollte allein mit den Daten aus ClearQuest möglich sein. Es wird lediglich ein Werkzeug benötigt, was diese Durchführung unterstützt. Der Manual Tester wird dafür eigentlich nicht benötigt.

3.4.6 Bestehendes

Nachdem wir ein aktuelles Diagramm vom Testprozess erstellt hatten, stellten wir fest, dass der Prozess selbst sehr gut durchdacht und auch sinnvoll ist. Alle Schwachstellen zum Prozess betreffen nur die Dokumentation, die Umsetzung und kleine Details. Die Grundstruktur sollte aber bleiben wie sie ist.

4 Werkzeuge

4.1 Analyse

Die Daimler FleetBoard GmbH verwendet für die Testfallverwaltung und -durchführung einige Produkte von IBM Rational, Eviware soapUI und eine teilweise selbst entwickelte Weboberfläche. Diese Werkzeuge untersuchen wir in diesem Kapitel genauer auf Schwachstellen.

4.1.1 ClearQuest

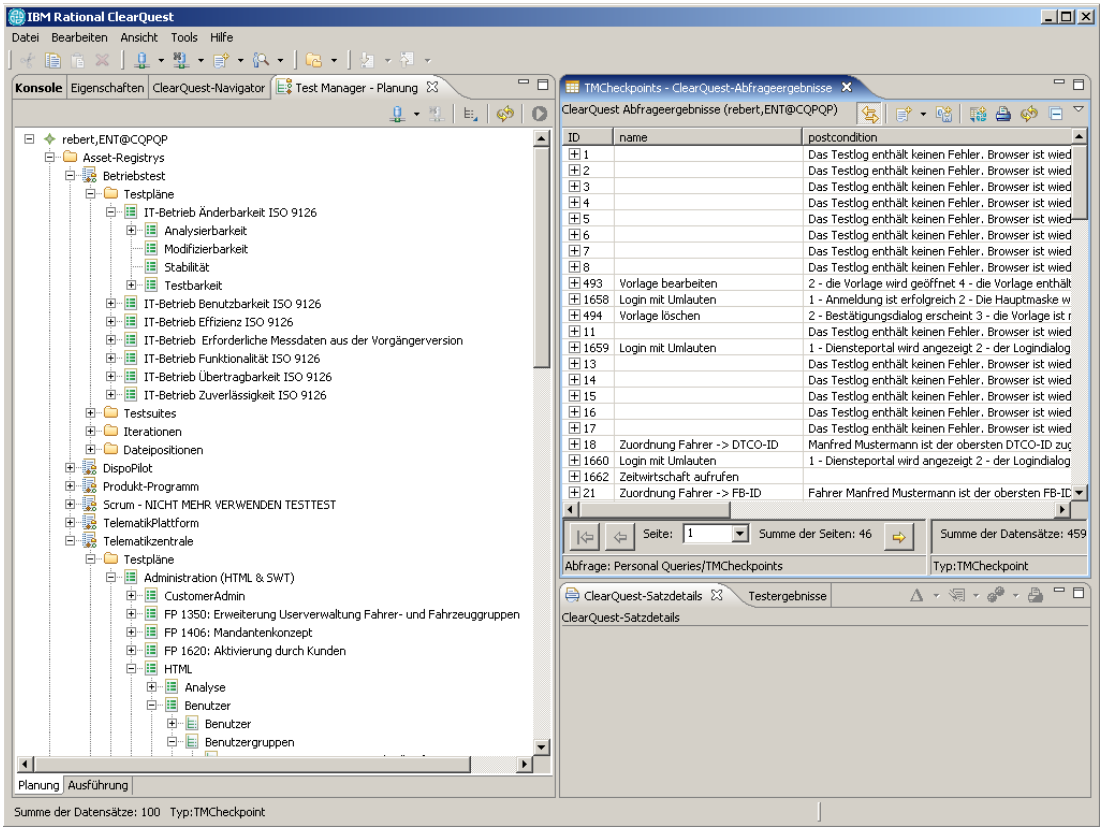


Abbildung 4.1: ClearQuest

ClearQuest ein Änderungsmanagementwerkzeug mit Unterstützung für Bugtracking und Prozessautomatisierung. Bei FleetBoard wird allerdings nur die Komponente „ClearQuest TestManager“ verwendet. Der ClearQuest TestManager ist ein Werkzeug, das die Testplanung und Spezifikation unter einem Hut vereinen soll. Es besitzt eine zentrale Datenbank, in der alle Daten der Testplanung, -spezifikation, des Testfortschritts sowie Testergebnisse gespeichert wer-

den. Jeder Mitarbeiter kann mit seinem ClearQuest-Client auf die zentrale Datenbank zugreifen und Änderungen an ihr vornehmen.

ClearQuest ist ein sehr frei konfigurierbares Werkzeug. Sowohl die Oberfläche, als auch die Datenstruktur sind bis auf wenige Einschränkungen frei änderbar. In neuen Projekten stehen in ClearQuest einige vorkonfigurierte Profile zur Verfügung, aus denen eines ausgesucht werden kann. FleetBoard hat sich bei der ClearQuest-Einführung 2007 für ein sehr umfangreiches Profil entschieden und die darin für den FleetBoard-Prozess fehlenden Dialoge ergänzt. Dabei wurde versäumt, die nicht benötigten Dialoge zu entfernen, wodurch das Programm relativ unübersichtlich wurde.

Zur Datenstruktur: ClearQuest verwendet zur Strukturierung der Testdaten einen Baum. Die höchste Ebene sind sogenannte „Asset-Registries“. Sie sind dazu gedacht, die Tests nach verschiedenen Softwarekomponenten oder komplett verschiedener Software zu unterteilen. Bei FleetBoard wird hier die Unterteilung zwischen Betriebstest, Telematikplattform, Dispopilot usw. vorgenommen.
Den Asset-Registries untergeordnet sind die Testpläne, die lediglich ein Strukturelement darstellen. Die Testpläne können ineinander geschachtelt werden, sodass ein Testplan weitere Testpläne enthalten kann. Beim Betriebstest wird z.B. durch die Testpläne eine Unterteilung in Änderbarkeit, Benutzbarkeit, Effizienz, usw. nach ISO 8126 vorgenommen.

Den Testplänen untergeordnet sind die Testfälle. Dem Testfall sind die zum Testen und zur Testaufwandsplanung wichtigen Daten zugeordnet. Dazu gehören die Testfallspezifikation, die „Prüfpunkte“ und die „konfigurierten Testfälle“. Ein Testfall in ClearQuest entspricht nicht dem Begriff des Testfalls aus dem Lehrbuch [vgl. LL07, Kap. 19.1.1]. Der Lehrbuchtestfall enthält die Vorbedingung, die Aktion und das Sollresultat. Der Testfall in ClearQuest kann durch das Konzept der Prüfpunkte mehrere Lehrbuchtestfälle enthalten.
Ein Prüfpunkt enthält genau die Daten, die der Lehrbuchtestfall vorsieht. Er wurde bei FleetBoard aus technischen Gründen eingeführt, da der Prüfpunkt in Functional Tester bereits existierte und eine Verfolgung des Prüfpunktes zur Spezifikation des Testfalles ohne die Einbindung des Prüfpunktes in ClearQuest nicht oder nur schwer möglich war. In Functional Tester werden zur Reduktion des Programmieraufwandes mehrere Prüfpunkte von einem Testskript abgedeckt. Daher enthalten die Testfälle, die mit Functional Tester automatisiert sind, in der Regel mehrere Prüfpunkte.

Ein „konfigurierter Testfall“ ist mit einem Testfall verknüpft. Ein Testfall kann beliebig viele konfigurierte Testfälle enthalten. In einem konfigurierten Testfall kann festgehalten werden, auf welchem System mit welcher Konfiguration getestet werden soll. So könnte man für verschiedene Rechner mit verschiedenen Betriebssystemen und Hardwarekonfigurationen konfigurierte Testfälle anlegen. Diese könnten dann von einem Rechner gestartet werden und ClearQuest würde den Test auf dem entsprechenden Zielrechner ausführen und die Testlogs in die ClearQuest-Datenbank schreiben. Bei FleetBoard wird diese Funktionalität nur von den Hardware-Teams genutzt. Bei den Software-Teams hat jeder Testfall genau einen konfigurierten Testfall. Dieser muss vorhanden sein, da die Testlogs, der Tester und der geplante Aufwand an konfigurierte Testfälle angehängt werden.

Dokumentation

Die Dokumentation beschränkt sich auf das Wiki des schwarzen Teams. Sie ist auf zwei Seiten verteilt, die untereinander nicht verlinkt sind. Auf der einen Seite, „Tipps rund um ClearQuest“, die unter der Kategorie „Testing Tools“ -> „Anleitungen“ zu finden ist, werden nur die Konfiguration des Programms für den Einstieg in die FleetBoard Entwicklungs-ClearQuest-Datenbank und ein paar technische Details wie zum Beispiel das unlocken einer Datenbank erklärt. Für Fragen rund um das Programm verweist die Wiki Seite auf verschiedene externe Seiten z.B. von IBM. Wie ein neuer Testplan oder -fall angelegt werden kann, wird hier nicht geklärt.

Eine recht oberflächlich gehaltene Beschreibung zur Verwendung von ClearQuest bei FleetBoard ist auf der zweiten Wiki Seite zu finden. Diese Seite liefert für den Einstieg etwas zu knappe Informationen. Hier wird lediglich das Anlegen von Testplänen, jedoch nicht das Anlegen von Testfällen erklärt.

4.1.2 ITT - Testmanagement (auf BiRT basierend)

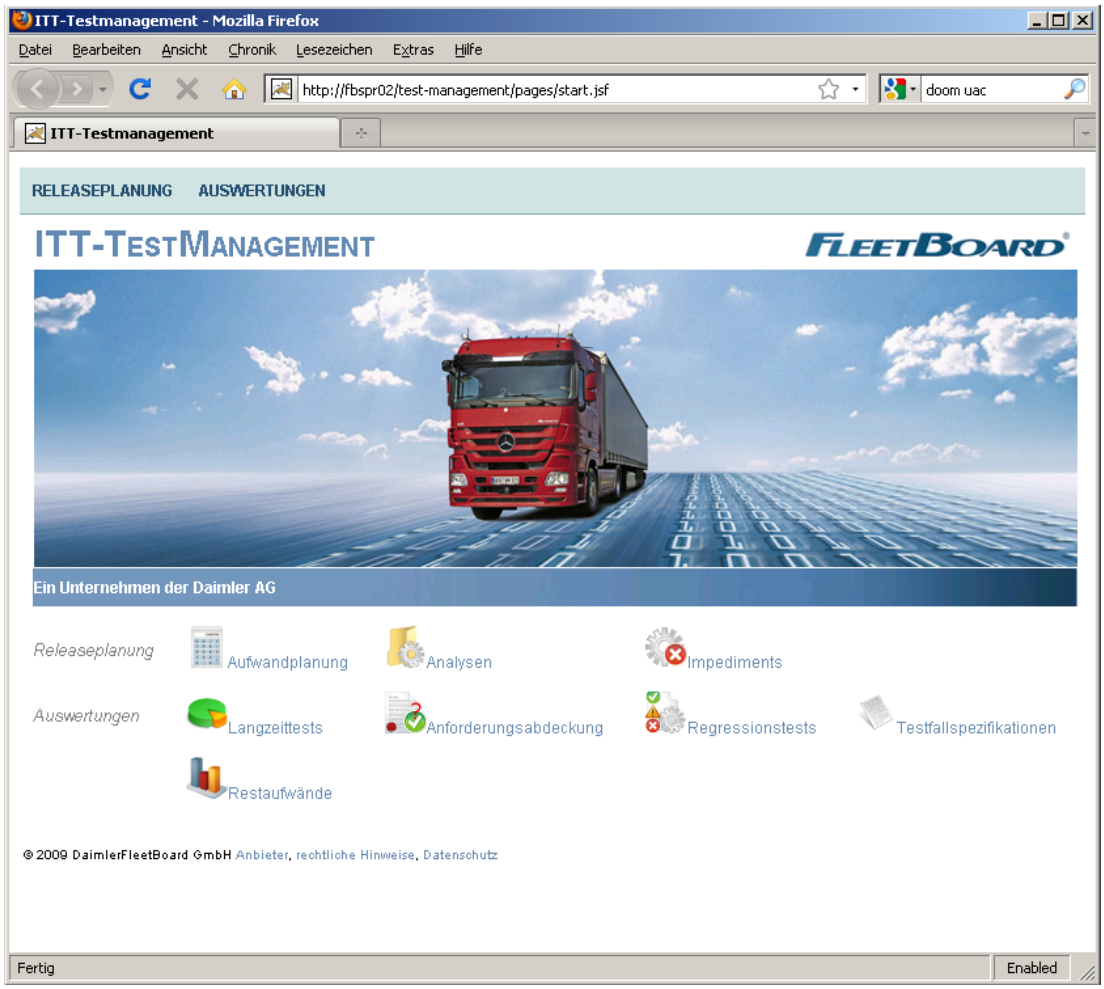


Abbildung 4.2: ITT - Testmanagement

Das ITT-Testmanagement Tool ist eine vom Integrations-Test-Team entwickelte Weboberfläche. Es ist an die Datenbank von ClearQuest angebunden und wird zum Eintragen von geplanten Aufwänden und Restaufwänden verwendet. Zudem können über diese Oberfläche Testergebnisse, die nicht über Functional Tester oder Manual Tester in die ClearQuest Datenbank gelangt sind, von Hand eingetragen werden.

Das Tool hat eine Startseite, auf der es eine klare Trennung zwischen den beiden abgedeckten Bereichen gibt. Zum einen die Releaseplanung und zum anderen die Auswertungen. Diese klare Trennung ist allerdings nur oberflächlich vorhanden. Klickt man sich durch das Tool, gelangt man auf Seiten, die einen anderen Titel haben, als der Link, auf den man geklickt hat. So entsprechen sich z.B. die Ansicht Aufwandsplanung in der Releaseplanung und Restaufwände in den Auswertungen.

Die Releaseplanung bietet nur eine für unsere Fachstudie relevante Ansicht: „Aufwandsplanung“.

In der Aufwandsplanung können konfigurierte Testfälle nach verschiedenen Kriterien gefil-

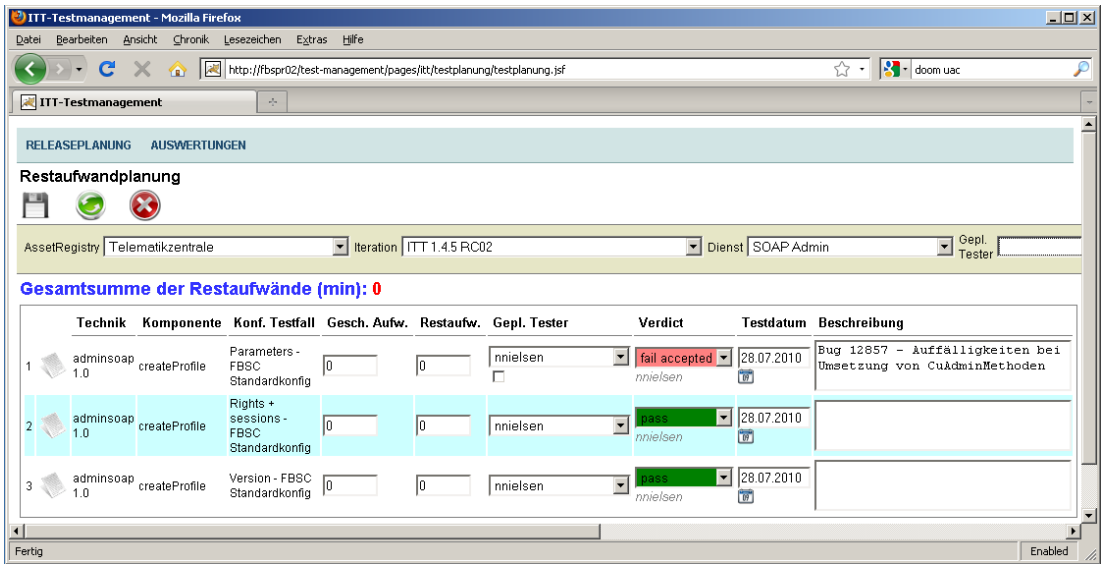


Abbildung 4.3: ITT - Testmanagement

tert angezeigt werden. In dieser Ansicht nimmt einerseits das schwarze Team die Einplanung des Testpersonals vor, andererseits wird die selbe Ansicht von den Testern genutzt, um ihre geplanten Aufwände und den Restaufwand einzutragen. Hier erhalten sowohl die Tester, als auch das schwarze Team einen Überblick über die noch durchzuführenden Testfälle und den Restaufwand, der hierfür noch benötigt wird.

Die Testfallspezifikationsansicht bietet eine einfache Möglichkeit, sich einzelne Testfälle mit den dazugehörigen Prüfpunkten anzeigen zu lassen.
Der Anforderungsabdeckungsbericht liefert eine Statistik zur Gesamtzahl der Anforderungen (aus Focal Point) und denen, die in Form von Testfällen in ClearQuest abgebildet sind.

Die restlichen Ansichten im Bereich Auswertungen sind für die Fachstudie uninteressant und ihr Nutzen erscheint grundsätzlich fragwürdig.

Dokumentation

Es wird im Wiki des schwarzen Teams auf die Seite des BiRT Projektes verwiesen und erklärt, dass das ITT-Testmanagement innerhalb der Auswertung der automatisierten Tests zum Einsatz kommt. Zudem gibt es ein langes PDF als Dokumentation zur Verwendung welches ein sehr niedriges Niveau hat.

4.1.3 Functional Tester

IBM Rational Functional Tester ist ein auf Eclipse aufsetzendes Werkzeug für die Aufnahme, Wiedergabe und Protokollierung von GUI-Tests. Es unterstützt dabei u.a. Windows-, HTML-, Swing- und SWT-Oberflächen. Die Testskripte werden als Java-Programme gespeichert, können nach der automatischen Aufnahme beliebig bearbeitet werden und bieten den vollen Funktionsumfang von Java. Neben der Navigation in Oberflächen bietet Functional Tester umfangreiche Möglichkeiten zum Auslesen von Daten aus der getesteten Oberfläche und zum Abgleich mit Soll-Ergebnissen. Während der Testdurchführung erstellt Functional Tester außerdem ein detailliertes Testlog im HTML-Format.
Eng betrachtet ist Functional Tester also eine API, die das Auslesen von Stuktur- und Inhaltsinformationen aus verschiedenen Benutzeroberflächen ermöglicht. Um Objekte und Inhalte in der zu testenden Software einfacher und zuverlässiger auffinden zu können, werden Objekte in den FleetBoard-Oberflächen mit eindeutigen Objekt-IDs versehen. Da diese Objekt-IDs

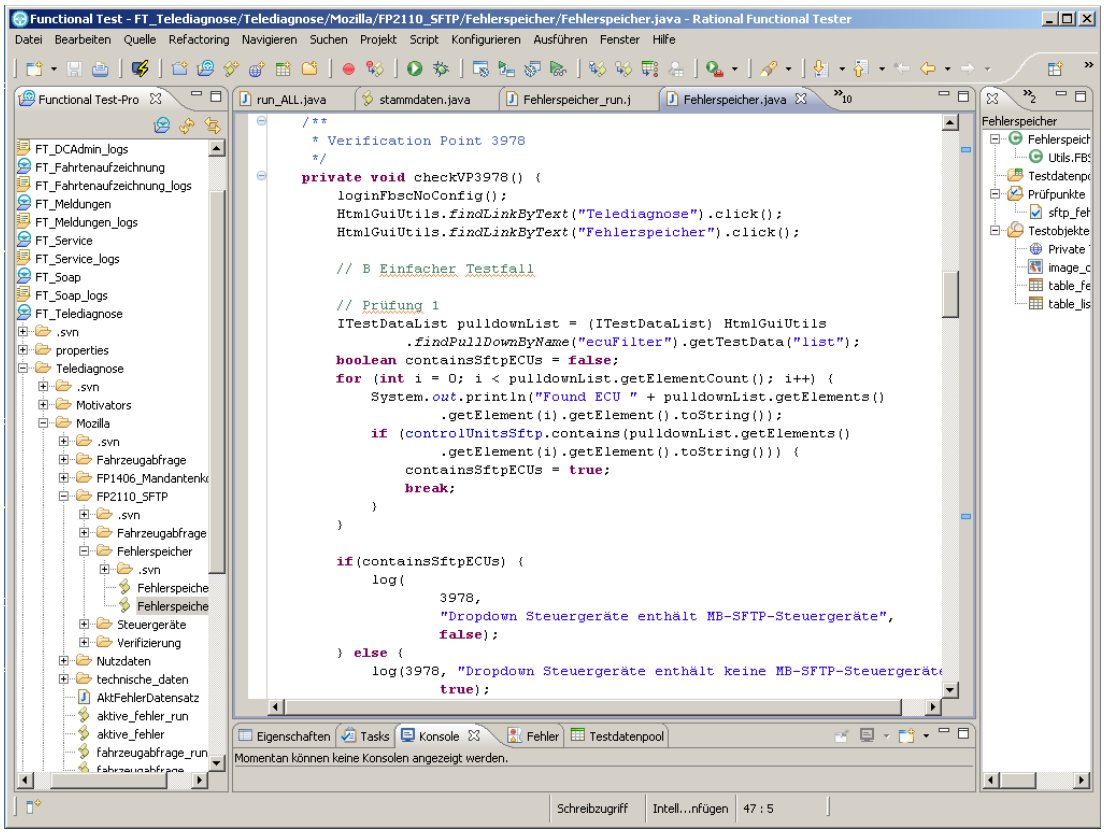


Abbildung 4.4: Functional Tester

das Verhalten der zu testenden Software nicht beeinflussen, ist dies keine Instrumentierung im klassischen Sinne. Die Produkte werden mit Objekt-IDs zum Kunden ausgeliefert.
Um Functional Tester einfacher verwendbar zu machen und um ihn an die FleetBoard-Oberflächen anzupassen, wurden verschiedene Erweiterungen vorgenommen, insbesondere um die Verfolgbarkeit von Anforderungen über Testfälle zu Testergebnissen zu ermöglichen. Dazu wird die sogenannte Prüfpunkt-ID aus ClearQuest („ClearQuest-ID“ oder kurz „cqid“) in Testskripte und Testprotokolle aufgenommen. Weiterhin gibt es einen Wrapper, um soapUI-Testfälle in Skripte von Functional Tester einzubinden und um die soapUI-Testergebnisse in das von Functional Tester erstellte Testlog zu integrieren. Das Testergebnis („fail“, „pass“, „known bug“) wird nach der Testdurchführung zum zugehörigen Testfall (falls vorhanden) in die ClearQuest-Datenbank geschrieben.
Verwendet wird Functional Tester zur Testautomatisierung:

- Von den Spezifizierern und Qualitätssicherern in den farbigen Scrum-Teams.
- Von den Mitgliedern des Teams IT-Betrieb.
- Von den Mitgliedern des schwarzen Teams.
- Zukünftig von den Mitarbeitern im Hardware-Team, wo momentan keine Zeit zur Testautomatisierung zur Verfügung steht.

Dokumentation

Im Wiki des schwarzen Teams existieren mehrere Seiten zu Functional Tester. Dort sind die Einrichtung von Projekten, die grundsätzliche Bedienung, die ersten Schritte sowie Tipps und Tricks ausführlich beschrieben. Zusätzlich existieren Beschreibungen der selbst erstellten Zu-

sätze und Wrapper, beispielsweise der soapUI-Integration. Diese sind allerdings teilweise veraltet. Durch die hohe Komplexität ist eine persönliche Einführung in Functional Tester ohnehin notwendig. Richtlinien zur Testskriptgestaltung sind vorhanden, aber durch die Weiterentwicklung der Testinfrastruktur ebenfalls nicht mehr aktuell.

4.1.4 soapUI

Eviware soapUI ist ein OpenSource-Werkzeug zur Durchführung von Tests von SOAP-Schnittstellen. Es lassen sich mit Hilfe von WSDL schnell SOAP-Abfragen erstellen, parametrisieren und durchführen. Allerdings reicht der normale Funktionsumfang von soapUI nicht für die komplexen Tests bei FleetBoard aus, weswegen das Tool stark erweitert wurde. Das erhöht zusätzlich die Einstiegshürde. Diese Erweiterungen bestehen in einem komplexen Mock-Service, der lokal eine Webservice-Schnittstelle simuliert, die in Java implementiert ist und wiederum komplexe Aufgaben wie Direktzugriffe auf die Datenbank erledigt.
Testfälle bestehen aus einzelnen Testschritten und werden in Testsuiten gruppiert, die sich beliebig verschachteln lassen. Die Ergebnisse von Testschritten können mit sogenannten Assertions (Annahmen) geprüft werden. Dabei werden unter anderem XPath-Ausdrücke verwendet.
Bei der Testdurchführung in soapUI wird kein Log erzeugt. Es wird dem Benutzer lediglich angezeigt, ob sein Test erfolgreich war oder nicht.

Dokumentation

soapUI ist im Wiki des schwarzen Teams sehr ausführlich beschrieben. Beschrieben werden das Layout der Testsuite, die enthaltenen Projekte, die Bedienung des Programms und die Richtlinien zum Schreiben von Testfällen. Die Erweiterungen zu soapUI sind ebenso dokumentiert, wie deren Entwicklung und die soapUI-Seite der Functional Tester Integration.

4.1.5 Manual Tester

IBM Rational Manual Tester ist ein Werkzeug zur Unterstützung von manuellen Tests, das mit Functional Tester mitgeliefert wird und sein Pendant für manuelle Tests darstellt. Hier können Testschritte, Prüfschritte und Sollergebnisse genau in Form eines stark annotierten Dokumentes beschrieben werden. Diese Testbeschreibung aus Benutzeraktionen und Prüfungen kann bei der Durchführung des Tests abgearbeitet werden. Dabei wird der Tester Schritt für Schritt durch den Testablauf geführt. Es können bei einzelnen Testschritten automatisch Texte in oder aus der Zwischenablage kopiert werden oder dem Tester andere nützliche Informationen eingeblendet werden.
Während der Testdurchführung erzeugt Manual Tester ein Testlog, das die durchgeführten Testschritte sowie die Soll- und Istergebnisse enthält. Manual Tester Skripte können über ClearQuest gestartet werden, um das Zurückschreiben der Ergebnisse in die ClearQuest-Datenbank zu ermöglichen.

Dokumentation

Für Manual Tester gibt es keine allgemeine Dokumentation. Wir konnten lediglich eine Kopfzeile finden, die für Manual Tester Skripte verwendet werden soll und einige Änderungsrichtlinien, die die Verfolgung von Änderungen in Manual Tester Skripten erleichtern soll.

4.1.6 Focal Point

Das Produktmanagement verwendet Focal Point um seine Anforderungen einzutragen, zu priorisieren und zu verwalten. Die Entwickler holen aus Focal Point die Anforderungen als Datengrundlage für die Spezifikation. Gegebenenfalls werden Bugs mit Anforderungen in Focal Point verknüpft. Wir haben dieses Tool für die Fachstudie nicht weiter untersucht, da es nicht

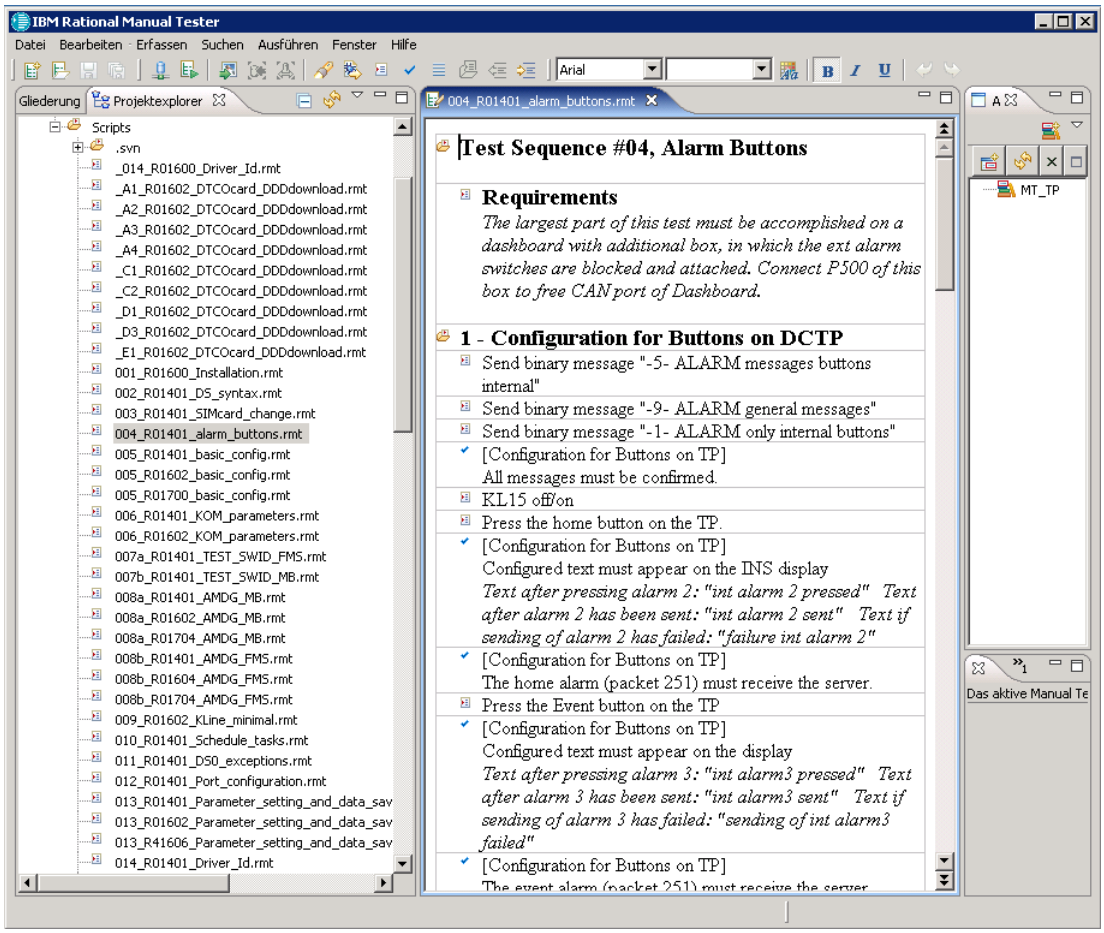


Abbildung 4.5: Manual Tester

für die Testfallverwaltung relevant ist. Lediglich die Anbindung an die anderen Werkzeuge per ID wurde von uns beachtet.

Dokumentation

Zu Focal Point haben wir keine Dokumentation gefunden.

4.1.7 RequisitePro

RequisitePro ist ein Werkzeug zur Verwaltung von Spezifikationsdokumenten und Use Cases. Die Dokumente werden dabei als annotierte Microsoft Word Dokumente gespeichert, in denen einzelne Anforderungen markiert sind. Datengrundlage für die Spezifikationen in RequisitePro sind die Anforderungen aus Focal Point und die mündliche Absprache mit den Produktmanagern. Verwendet wird RequisitePro von den Spezifizierern, die in den Entwicklerteams arbeiten, um Spezifikationen zu schreiben. RequisitePro hat für diese Fachstudie die gleiche Bedeutung wie Focal Point.

Dokumentation

Zu RequisitePro gibt es lediglich Informationen zum Einrichten des Programms. Diese sind im IT-Realisierungs-Wiki hinterlegt. Zur Verwendung konnten wir keine Dokumentation finden.

4.2 Bewertung durch die Mitarbeiter

Bei FleetBoard werden einige Werkzeuge im Zusammenhang mit den Tests verwendet, welche teilweise auf die gleichen Daten zugreifen. Es gibt eine enge Verzahnung der Aufgaben in den einzelnen Werkzeugen, die meist nicht zufriedenstellend unterstützt wird, und oft für Reibung und Frust bei der täglichen Arbeit sorgt.

4.2.1 ClearQuest

Als Verwaltungswerkzeug für die Testfälle ist es nicht nur für unsere Fachstudie, sondern auch für den Testprozess von zentraler Bedeutung. Fast alle Mitarbeiter, die etwas mit Tests zu tun haben, verwenden ClearQuest. Entsprechend viele Meinungen haben wir erhalten.

Allgemeine Verwendung von ClearQuest

Generell wird ClearQuest so eingesetzt, wie es vom schwarzen Team gedacht ist. Es werden Testfälle strukturiert und spezifiziert bzw. wer die Automatisierung implementiert findet darin seine Datengrundlage. Aus diesem Rahmen fallen allerdings die Hardware-Teams. Das Team der Telematikplattform verwendet ClearQuest für die Spezifikation im eigentlichen Sinne und spezifiziert seine Testfälle im Manual Tester. Für die Tests des DispoPilots gibt es in ClearQuest nur eine leere Hülle von Testfällen ohne Prüfpunkte um eine Strukturierung zu erreichen, an die die Manual Tester Skripte gekoppelt sind. Zu erwähnen ist noch, dass die SOAP-Tests nur in ClearQuest existieren um die Ergebnisse eintragen zu können. Prüfpunkte gibt es keine. ClearQuest bildet lediglich die Projektstruktur aus soapUI ab.

Strukturierung der Daten

Dadurch, dass es keine einheitlichen Richtlinien zur Strukturierung gibt, sieht der Datenbaum für jedes Team anders aus. Es gibt alte Daten, die nach Sprint strukturiert waren. Später wurde dann nach Funktionalität umstrukturiert. Innerhalb der Funktionalitäten trennen manche Teams aber wiederum nach Versionen. Dann ist die Trennung zwischen Testfall und Prüfpunkt nicht klar. Es gibt Teams, die in einen Testfall genau einen Prüfpunkt stecken (z.B. das Team, das die Telematikplattform testet), andere haben pro Testfall eine lange Liste von 30 Prüfpunkten. Das DispoPilot-Testteam verwendet eine Nummerierung vor dem Testfallnamen für verschiedene Konfigurationen. Es gibt also momentan zehn Kopien eines Testfalls mit jeweils der Zahl dazu vor dem Namen, obwohl ein Testfall eigentlich mehrere Konfigurationen zulässt. Innerhalb der eigenen Daten wird z.B. für den DispoPilot die Suchfunktion von ClearQuest verwendet, um die passende Konfiguration „auszuwählen“. Durch vorherige Migrationen gibt es zudem sehr viele leere Testfallhülsen ohne Prüfpunkte.

Die Existenz der Prüfpunkte, die eigentlich das sind, was laut Lehrbuch ein Testfall ist, und die Verwendung der Testfälle als weitere Kategorisierungsmöglichkeit neben den Testplänen, kommt niemandem falsch vor und wird akzeptiert, da es anscheinend von ClearQuest so vorgegeben ist. Der Ursprung dieser Strukturierung ist allerdings technischer Natur und eine Erfindung des schwarzen Teams. Das war niemandem aus den anderen Teams klar.

Testfälle

Testfälle sind bei FleetBoard lediglich Container für Prüfpunkte. Diese Prüfpunkte werden in einem extra Fenster an einen Testfall gehängt. Bemängelt wurde dabei, dass keinerlei Abhängigkeiten oder Reihenfolge zwischen Prüfpunkten möglich ist. In der Praxis läuft man durch einen Prüfpunkt durch, der z.B. alle Tests in einem Abschnitt der grafischen Oberfläche enthält, und hat dafür eine gemeinsame Vorbedingung und Nachbedingung. Zwischen den einzelnen Schritten gibt es eine bestimmte Reihenfolge, da sie aneinander anknüpfen. Diese Vorgehensweise ist momentan nicht sinnvoll in ClearQuest abbildbar. Umgangen wird das Problem fast einheitlich durch einen „Setup-Prüfpunkt“, der Vorbedingung für alle anderen Prüfpunkte ist. Ebenfalls bemängelt wird, dass man Testfälle nicht in einer Ansicht „runterschreiben“ kann,

sondern umständlich für jeden Prüfpunkt ein extra Fenster braucht, in dem man vor dem Bearbeiten noch einen „Bearbeiten-Knopf“ drücken muss.

Auffällig bei den Testfällen ist weiterhin, dass es viele von niemandem verwendete Eingabefelder gibt. Es handelt sich meist um Standardfelder, die ClearQuest mitbringt, die aber auch ohne weiteres entfernt werden könnten. Die „Beschreibung des Testfalls“ hat einen Knopf zum Aktualisieren, welcher von Hand gedrückt werden muss, wenn man diese Daten aktuell braucht.

Prüfpunkte

Die Eingabe ist recht übersichtlich, da sie nicht so überladen ist wie bei den Testfällen. Die drei Felder für Vorbedingung, Aktion und Nachbedingung sind für die meisten Tester ausreichend, da sie Freitext zulassen. Manche bemängeln daran, dass Freitext zu viel Spielraum lässt und somit eine einheitliche Verwendung kaum möglich ist. Andere bemängeln, dass die meisten ihrer Testfälle gleich aufgebaut sind und die Felder somit fast immer den gleichen Inhalt bekommen. Man wünscht sich Vorlagen. Die GUI-Tester prüfen z.B. sämtliche Eingabefelder auf standardisierte Wertebereiche. Das ist für alle Felder identisch. SOAP-Tests sind ebenfalls immer gleich aufgebaut. Die Idee der Templates geht allerdings einher mit der Annahme, dass drei Freitextfelder die passende Eingabe für die Prüfpunkte sind.

Suchfunktion

Um Mängel der Werkzeuge zu umgehen sind einige Mitarbeiter gezwungen, die Suchfunktion von ClearQuest zu verwenden. Bemängelt wurde dabei, dass die angebotene Methode über das grafische Zusammenklicken von SQL-Queries unflexibel und kompliziert ist, zusätzlich ist der Funktionsumfang von SQL nicht ausschöpfbar. Die Ausgabe wiederum stellt keinen Zusammenhang zwischen dem Suchergebnis und der Baumstruktur der Testpläne und Testfälle her, da nur eine Liste gefundener Datensätze angezeigt wird. Die Suche eines gefundenen Testfalls in der Baumstruktur geschieht von Hand!

Allgemeine Bewertung

Dass ClearQuest nicht schwer zu erlernen ist, finden eigentlich alle. Auch wir als Außenstehende kommen relativ schnell damit zurecht. Es wurde allerdings durchgehend von allen Beteiligten bemängelt, dass ClearQuest bei der täglichen Verwendung viel zu umständlich ist. Man benötigt extrem viele Klicks, selbst für die einfachsten Aufgaben. Für alles gibt es ein neues Fenster.

Etwas paradox scheint, dass ClearQuest extrem starr und unflexibel ist, obwohl es doch als extrem flexibles Werkzeug für vielerlei Aufgaben konzipiert ist. Leider ist es nur für den Admin bei der einmaligen Konfiguration flexibel. Im Alltag nervt es jeden, der damit arbeiten muss mit seinen unzähligen unsinnigen Macken. Die Bewertungen auf einer Skala von 1-10 gehen zwar weit auseinander und reichen von 2 bis 9,5, allerdings werden die positiven Meinungen nur durch das bloße Erfüllen der Aufgaben von ClearQuest begründet. Angenehm findet das Werkzeug keiner der Befragten.

4.2.2 ITT - Testmanagement

Diese Sammlung von diversen selbst entwickelten Werkzeugen arbeitet komplett auf den Daten von ClearQuest und gleicht dadurch im Grunde lediglich dessen Schwächen aus. Daher ist die Bewertung dieser Werkzeuge recht positiv.

Die Ansicht „Testfallspezifikation“ zeigt alle Testfälle aus der Baumstruktur von ClearQuest tabellarisch wie man es von früheren Excel-Tabellen kennt. Der Nutzen ist uns nicht klar. Keiner der Befragten hat diese Ansicht erwähnt. Sie scheint nicht verwendet zu werden.

„Restaufwandplanung“ bietet die Möglichkeit, die Dauer von Tests vorab zu schätzen, sowie Ergebnisse der Tests manuell einzutragen. Viele Testfälle haben hier den Standardwert von 20 Minuten. Anstatt zur Planung wird das Werkzeug vor allem zum Eintragen der Ergebnisse von manuellen Tests verwendet. Wir haben festgestellt, dass mit dieser Übersicht die Planung der manuellen Tests gemacht wird. Durch das noch fehlende Ergebnis sieht der Tester die Tests, die nach den automatisierten Tests noch nicht durchgeführt wurden. Wenn sie nicht ausgelassen wurden, handelt es sich dabei dann um manuelle Tests.

Mit „Anforderungsabdeckungsbericht“ wird eine Verbindung der Anforderungen in Focal Point und den dazugehörigen Testfällen in ClearQuest hergestellt. Man kann damit recht gut sehen wofür noch Testfälle erstellt werden müssen. Bemängelt wurde daran nichts.

4.2.3 Functional Tester

IBM Rational Functional Tester wird von den Mitarbeitern, die ihn verwenden, durchweg positiv bewertet. Insbesondere der Funktionsumfang und die Flexibilität des Werkzeugs wurden von den Befragten hervorgehoben. Durch die Verwendbarkeit des vollen Java-Funktionsumfangs ist Functional Tester sehr mächtig und kann von den Mitarbeitern für Zwecke eingesetzt werden, die ursprünglich nicht zum Funktionsumfang gehörten, beispielsweise die Fernsteuerung von Servern über SSH.
Weniger gut bewertet wurden Erlernbarkeit und Bedienbarkeit. Der Einarbeitungsaufwand sei recht hoch und die Bedienung ließe zu wünschen übrig. Allerdings ist nach der Einarbeitung ein effizienter Betrieb gut möglich.

4.2.4 soapUI

soapUI wird von allen als sehr nützlich empfunden. Sein purer Zweck, das Testen von SOAP-Schnittstellen, verschafft diesem Werkzeug eine durchweg positive Bewertung, dabei wird die Einstiegshürde von allen als sehr hoch empfunden und es gibt doch einige Macken.

Problematisch ist, dass große Teilbäume aus der Datenstruktur als Projekt in einer einzigen sehr großen XML-Datei abgespeichert werden. Da XML sich über SVN nicht zuverlässig zusammenführen lässt, werden ganze Projekte von einem einzigen Mitarbeiter gesperrt, sodass nur er daran arbeiten kann. Eine Zusammenarbeit mehrerer Benutzer ist also äußerst schwierig bzw. in der Praxis unmöglich und produziert sehr häufig Wartezeiten und die Notwendigkeit der mündlichen Absprache.

In Kauf genommen wird ohne große Beschwerden, dass es manchmal Darstellungsfehler hat und gelegentlich abstürzt. Immerhin gibt es keine bessere Alternative zu dem Programm.

Die Verbindung mit ClearQuest und Functional Tester ist nicht optimal. soapUI hat eine eigene Datenstruktur, die in ClearQuest lediglich mit Dummy-Testfällen abgebildet wird um die Testergebnisse hinterlegen zu können. Für diese Verbindung sorgt Functional Tester indem es zu einem Testfall aus ClearQuest das passende soapUI-Skript startet. Zusätzlich gibt es „Mockup“-Services, SOAP-Server, die dazu da sind um beliebigen Code ausführen zu können. Bei der Verwendung dieser ganzen Verbindungen werden diverse SSH-Verbindungen und gestartete Server benötigt, was dem Tester das Leben unnötig schwer macht.

In soapUI gibt es mit der Strukturierung der Daten das gleiche Problem wie in ClearQuest: Es gibt keine Richtlinien.

Erwähnenswert sind noch die riesigen Property-Dateien, welche Einstellungen sowie eine Art Datenbank enthalten. Diese sind weder übersichtlich noch wartbar.

4.2.5 Manual Tester

Wir haben zwei Personen zum Manual Tester befragt. Je eine aus dem Team für die Telematikplattform und für den DispoPilot. Manual Tester wird lediglich für die Hardwaretests

verwendet, da eine Automatisierung nicht in dem Maße möglich ist wie bei den reinen Softwaretests. Beide Personen bewerten den Manual Tester völlig unterschiedlich. Im Team der Telematikplattform ist man ziemlich unglücklich mit dem Werkzeug. Das Team des DispoPilots ist halbwegs zufrieden.

Vorteile des Manual Testers sind die Anbindungsmöglichkeit an die ClearQuest-Datenbank sowie die automatische Erstellung eines Testlogs.

Zu den Nachteilen:
Auf den ersten Blick ähnelt ein geöffnetes „Test-Skript“ einem Word-Dokument. Es gibt Überschriften, Absätze und Aufzählungen. Zwar hat jedes Element eine formale Bedeutung im Sinne des Tests, in der Praxis spielt das aber keine Rolle, da es sich ja ohnehin um einen manuellen Test handelt, der daher auch nicht automatisch verarbeitet werden kann. Der Unterschied zu Word besteht daher nur in der Anwesenheit eines Pfeils neben dem Dokument während der Durchführung, welcher auch durch einen Finger ersetzt werden könnte, sowie dem automatischen Anlegen einer Kopie des Dokuments in Form eines Testlogs. Dazu kommen diverse Bedienungsschwächen vom Manual Tester. So geht z.B. kein Copy & Paste, auf Reaktionen der Software bei Eingaben wartet man schon mal mehrere Sekunden, Bedienelemente sind viel zu klein für den enthaltenen Text und generell ist die Usability sehr schlecht.

Aus der Abteilung Telematikplattform kam nun die Idee, Test-Skripte in ein XML-Format zu stecken, einen „GUI-Wrapper“ für den Functional Tester zu basteln um anschließend die manuellen Test ebenfalls mit Functional Tester starten zu können. Der Sinn und die Umsetzbarkeit dieses Vorhabens ist fraglich und wir haben dazu geraten zuerst das Ergebnis dieser Fachstudie abzuwarten.

4.3 Schwachstellen

Die Werkzeuge, die sich in der Analyse als relevant herausgestellt hatten, wurden näher betrachtet und ausgewertet. Einen zentralen Stellenwert hat ClearQuest, da es die Testfallverwaltung übernimmt. Dazu gehören erweiterte Ansichten in Form von Webinterfaces im ITT Testmanagement. Direkt im Anschluss in der Prioritätenliste kommen Functional Tester und soapUI, da sie die engste Anbindung an die ClearQuest-Datenbank haben und die meiste Arbeitszeit einsparen. Nicht weniger wichtig, aber weniger präsent bei FleetBoard, ist Manual Tester. Bei FocalPoint und RequisitePro ist für uns nur die Anbindung an die ClearQuest-Datenbank relevant.

Die Abschnitte der einzelnen Werkzeuge sind wie folgt unterteilt:

- Dokumentation: Das ist die Grundlage zur Einlernung in das Werkzeug. Sie wird daher zuerst überprüft.
- Strukturierung der Daten: Alle Programme haben eine Datenstruktur auf der sie arbeiten. Wir werten sowohl das Datenmodell auf technischer Ebene, als auch die sichtbare Strukturierung auf Anwendungsebene aus.
- Erlernbarkeit, Effizienz und Usability sind die drei Kernpunkte, die ein Programm angenehm benutzbar machen. Usability ist sicher auch eine Ursache für die anderen beiden Eigenschaften. Die Aufteilung wurde aber wegen der Auswirkungen so gewählt. Erlernbarkeit sagt aus, wie leicht es für neue Mitarbeiter ist, sich mit dem neuen Programm zurecht zu finden. Effizienz lässt indirekte Rückschlüsse auf benötigte Arbeitszeit zu und Usability allgemein korreliert mit der Mitarbeiterzufriedenheit.
- Anbindung an andere Werkzeuge: Da sämtliche betrachtete Werkzeuge recht eng miteinander verzahnt sind, ist es besonders wichtig dort Schwachstellen zu finden und Anforderungen an neue Lösungen festzuhalten.

- Kosten: Dieser Aspekt ist notwendig für den Vergleich mit neuen Lösungen. Besonders teure Werkzeuge fallen hier auf. Beachtet wurden verschiedene Arten von Kosten, wie sie auch in LL07 Kapitel 2.1 vorkommen. Allerdings werden hier nur auffällige Aspekte erwähnt.

4.3.1 ClearQuest

Dokumentation

(SS 4.1) Generell wird die Dokumentation zu ClearQuest im internen Wiki der zentralen Aufgabe dieses Werkzeugs nicht gerecht. Es sind lediglich zwei Seiten zu ClearQuest vorhanden. Die eine beschränkt sich auf die Einrichtung auf einem lokalen Entwicklungssystem. Solch eine Seite ist notwendig und muss gepflegt werden. Die zweite Seite beschäftigt sich mit der Verwendung von ClearQuest, geht dabei aber nicht genug ins Detail, deckt nicht den gesamten Aufgabenbereich ab und vermittelt kein Gesamtkonzept der Anwendung. Ebenfalls fehlt die Positionierung von ClearQuest relativ zu den anderen Anwendungen im Bereich Test.

Bisher wurde die Aufgabe der Dokumentation komplett durch mündliches Einlernen übernommen. Das ist generell sehr gut, auf eine schriftliche Dokumentation zum Nachschlagen sollte aber dennoch nicht verzichtet werden.

Strukturierung der Daten

(SS 4.2) ClearQuest ist ein sehr generisches Werkzeug. Leider ist das auch im Alltag sehr gut sichtbar. So sind zum Beispiel die ersten beiden Ebenen im Datenbaum für den Anwender komplett überflüssig, denn sie verraten lediglich den eigenen Benutzernamen und die Datenbank zu der verbunden wurde, und liefern zusätzlich die leere Worthülse „Asset-Registries“ als eigene Hierarchieebene. Das Konzept der „Asset-Registries“ ist im Fall von FleetBoard weder verständlich noch nützlich und vermischt sich mit den Testplänen und Testfällen, die die eigentliche Datenstruktur darstellen.

(SS 4.3) Die Strukturierung der Testpläne und Testfälle wurde ursprünglich vom schwarzen Team erdacht. Im Laufe der Zeit musste man die Struktur allerdings ändern, da sie sich als ungeeignet herausgestellt hat. Dadurch gibt es Altlasten, die aufgrund von Schwächen von ClearQuest nie umsortiert wurden und bis heute existieren. Seither wächst die Struktur in ClearQuest weiter ohne Kontrolle durch das schwarze Team. Beispiele dazu sind im Anhang zu finden.

(SS 4.4) Zu dem Knoten „Asset-Registries“, den Asset-Registries selbst, einer von ClearQuest vorgegebenen Ordnerstruktur innerhalb der Asset-Registries, sowie den Testplänen, welche alle auch einfach als Ordner oder Container bezeichnet werden könnten und somit keine Unterscheidung bräuchten, gesellt sich zusätzlich der Testfall, welcher eigentlich als Blatt im Baum gedacht ist, aber nur ein weiteres Containerelement zur Bündelung von Prüfpunkten darstellt. Es gibt also sechs unterschiedliche Containertypen, die sich in der Oberfläche aber lediglich durch ihren Namen unterscheiden und somit keine Typisierung bräuchten. Diese aufwendige Struktur ist sicherlich in manchen Anwendungen sinnvoll, ist für den Test bei FleetBoard aber unnötig kompliziert.

(SS 4.5) Das Konzept der Testfälle und Prüfpunkte wurde bisher technisch begründet. Ein Prüfpunkt bei FleetBoard ist ein Testfall im eigentlichen Sinne. Ein Testfall bei FleetBoard gruppiert lediglich Prüfpunkte und enthält einige Zusatzinformationen, die aber genau so gut in einem Testplan gespeichert werden könnten. Von den Mitarbeitern wird bemängelt, dass das Konzept eine Abfolge von Testschritten mit Priorisierung nicht abdecken kann. Es fehlt die Möglichkeit, die Reihenfolge von Prüfpunkten zu definieren und Abhängigkeiten zwischen ihnen herzustellen. Zudem fehlt die Möglichkeit, für einen Testfall eine Vorbedingung anzugeben, die für alle enthaltenen Prüfpunkte gilt.

(SS 4.6) Es existiert keine Vorgabe zum Umfang und Inhalt eines Testfalls. Der Mitarbeiter entschei-

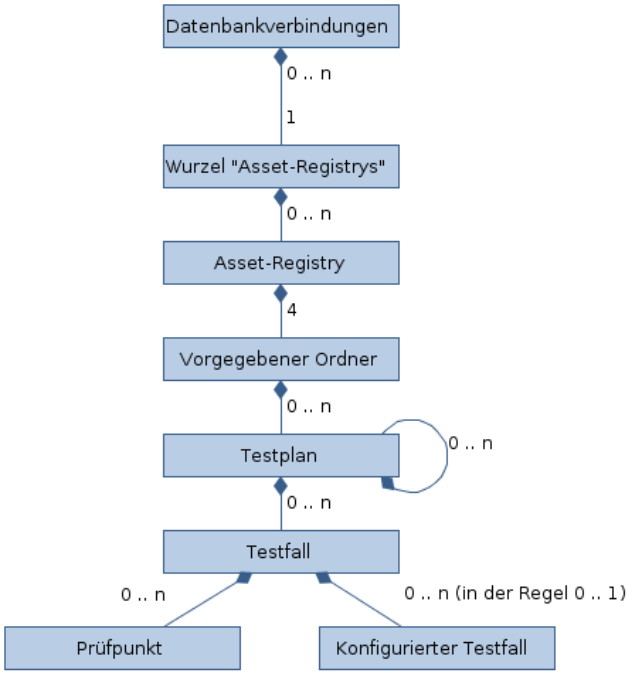


Abbildung 4.6: Datenstruktur in ClearQuest

det also selbst nach unbekannten Kriterien, ob er einen Testfall mit 25 Prüfpunkten, oder 5 Testfälle mit je 5 Prüfpunkten erstellt. Auch in seinem Ermessen liegt die Positionierung des Testfalls in der Hierarchie. Wird ein neuer Testplan angelegt? Gehört der Testfall zu schon vorhandenen Testplänen? Hierfür müssen Vorgaben eingeführt werden. Die Kontrolle dessen ist schwierig, weil es keinen Testfallverantwortlichen gibt. Bei Prüfpunkten wird nicht einmal der Autor gespeichert.

(SS 4.7) Die Konfigurierten Testfälle sind grundsätzlich nur für die Hardware-Teams relevant. Alle anderen Teams verwenden dieses Konzept nur, weil es eine technische Notwendigkeit für die Integration von Functional Tester ist. Dort hat dann ein Testfall genau eine Konfiguration. Bei denen, die es richtig verwenden, fehlt allerdings eine Filterfunktion für Konfigurationen.

Erlernbarkeit

In diesem Punkt haben wir keine gravierenden Mängel zu beanstanden. Der Einstieg in ClearQuest ist relativ unproblematisch. Auch dem Team der Fachstudie war schnell klar, wie dieses Werkzeug funktioniert.

Effizienz

ClearQuest kann bei dem einmaligen Einrichten der Architektur sowie der Einlernung der benutzenden Mitarbeiter punkten. Leider schwächelt es bei der Effizienz und zwar in seiner kompletten verwendeten Funktionalität.

(SS 4.8) Beginnend mit der Übersicht in Form der Baumstruktur gibt es zu bemängeln, dass die Struktur nicht einheitlich definiert ist und entsprechend von jedem anders verwendet wird. Eine gute Suchfunktion könnte dies bis zu einem gewissen Grad kompensieren, aber die Suchfunktion in ClearQuest schafft es weder schnell zielstrebige Eingaben zu ermöglichen, noch eine Verbindung zwischen Suchergebnis und Baumstruktur, und damit eine Übersicht im mentalen Modell des Benutzers, zu schaffen. Die Folge ist, dass man sehr viel Zeit damit verbringt, sich in den Datenstrukturen anderer Mitarbeiter oder sogar seiner eigenen zurecht zu finden, ohne dabei

produktiv arbeiten zu können.

(SS 4.9) Für die Eingabe eines neuen Testfalls öffnet sich ein neues Fenster. Das ist unproblematisch, da ein Testfall bei FleetBoard sehr umfangreich werden kann. Weniger praktisch ist, dass für die Eingabe eines Prüfpunktes erneut ein weiteres Fenster erscheint. Damit sind wir bei drei offenen Ebenen auf dem Bildschirm, was dem Ganzen etwas die Übersicht nimmt. Um Änderungen an einem Formular machen zu können, muss zuerst der Bearbeiten-Modus durch Klick auf einen dafür vorgesehenen Knopf aktiviert werden. Anschließend verlässt man diesen Modus durch Speichern des Formulars. Möchte man schnell diverse Testfälle „runterschreiben" muss man bei jedem Testfall und jedem seiner Prüfpunkte diesen Bearbeiten-Modus verwenden. Man muss also diverse Male von der Tastatur zur Maus wechseln um unsinnige Klicks zu tätigen. In Excel z.B. würde man die komplette Arbeit am Stück erledigen ohne die Maus zu verwenden und dabei erheblich Zeit und Nerven sparen.

(SS 4.10) Die fehlende Möglichkeit für Drag & Drop, sowie Copy & Paste von Testfällen und Prüfpunkten macht den Benutzern das Leben unnötig schwer. Umsortieren ist so aufwändig, dass man es in der Praxis einfach nicht macht. Viele Testfälle in bestimmten Bereichen sind sehr ähnlich und auch Prüfpunkte sind oft nahezu identisch. Trotzdem muss man alles von Hand

(SS 4.11) tippen. Hier gibt es ein großes Einsparungspotenzial. Zusätzlich wurde von den Benutzern eine Funktionalität für Vorlagen (Templates) gewünscht. Damit wären Testfälle nicht nur wesentlich schneller, sondern auch einheitlich befüllt.

(SS 4.12) Ebenfalls Zeit kostet die Tatsache, dass es zwar grundsätzlich möglich ist, Tests von ClearQuest heraus zu starten, was in der Praxis wegen Problemen mit der verwendeten Architektur aber nicht geht. Da aber die ClearQuest-Daten zur Planung der Tests verwendet werden, muss man zuerst anhand der ClearQuest-Daten feststellen, welcher Test als nächstes folgt und diesen anschließend in der Datenstruktur des verwendeten Testwerkzeugs finden und ausführen.

Die Generierung von Java Code für den Funktional Tester hingegen ist sehr nützlich und spart Zeit. Auch wenn hier sicher noch eine engere Anbindung mit mehr Automatisierung denkbar wäre, so spart sie auch so schon Arbeit.

Usability

Effizienz und Usability hängen sehr stark zusammen. Entsprechend schlecht fällt auch die Bewertung der Usability aus.

Die Baumstruktur an sich bietet enorme Vorteile in der Übersicht gegenüber der früheren Strukturierung in Exceltabellen, allerdings wird dieser Vorteil überhaupt nicht ausgespielt. Mit Drag & Drop Funktionalitäten wie man sie z.B. aus dem Windows Explorer kennt, hätte man ein mächtiges Strukturierungswerkzeug welches jeder Exceltabelle überlegen ist. Die Suchfunktion müsste ihr lineares Ergebnis mit dem Baum verknüpfen, sodass dem Benutzer die Navigation an die entsprechende Stelle im Baum abgenommen wird. Die Folge ist ein Strukturchaos in den Daten und eine fehlende Möglichkeit, sich trotzdem in dem Chaos zurecht zu finden.

Neben der Ergebnisdarstellung der Suche ist auch die Eingabe sehr unintuitiv. Es kann nur nach verschiedenen Feldern gesucht werden, die mit und- und oder-Operatoren in Form einer Baumstruktur verknüpft sind. Diese Umsetzung einer Sucheingabe ist völlig ungeeignet und ist nicht nur unübersichtlich, sondern schränkt auch in den Möglichkeiten ein.

Sehr störend bei sämtlichen Eingaben ist der „Bearbeiten"-Modus. Es handelt sich bei der Darstellung sowieso um Formularfelder, die im Normalfall einfach nur deaktiviert sind, um Bearbeitungen zu verhindern. Jede Textbearbeitung hat diese Probleme aber schon vor 15 Jahren besser behandelt als ClearQuest.

Der Arbeitsfluss bei der Eingabe eines Testfalls mit allen Prüfpunkten ist verbesserungswürdig, denn das ist momentan nicht ohne Unterbrechung möglich. Bemängelt wurden zu viele unnötige Klicks und wechselnde Ansichten.

(SS 4.13) Fern von jeglicher Usability ist die Implementierung der textuellen Zusammenfassung eines kompletten Testfalls für die weitere Verwendung in externen Werkzeugen. Die Generierung der Zusammenfassung muss nach Änderungen von Hand angestoßen werden, was natürlich schnell mal vergessen wird. Das ist besonders gravierend, da man sich bei FleetBoard im Laufe der Zeit immer mehr von ClearQuest wegbewegt hat und vermehrt auf die externen Werkzeuge setzt.

Anbindung an andere Werkzeuge

Grundsätzlich bietet ClearQuest die technischen Voraussetzungen für eine Anbindung an externe Werkzeuge an. Leider kamen bei der Einrichtung der gewünschten Architektur Probleme auf, die die Anbindung erschwerten. Tests, die mit Functional Tester automatisiert wurden, können momentan nicht von ClearQuest heraus gestartet werden, obwohl das ursprünglich so vorgesehen war. Stattdessen werden die Tests in Functional Tester selbst gestartet, welche anschließend die Ergebnisse in die ClearQuest-Datenbank schreiben. Möchte man Testergebnisse der Tests in soapUI ebenfalls in der ClearQuest-Datenbank speichern, so muss man diese in ein Functional Tester Skript integrieren und von dort aus starten. Die Anbindung von Manual Tester ist besser, da diese Tests von ClearQuest aus gestartet werden können. Tut man das nicht, dann werden auch keine Testergebnisse gespeichert.

Die Anbindung der Webinterfaces spielt nur bei deren Implementierung eine Rolle, ist aber bei der Verwendung (bis auf die Zusammenfassung der Testfälle) nicht sichtbar.

Kosten

Laut Aussage der Mitarbeiter spart ClearQuest in Zusammenhang mit Functional Tester und soapUI, trotz aller Macken, erheblich Zeit und Nerven. Früher wurde anhand von Exceltabellen komplett manuell getestet. Für einen kompletten Durchlauf hat man ca. drei Wochen benötigt. Aktuell benötigt man zwei Wochen, allerdings mit höherem Funktionsumfang der zu testenden Software, dem Vielfachen an Testfällen und jeweils deutlich mehr Eingaben. Der Verwaltungsaufwand ist höchstens genau so hoch wie zu Excel-Zeiten. Die Automatisierung ist Mehraufwand, den es vorher nicht gab. Man kann also sagen, dass ClearQuest im Vergleich zur Zeit vor seiner Einführung weniger Kosten verursacht als es einspart. Die Einführung war also sinnvoll.

Kennt man die Situation vorher nicht, dann erkennt man aber trotzdem noch sehr viele Möglichkeiten zu sparen. Ein Werkzeug, das die Benutzer weniger nervt und nicht von der eigentlichen Arbeit abhält, kann die Arbeit noch deutlich effizienter erledigen als es ClearQuest momentan tut.

Abgesehen von der reinen Zeit, die ClearQuest bei der Verwendung beansprucht, kommt noch hinzu, dass ein genervter Mitarbeiter wesentlich weniger produktiv ist als ein zufriedener. Leider ist so gut wie niemand bei FleetBoard mit ClearQuest wirklich zufrieden, sondern regt sich eher noch lauthals darüber auf.

Die Lizenzkosten sind auch nicht zu vernachlässigen. Mit mehreren tausend Euro pro Floating License ist die Software von IBM kein Schnäppchen. Man könnte vermuten, für so viel Geld bekommt man eine Software, die bei der Verwendung Einsparungen von Mitarbeiterzeit mit sich bringt. Im Vergleich zu dem, was theoretisch möglich wäre (wir machen hier ja relativ viele konkrete Vorschläge), scheint dem aber leider nicht so.

Sonstiges

Allgemein wurde noch bemängelt, dass es kein einheitliches Vokabular gibt und teilweise Abkürzungen verwendet werden, die nicht jeder kennt. Auch wird toleriert, wenn jemand Testfälle auf englisch beschreibt.

Abbildung 4.7: Missverständliche Icons

(SS 4.14) Außerdem sollte auf überflüssige Formularfelder verzichtet werden. Die Standardformulare von ClearQuest zur Eingabe eines Testfalls sind alle noch vorhanden, werden aber größtenteils nicht verwendet.

Fazit

Die von FleetBoard selbst entwickelten Webinterfaces übernehmen mehr und mehr Aufgaben, die ClearQuest so nicht gut unterstützt. Es kann seiner zentralen Rolle also nicht gerecht werden. Viel mehr wird eine Werkzeugwelt drum herum aufgebaut, die es ermöglicht, so viel wie möglich auf ClearQuest zu verzichten. Das zeigt uns, wie unangenehm ClearQuest allgemein für die Mitarbeiter ist und wie groß die Schwächen sind, wenn es darum geht, die Daten für mehr zu verwenden, als sie nur zu verwalten. Da die Verwaltung aber ohne weitere Verwendung reiner Selbstzweck ist, kann man von einem guten Testfallverwaltungswerkzeug die Anbindung an andere Werkzeuge und die Möglichkeit zur weiteren Auswertung der Daten eigentlich erwarten.

4.3.2 ITT Testmanagement

Dokumentation

(SS 4.15) Direkt im ITT-Wiki gibt es nur eine Seite, die für die Entwickler des Testmanagement-Werkzeugs interessant ist.

Der Teil, der für den normalen Benutzer gedacht ist, ist auch im ITT Wiki verlinkt, weicht aber von der sonstigen Dokumentation ab, da es ein PDF ist und nicht wie der Rest im Wiki direkt dokumentiert wurde. Dieses Dokument wird dem Niveau von FleetBoard nicht ganz gerecht. Es beschreibt einzelne Oberflächen auf niedriger Ebene indem einzelne Schaltflächen beschrieben werden. Das reicht, um eine aus Usability-Sicht schlechte Oberfläche benutzbar zu machen, aber die Konzepte und der Nutzen der Oberflächen werden nicht beschrieben. Ein Zusammenhang zum Testprozess wird in diesem Dokument nicht ausreichend hergestellt. Beispiel: Es wird inklusive Screenshot ausführlichst erklärt wie der Login zur Weboberfläche funktioniert, allerdings nicht, warum ein Login überhaupt erforderlich ist. Begrifflichkeiten, wie z.B. der „Langzeittest“, werden hier nicht erklärt. Es wird lediglich jede Ansicht mit dem Kommentar vorgestellt, dass sie selbsterklärend ist. Die Erklärung der Begriffe erfolgt mündlich bei der Einlernung durch Kollegen.

Erlernbarkeit

Abgesehen von ein paar kleineren Mängeln ist der Umgang mit dem Werkzeug schnell erlernbar. Es gibt Icons, die eher Verwirrung stiften, als zum besseren oder schnelleren Verständnis beizutragen. So ist ein grüner Button, der nach „Ansicht aktualisieren“ aussieht, für die Funktion „Aktion rückgängig machen“ zuständig. Ein weißes Kreuz auf rotem Grund steht für „zurück zur Hauptseite“ und nicht etwa für Löschen oder ähnliches (siehe Abbildung 4.7).

(SS 4.15) Auch Unklar ist die Bedeutung mancher Begriffe. Was bedeutet „Langzeittest“? Von was für Analysen ist in der „Analysenübersicht“ die Rede? Warum gibt es eine Ansicht für „Regressionstests“, wenn doch alle Tests bei FleetBoard Regressionstests sind?

Der Rest ist tatsächlich selbsterklärend.

Effizienz

(SS 4.16) Die Effizienz der Weboberfläche wird dadurch negativ beeinflusst, dass teilweise noch nicht automatisierte Tests mit Hilfe dieser Oberfläche durchgeführt werden und die dafür angebotenen Funktionen etwas unpraktisch sind. Es gibt keine Möglichkeit, mehrere Tests zu gruppieren und hintereinander „durchzuklicken“. Daher müssen die Tester in der Oberfläche mehrfach auf- und abscrollen bzw. vor und zurück. Ein fehlender Filter nach Restaufwand macht die Suche nach manuell durchzuführenden Tests aufwändiger als nötig. Außerdem skalieren einige Ansichten mit den enormen Datenmengen relativ schlecht, was sich teilweise durch Ladezeiten, die größer als 20 Sekunden sind, äußert.

Usability

(SS 4.17) Auf den ersten Blick erweckt das Werkzeug den Eindruck einer gut durchdachten Oberfläche, die klar strukturiert ist. Doch bei genauerem Hinsehen wird klar, dass diese vermeintlich gut durchdachte Struktur wohl im Nachhinein angepasst wurde, was sich dadurch äußert, dass man teilweise auf anderen Seiten landet, als denen, die man erwartet, wenn man in der Übersicht auf einen Link klickt. So landet man nach dem Klick auf „Aufwandplanung“ auf einer Seite mit der Überschrift „Restaufwandplanung“. Die Bezeichner für die Seiten wurden also nicht einheitlich gewählt und sorgen für Verwirrung beim Benutzer.

(SS 4.18) Neben der Namensverwirrung fällt ein Detail erst bei tieferen Überlegungen auf: Das ITT Testmanagement wurde allein für den Zweck entwickelt Aufgaben zu erleichtern, die ClearQuest nicht ausreichend unterstützt. Es liegt nahe das ITT entsprechend diesen Aufgaben zu strukturieren und jede Zielgruppe dadurch auf die richtige Seite zu führen. Die Übersichtsseite ist aber nur eine Feature-Aufzählung, aus der sich jeder Mitarbeiter die passenden raussuchen muss, um seine Aufgaben zu erledigen. Dadurch weist das ITT ebenfalls Schwächen auf, die eine Dokumentation überhaupt erst notwendig machen.

(SS 4.19) Weitere Minuspunkte sammelt die Anwendung bei Darstellungen, die über die dreifache Breite eines Bildschirmes hinausreichen und einem somit den Überblick über das Angezeigte verwehren. Die Buttons, die schon unter dem Punkt Erlernbarkeit erwähnt wurden, tragen auch nicht zu einer guten Usability bei.

Anbindung an andere Werkzeuge

Das ITT Testmanagement ist ein zusätzliches Frontend für die ClearQuest-Datenbank. Eine Anbindung an andere Werkzeuge speziell für das ITT Testmanagement existiert daher nicht.

Kosten

Lizenzkosten entstehen bei diesem Werkzeug keine, da es eine Eigenentwicklung ist. Daraus resultiert allerdings, dass Kosten in Form von Arbeitszeit der Mitarbeiter entstehen. Diese sind für uns aber nicht quantitativ nachvollziehbar.

Die Nutzer sind mit dem Werkzeug bis auf eine etwas umständliche Bedienung beim Durchführen manueller Tests zufrieden.

4.3.3 Functional Tester

Dokumentation

Das Wiki des schwarzen Teams enthält folgende Anleitungen bzw. Informationen:

- Installation und Konfiguration von Functional Tester
- Anlegen und Konfigurieren bzw. Importieren eines Projekts

- Schreiben, Aufnehmen und Wiedergeben eines einfachen Skriptes
- Funktionsweise der einzelnen selbst entwickelten Frameworks
- Beschreibungen und Lösungen zu häufig auftretenden Problemen

(SS 4.20) Es fehlen:

- Ein aktueller Styleguide mit Code- und Qualitätsrichtlinien für Testskripte (der vorhandene ist längst überholt)
- Eine Schritt-für-Schritt-Anleitung für ein gutes Testskript

Erlernbarkeit

Functional Tester ist nicht selbsterklärend. Ohne längere Auseinandersetzung mit der Dokumentation und einer persönlichen Einweisung ist das Schreiben von Testskripten auf „FleetBoard-Niveau“ nicht zu erlernen. Dabei wirkt besonders erschwerend, dass Functional Tester Skripte bei FleetBoard kaum etwas mit typischen Functional Tester Skripten zu tun haben, da das selbst gebaute Objekterkennungsframework die Methoden von Functional Tester großteils ersetzt. Zusätzlich gibt es für verschiedene GUI-Arten (HTML, Swing, SWT) spezielle Methoden und Frameworks.
Functional Tester zeigt an mehreren Stellen eigenartige Bugs oder unerwartete Verhaltensweisen, die ohne Erklärung eines erfahrenen Benutzers viel Zeit und Nerven kosten.
Für eine gute Einarbeitung sind ca. zwei Stunden mit einem erfahrenen Mitarbeiter notwendig, plus einige Stunden zum Lesen vorhandener Skripte, der Dokumentation und zum Ausprobieren.

Effizienz

Zur Effizienz gab es keine Beschwerden. Alle Benutzer waren durchweg der Meinung, dass das Schreiben von Testskripten schnell genug geht und dass es keine unnötigen Arbeitsschritte gibt. Auch die Ausführung verbraucht nicht unnötig Zeit. Einige Mitarbeiter wünschten sich eine Möglichkeit, die Texte der zu implementierenden Prüfpunkte in Funktional Tester neben dem aktuellen Testskript zu sehen.

(SS 4.21) Ein Problem ist allerdings, dass die Testskripte keinen festen Richtlinien folgen. Beispielsweise werden teilweise Testdaten hardcoded, spezielle Testflotten und Daten verwendet, die nicht reproduzierbar sind. Das sorgt dafür, dass ein Testskript im nächsten Release nur schlecht funktioniert. Bei der Ausführung des Skriptes im nächsten Releasetest führt dies möglicherweise zu Schwierigkeiten und vergrößert den Wartungsaufwand für die Testskripte.

Usability

Für den eingelernten Benutzer hat Functional Tester keine größeren Schwächen in der Bedienung.

Anbindung an andere Werkzeuge

Functional Tester ist das einzige Werkzeug, das Testergebnisse automatisch in die ClearQuest-Datenbank schreiben kann. Es wird daher benötigt, um die Ergebnisse von soapUI-Tests zu speichern. Dafür gibt es einen Wrapper zur Ausführung von soapUI-Testfällen, der die Logausgaben von soapUI in das Testlog von Functional Tester überträgt. Die Anbindung an soapUI scheint ausgereift und durchdacht zu sein und wird sehr viel eingesetzt.

Ein Testskript kann seine Ergebnisse nur an einen Konfigurierten Testfall anhängen. Manche Mitarbeiter wünschen sich, dass Functional Tester seine Ergebnisse flexibler mit mehreren Testfällen oder mehreren Prüfpunkten verbinden kann.

Lizenzkosten

Bei FleetBoard gibt es vier Floating-Lizenzen (flexibel zwischen PCs tauschbar) und neun Node-Locked-Lizenzen (an PCs gebunden). Die Kosten hierfür belaufen sich laut Listenpreis auf insgesamt ca. 95.000 €.

Fazit

Functional Tester wird von den FleetBoard-Mitarbeitern und den Autoren der Fachstudie für gut befunden. Die Integration zu ClearQuest ist in Details verbesserungsfähig und sollte vor allem flexibler werden.

4.3.4 soapUI

Dokumentation

soapUI ist im Wiki sehr gut beschrieben. Es geht daraus die Einrichtung und Verwendung von soapUI, sowie die Strukturierung der Daten hervor. Natürlich sind einige unten beschriebene Probleme hier verwurzelt, weil es nicht anders dokumentiert ist. Ansonsten gibt es aber nichts zu bemängeln.

Strukturierung der Daten

Während soapUI eigentlich sehr nützlich ist und dafür von den meisten Mitarbeitern gelobt wird, stellt die Strukturierung der Daten eine sehr große Schwäche dar, die in Form des Datenformats bei der Zusammenarbeit und der Wartbarkeit stört und in der sichtbaren Form der Baumstruktur ähnliche Probleme aufweist, wie die Daten in ClearQuest.

(SS 4.22) soapUI enthält bei FleetBoard Projekte als oberste Hierarchieebene. Jedes dieser Projekte enthält sehr viele Testfälle und wird jeweils komplett in einer einzelnen XML-Datei gespeichert. Da SVN von Haus aus keine gute Möglichkeit bietet, XML zuverlässig zusammen zu führen, wird das entsprechend auch nicht gemacht, sondern ein Mitarbeiter sperrt ein Projekt auf SVN-Ebene, wenn er Änderungen vornehmen möchte. Die Folge ist, dass man andauernd von der Arbeit abgehalten wird, weil das benötigte Projekt gerade von jemand anderem gesperrt ist. Eine Zusammenarbeit ist so nicht wirtschaftlich möglich.

Bei der sichtbaren Baumstruktur gibt es zum einen das Problem, wie bei ClearQuest auch, dass keine klaren Regeln zur Strukturierung vorgegeben sind. Neue Daten werden von den Mitarbeitern nach besten Gewissen eingefügt, aber es gibt keine Kontrolle durch das schwarze Team. Zum anderen ist die Baumstruktur in soapUI eine Parallelstruktur zu der Baumstruktur in ClearQuest und muss stets damit synchronisiert werden. Auffällig ist, dass soapUI eigentlich viel mehr macht, als einfach nur SOAP-Schnittstellen zu testen.

(SS 4.23) Zusätzlich zu den Projektdaten gibt es noch globale Daten, welche für Tests verwendet werden können. Dabei handelt es sich um relativ große Datenmengen, die alle in einer einzigen Properties-Datei abgespeichert werden. Dadurch ist es aufwändig, diese Daten zu verwalten und zu warten.

Erlernbarkeit, Effizienz und Usability

Hier weist soapUI einige Schwächen auf, die aber im Rahmen des Ertragbaren bleiben. Da soapUI selbst nicht im Fokus unserer Fachstudie steht, sondern nur die Strukturierung der Daten

sowie die Anbindung an die anderen Werkzeuge, werden wir hier nicht weiter auf Erlernbarkeit, Effizienz und Usability eingehen. Es ist nicht vorgesehen, soapUI selbst zu ersetzen, zumal es die gängigste Lösung im Bereich der SOAP-Tests ist.

Anbindung an andere Werkzeuge

Eigentlich braucht soapUI lediglich eine Anbindung an ClearQuest, um die Ergebnisse der Tests speichern zu können. In der Realität wird Functional Tester aber verwendet, um die Testergebnisse in die ClearQuest-Datenbank zu speichern, denn mit soapUI selbst geht das nicht. Die Testfälle, die noch keine Anbindung an Functional Tester haben, werden aus soapUI heraus gestartet und das Ergebnis wird von Hand eingetragen. Damit ist die Verbindung zu Functional Tester ebenfalls notwendig.

In die andere Richtung kommt noch hinzu, dass man aus soapUI heraus beliebigen Code ausführen können möchte. Als Lösung verwendet man sogenannte „Mockup Services“ welche per SOAP in soapUI verwendet werden können und ein, oft lokal laufendes, Programm starten. Etwas umständlich ist die Notwendigkeit diverse SSH-Verbindungen aufbauen und Server starten zu müssen. Das hält von der eigentlichen Arbeit ab.

Kosten

soapUI selbst ist Open Source, bringt also keine Lizenzkosten mit sich. Relevante Kosten entstehen durch die Wartung von eigenen Anpassungen am Quellcode, die gemacht wurden, um soapUI mit Functional Tester zu verbinden.

Trotz der gefundenen Schwächen sorgt soapUI auch für eine hohe Zufriedenheit, denn manuell Testen ist sehr viel aufwändiger. Es entstehen also keine zusätzlichen Kosten durch unzufriedene Benutzer.

4.3.5 Manual Tester

Dokumentation

(SS 4.24) Zu Manual Tester gibt es keine allgemeine Dokumentation. Es gibt lediglich ein paar Richtlinien zur besseren Verfolgung von Änderungen an Manual Tester Skripten und eine vorgeschriebene Kopfzeile, die bei den Skripten verwendet werden soll.

Erlernbarkeit

Da Manual Tester keinen großen Funktionsumfang hat, ist es auch schnell erlernt.

Effizienz

(SS 4.25) Die Effizienz ist sehr viel niedriger als sie sein könnte und auch sein müsste. Schlechte Usability, lange Wartezeiten bei jeglichen Bearbeitungs- und Durchführungsaktionen, sowie fehlende Unterstützung für Copy & Paste, rauben Manual Tester jeden Vorteil gegenüber dem manuellen Testen.

Usability

(SS 4.26) Die Oberfläche lässt sich nicht intuitiv bedienen. Es werden Symbole verwendet, deren Bedeutung a priori nicht klar ist. Die Hauptfunktionen beim Durchführen eines Tests sind nicht als solche zu erkennen und zum Teil noch nicht einmal richtig lesbar. So gibt es ein Dropdown Menü zur Auswahl ob ein Testschritt bestanden ist oder nicht, das so schmal ist, dass nur „Fehl“ von Fehlgeschlagen angezeigt wird. Es ist weder ersichtlich, wofür das Dropdown Menü ist, noch wofür der Button links davon ist, von dem nur „nwende“ zu erkennen ist und das obwohl

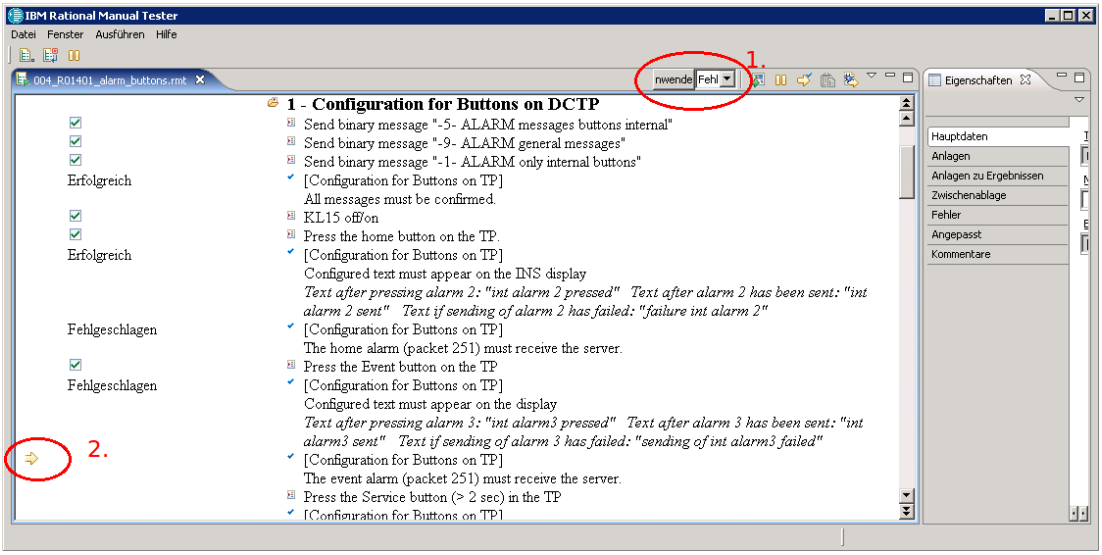


Abbildung 4.8: Manual Tester Screenshot

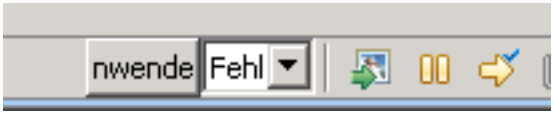


Abbildung 4.9: Unkenntlicher Button und zu kleines Dropdown Menü

mehr als genügend Platz für die beiden Felder vorhanden ist. (siehe 1. in Abbildung 4.8 bzw. Abbildung 4.9)

Anbindung an andere Werkzeuge

Manual Tester wird bei FleetBoard ausschließlich mit ClearQuest verbunden. Diese Anbindung funktioniert so, wie sie von IBM gedacht ist. Wenn Testskripte aus ClearQuest gestartet werden, wird das Testlog in die ClearQuest-Datenbank geschrieben. Wird das Skript direkt aus Manual Tester gestartet, wird das Ergebnis lokal gespeichert.

Kosten

Lizenzkosten entstehen durch die Verwendung nicht, da Manual Tester bei Functional Tester mitgeliefert wird. Inzwischen wurde die Entwicklung von Manual Tester durch IBM eingestellt und dessen Funktionalität in den „Quality Manager“ übernommen.

Zur Zufriedenheit der Benutzer:
Der Manual Tester wird nur von zwei Benutzergruppen bei FleetBoard eingesetzt: Von den Teams DispoPilot und Telematikplattform.

Das Team DispoPilot verwendet ihn nicht so wie er gedacht ist, sondern testen überwiegend manuell auf Papier. Die Mängel die beim „richtigen“ Betrieb auftreten, fallen ihnen daher nicht auf. Somit ist auch niemand unzufrieden mit dem Werkzeug.

(SS 4.27) Anders sieht es bei dem Team der Telematikplattform-Tests aus, das versucht, den Manual Tester so einzusetzen, wie er gedacht ist. Die schlechte Usability schlägt in der Zufriedenheit voll zu Buche. Die Mitarbeiter sind sehr unzufrieden mit Manual Tester und haben sich bereits andere Lösungsansätze überlegt, um Manual Tester zu umgehen. Selbst mit Excel, was zuvor verwendet wurde, wären die Mitarbeiter zufriedener.

Sonstiges

(SS 4.28) Eine sehr markante Eigenart in der Verwendung von Manual Tester ist, dass das Team Dispo-Pilot-Test die Skripte nicht am PC ausführt, sondern ausdruckt und dann mit einem Stift Punkt für Punkt abarbeitet. Das ausgefüllte Formular wird dann als Ergebnis des Tests von Hand in das „ITT-Testmanagement“ Werkzeug eingetragen. Begründet ist das durch fehlende Lizenzen, denn zum Testen wurden einige Werkstudenten eingestellt, dabei gibt es nur vier verfügbare Lizenzen die allein durch die regulären Mitarbeiter verbraucht werden.

4.3.6 Focal Point

Als Werkzeug zur Verwaltung von Anforderungen ist Focal Point nicht direkt für die Fachstudie relevant. Lediglich die Anbindung an andere Werkzeuge ist interessant. Es wird durchweg mit IDs gearbeitet. Ein Testfall in ClearQuest speichert eine Focal Point ID. Man muss für die Verbindung also lediglich beide Programme offen haben und die ID vom einen in das andere eintragen. Das ist die einfachst mögliche Verknüpfung und bisher ausreichend. Wenn eine neue Lösung eine bessere Anbindung ermöglicht, ist das wünschenswert, aber nicht notwendig.

4.3.7 RequisitePro

Für RequisitePro gilt das gleiche wie für Focal Point. Es wird ebenfalls mit IDs gearbeitet, die in andere Werkzeuge eingetragen werden. Eine engere Verbindung gibt es auch hier nicht.

4.4 Anforderungen

4.4.1 ClearQuest

4.1 Unzureichende Dokumentation zu ClearQuest

- Ursache: Die Dokumentation zu ClearQuest ist nicht umfangreich und aktuell genug.
- Anforderung: Die Dokumentation zum verwendeten Testfallverwaltungswerkzeug muss auf dem aktuellen Stand gehalten werden. Ein neuer Benutzer sollte sich mit Hilfe der vorhandenen Dokumentation mit den Konzepten und Methoden in ClearQuest vertraut machen können.

4.2 Unnötige Elemente im Testfallbaum

- Ursache: Von ClearQuest vorgegebene Strukturierungselemente im Testdatenbaum werden von FleetBoard nicht (richtig) verwendet und erschweren die Übersichtlichkeit.
- Anforderung: Der Testdatenbaum sollte genau die wesentlichen Elemente anzeigen, die für die Arbeit mit dem Testfallverwaltungswerkzeug notwendig sind.

4.3 Unkontrollierte Struktur des Testfallbaums

- Ursache: Durch mangelnde Kontrolle und ungenügende Vorgaben ist die Strukturierung des Testfallbaums in ClearQuest inkonsistent und redundant.
- Anforderung: Klare Vorgaben zur Platzierung und Ordnung von Testfällen sollten existieren und deren Einhaltung muss überwacht werden. Die vorhandenen Daten müssen neu strukturiert werden.

4.4 ClearQuest-Testfall ist kein Testfall

- Ursache: Der ClearQuest-Testfall ist kein Testfall im Sinne des Lehrbuchs. Er enthält keine Vorbedingung, keine Benutzeraktion und keine Nachbedingung. Diese Daten sind in einer extra Entity dem Testfall untergeordnet. Der Testfall ist damit nur ein Strukturelement.
- Anforderung: Der Testfallbaum sollte nur aus Strukturelementen und Testfällen bestehen, wobei die Testfälle die eigentlichen Testdaten wie Vorbedingung usw. enthalten und Kindelemente im Baum sind.

4.5 Aktuelle Methode zur Testfallspeicherung erlaubt keine Ordnung

- Ursache: Weil momentan Prüfpunkte in Testfällen nicht geordnet werden können, sind keine Reihenfolge der Durchführung und keine Abhängigkeiten definierbar.
- Anforderung: Es müssen eine Ordnung und Abhängigkeiten zwischen Testfällen angegeben werden können.

4.6 Soll-Umfang und Soll-Inhalt eines Testfalls ist nicht definiert

- Ursache: Weil es keine Vorgabe zu Umfang und Inhalt eines Testfalls gibt, unterscheidet sich der Aufbau der Testfälle sehr.
- Anforderung: Es sollte eine Vorgabe zum Umfang des Testfalls geben. Auch zur Positionierung bzw. Einordnung im Testfallbaum sollte es eine Richtlinie geben, deren Einhaltung kontrolliert werden muss.

4.7 Uneinheitliche Verwendung von Konfigurierten Testfällen

- Ursache: Nur das Hardware-Team verwendet bei FleetBoard Konfigurierte Testfälle. Für alle anderen Teams sind diese nicht notwendig und werden nur verwendet, weil es technisch notwendig ist.
- Anforderung: Konfigurationen von Testfällen müssen ein optionales Element sein.

4.8 Schwer bedienbare Suchfunktion

- Ursache: Die vorhandene Suchfunktion unterstützt komplexe Suchanfragen, die Teile der SQL-Funktionalität unterstützen und als Queries gespeichert werden. Dabei ist es nicht möglich, „kurz mal“ nach einem Testfall zu suchen.
- Anforderung: Das Testfallverwaltungswerkzeug sollte für die Suche im Testdatenbaum eine einfache, schnell erreichbare Suchfunktion bereitstellen, die die wesentlichen Daten der Testfälle durchsucht. Als Darstellung wäre eine gefilterte Ansicht des Baumes denkbar.

4.9 Klick- und fensteranzahlintensive Bedienung

- Ursache: Für viele Bearbeitungen in ClearQuest wird ein neues Fenster geöffnet. Die rechte Seite der Ansicht neben dem Testfallbaum wird für die eigentliche Arbeit nicht verwendet und bleibt im Alltag nutzlos. Der Bearbeiten-Modus sorgt für zusätzliche Klicks.
- Anforderung: Ansichten zum Bearbeiten müssen besser integriert werden und mit weniger Klicks bedienbar sein. Im Alltag braucht man eine Möglichkeit schnell viele Testfälle einzugeben.

4.10 Schlechte Unterstützung der Strukturierung von Testfällen und Strukturierungselementen

- Ursache: Sind Testfälle oder Strukurierungselemente erst einmal angelegt, so gibt es keine einfache Möglichkeit, diese zu verschieben oder zu kopieren.
- Anforderung: Eine bessere Unterstützung zur Strukturierung von Testfällen und Strukturierungselementen durch z.B. Drag & Drop, sowie Copy & Paste muss geschaffen werden.

4.11 Fehlende Templates für Texte

- Ursache: Obwohl viele Testfallinhalte wie Vorbedingungen häufig gleichförmig sind, gibt es keine Layout- oder Inhaltsvorlagen.
- Anforderung: Für die verschiedenen Arten von Testfällen sollten die Ansichten zum Anlegen und Verändern von Testfällen Layout- und Inhaltsvorlagen anbieten.

4.12 Tests können nicht aus dem Testfallverwaltungswerkzeug gestartet werden

- Ursache: Wegen technischer Schwierigkeiten in der Vergangenheit wird die Möglichkeit des Anstoßens von Tests aus der Testfallverwaltung heraus nicht genutzt.
- Anforderung: Es sollte möglich sein, Testskripte und manuelle Tests aus dem Testfallverwaltungswerkzeug zu starten.

4.13 Generierung der Testfallzusammenfassung nicht vollautomatisch

- Ursache: Da ein Testfall momentan bei FleetBoard aus mehreren Prüfpunkten besteht, sind die einzelnen Prüfpunkttexte nur schwer zugänglich. Daher gibt es eine textuelle Zusammenfassung dieser Prüfpunkte, die im zugeordneten Testfall gespeichert wird. Diese Zusammenfassung wird jedoch unter manchen Umständen nicht aktualisiert, was Inkonsistenzen und veraltete Testdaten verursacht.
- Anforderung: Die Konsistenz der Testfalldaten muss gewährleistet sein. Auf eine manuelle Aktualisierung der Daten darf man sich nicht verlassen.

4.14 Überflüssige Formularfelder

- Ursache: Die ClearQuest-Standardformulare wurden nicht von unnötigen Feldern bereinigt. Diese werden nicht verwendet und überladen die Ansicht.
- Anforderung: Es sollten nur die in der Praxis verwendeten Formularfelder und Daten angezeigt werden.

4.4.2 ITT-Testmanagement

4.15 Fehlende einheitliche und gute Dokumentation

- Ursache: Die Dokumentation ist nicht wie sonst üblich im Wiki zu finden, sondern in einem PDF. Sie ist zudem auf einem zu niedrigen Niveau. Es wird z.B. der Login-Prozess erklärt. Welche Übersicht wofür verwendet werden kann ist aber nirgends zu finden.
- Anforderung: Die Dokumentation muss an den Testprozess angelehnt werden und Informationen über Begrifflichkeiten beinhalten. Sie sollte wie die anderen Dokumentationen direkt im Wiki zu finden sein.

4.16 Umständliche Bedienung der Oberfläche

- Ursache: Die Anwendung wird zur Unterstützung des manuellen Tests verwendet und ist dafür nicht ausgelegt.
- Anforderung: Ein Werkzeug zur Unterstützung der manuellen Tests auf Basis der ClearQuest-Datenbank.

4.17 Konsistenz bei den Seitennamen

- Ursache: Verschiedene Seitennamen sorgen für Verwirrung bei der Navigation durch die Anwendung.
- Anforderung: Klare eindeutige Namensgebung der einzelnen Seiten und deren Links.

4.18 Feature-Orientierung

- Ursache: Das ITT ist eine Ansammlung von Features, die auch als Ansammlung präsentiert wird. Der Ursprung liegt aber in der Ergänzung von Aufgaben, die ganz konkret als Ziele aufgezählt werden können.
- Anforderung: Die Übersichtsseite muss die Ziele, für die das ITT entwickelt wurde, übersichtlich als solche darstellen. Eine optische Einteilung (Kategorien) in unterschiedliche Benutzergruppen ist wünschenswert.

4.19 Skalierbarkeit der Darstellung

- Ursache: Es gibt Darstellungen, die das dreifache einer 19”-Bildschirmbreite an Platz einnehmen.
- Anforderung: Die Daten müssen so angezeigt werden, dass sie auf eine Bildschirmseite passen.

4.4.3 Functional Tester

4.20 Mängel in der Dokumentation

- Ursache: Fehlende Vorschriften und Anleitungen führen zu schlecht wartbaren und uneinheitlichen Testskripten.
- Anforderung: Ein aktueller Styleguide mit Code- und Qualitätsrichtlinien für Testskripte sowie eine Schritt-für-Schritt-Anleitung für ein gutes Testskript sind notwendig.

4.21 Fehlende Kontrolle der nach Anforderung 20. eingeführten Richtlinien

- Ursache: Die Testskripte werden geschrieben und ohne Kontrolle verwendet, was zu uneinheitlichen, teilweise schlecht wartbaren Skripten führt.
- Anforderung: Eine Kontrolle (z.B. ein Review) der Umsetzung der Richtlinien für Testskripte.

4.4.4 soapUI

4.22 Mergeprobleme

- Ursache: Große XML Dateien sind quasi nicht mergebar und sorgen für große Verzögerungen.
- Anforderung: Dateien müssen mergebar sein oder es muss ein Zeitplan zur Bearbeitung der Dateien existieren.

4.23 Riesige globale Properties-Datei

- Ursache: Für alle Projekte gibt es eine riesige Properties-Datei die unübersichtlich und schlecht wartbar ist.
- Anforderung: Prüfen, ob es eine bessere Lösung gibt.
- Bemerkung: Das liegt nicht mehr im Rahmen unserer Fachstudie, soll aber nicht gänzlich unerwähnt bleiben.

4.4.5 Manual Tester

4.24 Fehlende Dokumentation

- Ursache: Mitarbeiter verwenden Manual Tester sehr unterschiedlich.
- Anforderung: Dokumentation zu Manual Tester muss angelegt werden.

4.25 Ineffizient und langsam

- Ursache: Durch schlechte Usability und lange Wartezeiten ist die Arbeit mit Manual Tester sehr ineffizient.
- Anforderung: Werkzeug zur Unterstützung manueller Tests mit kurzen Antwortzeiten und Copy & Paste Möglichkeit.

4.26 Schlechte Usability

- Ursache: Abgeschnittene Texte bei Buttons oder Dropdownmenüs sowie unklare Symbole.
- Anforderung: Richtige vollständige Beschriftung von Bedienelementen.

4.27 Unzufriedenheit der Mitarbeiter

- Ursache: Die Mitarbeiter sehen keinen Vorteil von Manual Tester gegenüber Excel.
- Anforderung: Werkzeug mit erkennbarem Vorteil und echter Unterstützung des manuellen Tests.

4.28 Fehlende Lizenzen

- Ursache: Manuelle Tests werden auf Papier ausgeführt, da zu wenige Lizenzen zur Verfügung stehen.
- Anforderung: Genügend Lizenzen für das Werkzeug zur Unterstützung der manuellen Tests.

4.4.6 Bestehendes

Generell ist es sehr wichtig, eine konkrete Auswahl an Werkzeugen zu haben und deren Verwendung obligatorisch zu machen. Das stärkt die Fähigkeit zur Zusammenarbeit und die Kompatibilität der Arbeit unterschiedlicher Mitarbeiter. Änderungen an der Auswahl sollten weiterhin daran festhalten.

ClearQuest

Auch wenn ClearQuest nicht das perfekte Testfallverwaltungswerkzeug ist, so ist es weder sinnvoll noch machbar auf ein anderes proprietäres Werkzeug umzusteigen. Die riesigen Datenmengen lassen sich nicht ohne Weiteres automatisiert portieren. Manuelles Portieren ist zu aufwändig. Bei einer webbasierten Eigenentwicklung eines neuen Werkzeugs wäre die Portierung noch am einfachsten, allerdings ist der Aufwand durch die Eigenentwicklung dennoch groß. Zwei Alternativen für einen ClearQuest-Ersatz werden im letzten Kapitel erläutert.

ITT Testmanagement

Auch wenn dieses Werkzeug überwiegend Schwächen von ClearQuest kompensiert, so ist es für die bestehende Situation eine mehr als nützliche Hilfe, die Zeit und Mühe spart. Bleibt man bei ClearQuest als Testfallverwaltung, so ist es sinnvoll auch am ITT Testmanagement festzuhalten und es weiter zu verbessern.

Functional Tester

Functional Tester ist ein sehr gutes Werkzeug zur Automatisierung von GUI-Tests. Es haben sich keine Schwächen gezeigt, die einen Umstieg auf ein anderes Werkzeug rechtfertigen würden. Anforderungen gehen daher nur in die Richtung der einheitlichen Verwendung und der Anbindung an die anderen Werkzeuge.

soapUI

In den Interviews hat sich gezeigt, dass soapUI sehr nützlich und unverzichtbar ist. Trotz der groben Schwächen wird es sehr geschätzt. Wie bei ClearQuest wäre ein Wechsel auf ein anderes Werkzeug mit sehr großem Aufwand verbunden.

Gut ist auch, dass eine Schnittstelle für den Functional Tester nachimplementiert wurde.

5 Ideallösung

In diesem Kapitel skizzieren wir eine ideale Testumgebung, die die gefundenen Probleme lösen würde. Der Aufwand und damit die Kosten spielen in der Ideallösung keine Rolle. Sie bildet die Grundlage für die realistisch umsetzbare Lösung im folgenden Kapitel.

5.1 Organisation und Prozess

Für einen Lösungsvorschlag bietet es sich an, die Organisation und den Prozess gemeinsam zu betrachten, da für die Verbesserungen im Prozess erst die Voraussetzungen in der Organisation geschaffen werden müssen.

Grundlegend schlagen wir drei Veränderungen vor, wovon eine die Optimierung der gesamten Prozessqualitätssicherung (Prozess-QS) beinhaltet.

5.1.1 Mehr Mitarbeiter zum Testen

(Anf. 2.2, 3.2, 3.4) Die optimale Anzahl an Mitarbeitern ist schwer objektiv zu ermitteln. Wir wissen aber, dass es momentan zu wenige sind. Es sollten so viele eingestellt werden, dass keine wichtigen Aufgaben mehr ausgelassen werden müssen.

5.1.2 Lizenzen

(Anf. 4.28) Es müssend genügend Lizenzen von jeder Software vorhanden sein, sodass kein Leerlauf durch Lizenzmangel entsteht.

5.1.3 Prozess-QS

(Anf. 2.1, 3.2) Es muss eine klare Trennung zwischen Entwicklern, Testern (Produkt-QS) und den Personen, die Prozessentwicklung betreiben und Infrastruktur zur Verfügung stellen (Prozess-QS), geschaffen werden. Diese Rollen konzentrieren sich ausschließlich auf ihre zugeteilte Aufgabe.

Aufgaben der Prozess-QS

Die Prozess-QS benötigt eine konkrete Aufgabenliste. Wir bieten keine vollständige Liste mit allen Aufgaben der Prozess-QS, sondern bieten Vorschläge für zusätzliche Aufgaben zu den bisherigen.

Zu den Angaben der Wartungsintervalle: Diese spiegeln subjektive Einschätzungen der Änderungshäufigkeit sowie Wichtigkeit der Aktualität wieder. Sie sind daher nur ein Vorschlag, der ohne Weiteres angepasst werden kann.

(Anf. 3.10)

- Einrichtung eines zentralen Wikis für alle hier aufgelisteten Dokumente. Weitere Wikis werden damit überflüssig und sind nicht länger erlaubt.

- Erstellung einer Aufgabendokumentation für die Prozess-QS. Wartung dieser Dokumentation in einem jährlichen Intervall.

(Anf. 3.7)
- Erstellung eines Begriffslexikons für sämtliche FleetBoard-spezifischen Begriffe und Abkürzungen. Verschiedene Begriffe für eine Sache sollten vermieden werden. Jeder sollte daran mitarbeiten, aber die Prozess-QS prüft und verbessert deren Qualität 1-2 Mal jährlich.

(Anf. 2.1)
(Anf. 3.1)
- Erstellung je einer Dokumentationsseite zur Organisation, zum Entwicklungsprozess und zum Testprozess . Das Ziel dieser Dokumentation ist, einen Überblick zu liefern und neuen Mitarbeiten den Einstieg zu erleichtern. Dafür ist ein geringer Detailgrad notwendig. Wartung dieser Dokumentation in einem jährlichen Intervall oder bei Änderungen.

(Anf. 3.9, 3.11, 4.3)
- Erstellung einer präzisen Dokumentation zum Entwicklungsprozess sowie zum Testprozess. Der gesamte Prozess wird in seine Einzelteile zerlegt und Schritt für Schritt mit den wichtigen Details befüllt. Richtlinien, sowie eine Übersicht über verschiedene Testarten werden ebenfalls hier untergebracht. Wartung dieser Dokumentation in einem sechsmonatigen Intervall.

(Anf. 4.1, 4.15, 4.20, 4.24)
- Erstellung von Dokumentationsrichtlinien bzw. einem Schema für einzelne Werkzeuge sowie der Benennung je eines Beauftragten, der dieses Werkzeug dokumentiert. Bei den wichtigsten Werkzeugen kann das ebenfalls die Prozess-QS sein. Wartung der Richtlinien/Schemata sowie Prüfung derer Einhaltung und Qualität in einem jährlichen Intervall.

(Anf. 3.8)
- Kontinuierliche Verbesserung der Infrastruktur durch Ermitteln von automatisierbaren Zwischenschritten, die nicht zur eigentlichen Aufgabe des Testens gehören.

Benötigte Richtlinien in der Prozessdokumentation

In der detaillierten Prozessdokumentation wurde die Unterbringung von Richtlinien erwähnt. Folgende Richtlinien ergeben sich aus unseren Anforderungen.

(Anf. 3.3)
- Eine einheitliche Firmenweite Versionierungsrichtlinie, die für jedes Team verständlich ist und teamübergreifend funktioniert.

(Anf. 3.5, 4.6)
- Es gibt folgende Vorgaben für produktive Dokumente: notwendiger Inhalt, Detailgrad und erwarteter Umfang.

(Anf. 4.11)
- Unterscheidung verschiedener Testarten mit jeweils einer grundsätzlichen Strukturierung für Vorbedingung, Aktion und Nachbedingung. Diese Strukturierung dient als Grundlage für Vorlagen (Templates).

Aufbau einer Werkzeugdokumentation

Eine Dokumentation zu einem eingesetzten Werkzeug muss folgende Aspekte abdecken.

- Einrichtung und Start des Werkzeugs
- Grundlegende Funktionsweise
- Verweis auf die Richtlinien zur Strukturierung der Daten in der Prozessdokumentation
- Funktionsweise der Anbindung an andere Werkzeuge
- FAQ

5.2 Werkzeuge

Durch die umfassende Testverwaltung werden folgende bisher unterschiedene Werkzeugarten obsolet:

- Spezifikation
- Manuelle Tests
- Planungs- und Auswertungshilfe

Übrig bleiben, neben der Testverwaltung, lediglich Werkzeuge zur Testautomatisierung. Die Anforderungserhebung könnte man theoretisch auch mit unterbringen, allerdings liegt diese Überlegung nicht im Rahmen unserer Fachstudie.

5.2.1 Testverwaltung

Ursprünglich wurde von einer Testfallverwaltung gesprochen. In dem Lösungsvorschlag heißt es nun Testverwaltung. Das ist dadurch begründet, dass eine Testfallverwaltung für sich keinen Nutzen hat, wenn sie mit den verwalteten Daten in keiner Weise arbeiten kann. Sämtliche Übersichts- und Bearbeitungswerkzeuge die es momentan zusätzlich gibt, gehören in der idealen Lösung in die Testverwaltung und sind eng mit einander verknüpft. Ob man für verschiedene Benutzergruppen getrennte Werkzeuge erstellt, die auf den selben Daten arbeiten und eventuell nur Teilansichten gemeinsam haben, ist für unseren Vorschlag nicht relevant.

(Anf. 3.13, 4.2, 4.4, 4.13, 4.14, 4.16, 4.17, 4.18, 4.19, 4.24-28)

Bei einer idealen Lösung für ein Werkzeug spielt es keine Rolle, ob es sich um eine gekaufte Software handelt, oder um eine Eigenentwicklung. Es wird nicht beachtet, was ein Feature kosten würde, sondern nur was für den Zweck des Werkzeugs die optimale Lösung wäre. Es ist daher ohne Weiteres möglich, dass dieser Vorschlag unwirtschaftlich scheint bzw. ist.

Durch die Integrierung mehrerer bisheriger Werkzeuge in einer neuen einheitlichen Lösung, sind nebenstehende Anforderungen automatisch erfüllt.

Rollen

Um verschiedenen Benutzergruppen jeweils die passenden Ansichten zur Verfügung stellen zu können müssen zunächst die Benutzergruppen ermittelt werden. Eine Person kann für mehr als eine Aufgabe zuständig sein, daher nehmen wir die Rolle als atomares Element. Jede Rolle hat klare Ziele, die mit dem Werkzeug für die Testverwaltung erledigt werden sollen. Jeder Mitarbeiter kennt seine Rollen bei FleetBoard und wird daher keine Schwierigkeiten haben die passende Rolle im Werkzeug auszuwählen.

Die Rollen leiten sich aus dem Testprozess bei FleetBoard ab (siehe Abbildung 3.1). Die folgenden Betrachten wir:

- Spezifizierer
- Testfallspezifizierer
- Testautomatisierer
- Testplaner
- Tester
- Testauswerter

Im Folgenden werden die Rollen beschrieben und Ziele definiert.

Spezifizierer

Der Spezifizierer setzt Anforderungen in ein Gesamtkonzept um und formuliert daraus die Spezifikation. Der Einfachheit halber bekommt jedes Feature eine Spezifikation wodurch die kleinste Einheit in der Datenstruktur des Spezifizierers definiert ist. Spezifikationen werden mit Anforderungen verknüpft.

Zur Übersicht über die verbleibende Arbeit wird eine Filterung nach offenen Anforderungen benötigt.

Testfallspezifizierer

Der Testfallspezifizierer erstellt aus den Spezifikationen Testfälle. Mit Testfall ist ein Testfall im Sinne der Literatur gemeint. Die Ansicht zu den Daten baut auf der des Spezifizierers auf, ebenso wie die Daten selbst. Testfälle werden durch den Testfallspezifizierer jeweils einer Spezifikation zugeordnet.

(Anf. 4.9, 4.10, 4.11) Da die Eingabe von Testfällen für einige Gebiete Routinearbeit ist und viele ähnliche Testfälle angelegt werden müssen, will der Testfallspezifizierer eine gute Unterstützung für die Arbeit mit der Tastatur, eine Übersicht über mehrere Testfälle und deren Abhängigkeiten gleichzeitig, sowie Vorlagen für standardisierte Inhalte. Zu beachten ist, dass unterschiedliche Kategorien von Tests unterschiedliche Kategorien von Vorlagen benötigen. Ein Mitarbeiter wechselt aber nicht täglich sein Arbeitsgebiet und möchte daher nur eine persönliche Liste von Kategorien angezeigt bekommen. Viele ähnliche Testfälle möchte man zudem dadurch erzeugen können, dass vorhandene Testfälle kopiert, an die richtige Position verschoben und angepasst werden.

(Anf. 4.7) Beachtet werden muss ebenfalls, dass bei manchen Produkten unterschiedliche Versionen unterschiedlich getestet werden müssen. Dazu werden optionale Konfigurationen im Sinne einer Verknüpfung des Testfalls mit der Zielumgebung angeboten.

Bei den umfangreichen Anforderungen des Testfallspezifizierers darf allerdings niemals die, bei den anderen Rollen bereits geforderte, Übersicht durch Auswahl von Produkt und Versionsnummern, sowie Filterung nach bereits erledigter Arbeit, unterpriorisiert werden.

Testautomatisierer

Ähnlich wie der Implementierer benötigt der Testautomatisierer lediglich eine Übersichtliche Anzeige der für ihn momentan anstehenden Arbeit. Er arbeitet allerdings auf der Ebene der Testfälle und weist ihnen nach getaner Arbeit je ein Testskript zu. Er muss, falls notwendig, zwischen unterschiedlichen Konfigurationen unterscheiden und diese, falls der Testfallspezifizierer das nicht beachtet hat, diese selbst anlegen können.

Zur Übersicht über die verbleibende Arbeit wird eine Filterung nach offenen Automatisierungen benötigt.

Testplaner

Ein Testplaner legt Testpläne an, die Produkte und Versionsnummern aus dem vorhandenen Datenbestand selektieren und bündeln. Zur Verplanung der Mitarbeiter trägt er deren vorhandene Arbeitszeit ein und verteilt diese Zeit in einem Testplan auf die zu testenden Testfälle. Damit dies möglich ist werden Testfälle vorher mit einem Schätzwert für die Ausführungsdauer versehen. Eine automatisierte Unterstützung der Zeiterfassung nimmt dem Testplaner nervige Arbeit der Aufwandserfassung ab.

Ist ein Testplan erstellt, wählt der Testplaner ihn als momentan aktiven Testplan aus und gibt damit die Arbeit für die Tester vor.

Für die Inhalte eines Testplans (welche Teilmenge an Tests wird durchgeführt) werden Versionsnummern von Produkten ausgewählt. Damit man aber zuordnen kann für welche Version

getestet wird, legt der Testplaner außerdem eine Zielversion fest. Diese Zielversion ist der Proband und somit Grundlage aller durchgeführten Tests. Eine Version eines Produktes entspricht bei Software in der Regel einem Stand in der Versionsverwaltung auf Dateiebene (SVN). Damit kann ein Testplan später erneut mit gleichen Ergebnissen getestet werden. Bei Fleetboard entspricht so ein testbarer Stand einer Freigabe des Betriebsteams, die alle Versionen der Komponenten der FleetBoard-Software enthält.

Während des Tests muss der Testplaner die Auslastung der Tester kontrollieren und notfalls anpassen können. Ebenfalls möchte er sehen, wie weit die Dauer der Tests von den Schätzungen abweicht.

Tester

Der Tester möchte eine Übersicht über momentan noch offene Aufgaben (auszuführende Tests). Neben einer übersichtlichen Anzeige der von Testfallspezifizierer und Testautomatisierer produzierter Daten, benötigt der Tester diverse Filtermöglichkeiten um z.B. Konfigurationen in seiner gewünschten Reihenfolge zu testen.

Eine Zusammenfassung über Umfang, geschätzter Dauer und verbleibender Arbeit lässt ihn die Arbeit besser einteilen und das Ende besser abschätzen.

(Anf. 4.12) Sämtliche Tests werden aus der gleichen Ansicht heraus gestartet. Manuelle Tests werden – visuell unterstützt – Schritt für Schritt durchgeführt. Eine Ermittlung der tatsächlich benötigten Zeit gibt dem Tester Informationen über sein Arbeitstempo und bietet die Möglichkeit, Schätzwerte durch gemessene Werte anzupassen.

Am Ende eines Testdurchlaufs bekommt der Tester eine knappe sowie zusätzlich eine umfassende Zusammenfassung der Testergebnisse. Eine Filterung nach fehlgeschlagenen Tests vereinfacht zudem die Arbeit der Fehlerprotokollierung in einem Bugtrackingsystem.

Testauswerter

Zur Auswertung der Tests benötigt der Testauswerter eine Übersicht über alle durchgeführten Tests zu einem Testplan. Während des Tests möchte der Testauswerter übersichtlich sehen was noch alles offen ist und welche Mitarbeiter daran arbeiten.

Mit dem Fokus auf die gefunden Fehler, möchte der Testauswerter diese gefiltert und übersichtlich mit Details aus den Testergebnissen angezeigt bekommen. Liegt der Fokus auf die eingeteilte Zeit der Tester, dann möchte er dafür eine Übersicht, die ihm zeigt, welche Tester unterfordert oder überlastet waren. Bei einem Fokus auf den Test selbst, möchte der Testauswerter berechnete Daten wie Zahl der durchgeführten Tests, welcher Teil davon Manuell, wie lange haben die Tests gedauert, was war die geschätzte Zeit, inwiefern wurde diese angepasst.

Anforderungen

Aus den Beschreibungen der Rollen, zusammen mit den Anforderungen, die sich aus den bisherigen Analysen ergeben haben, ergeben sich folgende Anforderungen an die Ideallösung einer Testverwaltung.

Spezifizierer

Eine Spezifikation wird als ein Element gespeichert und dargestellt. Solch ein Element besitzt ebenfalls einen Umsetzungsstatus, welcher initial auf „Offen“ gesetzt wird.

Der Spezifizierer legt die Position seiner Spezifikation in der Baumstruktur fest. Er hat die Möglichkeit, nach Produkten, Versionen und Umsetzungsstatus der Anforderungen zu filtern, was ihn dabei unterstützt, offene Anforderungen zu erkennen. Anforderungen werden als Ordner eingebunden, sind aber deutlich als Anforderung zu erkennen. Einer Anforderung können

beliebig viele Spezifikationen zugeordnet werden, denn es ist oft so, dass Anforderungen sehr grob sind und in der Spezifikation aufgeteilt werden müssen. Nachdem der Spezifizierer eine Anforderung fertig hat, kann er dieser den Umsetzungsstatus „spezifiziert“ geben, welcher für den Produktmanager sichtbar ist.

Testfallspezifizierer

Der Testfallspezifizierer bekommt er eine Ansicht mit der Baumstruktur des Spezifizierers. Eine Spezifikation wird als Ordner dargestellt und sichtbar als solche gekennzeichnet. Ein Testfall wird als Element gespeichert und dargestellt, welches einer Spezifikation zugewiesen werden
(Anf. 4.5) kann und ebenfalls einen initial auf „Offen„ gesetzten Umsetzungsstatus hat. Testfälle innerhalb einer Spezifikation haben eine Reihenfolge, welche in diesem Baum beliebig festgelegt werden kann, und können von Vorgängern abhängig sein. Zusätzlich zum Umsetzungsstatus gibt es einen Automatisierungsgrad, welcher bei der Testfallspezifikation angegeben werden kann. Initial hat er den Wert „Automatisiert (nicht implementiert)“. Für manuelle Tests kann dieser Grad direkt bei der Testfallspezifikation auf „Manuell“ gesetzt werden. Durch die kom-
(Anf. 3.13) plette Integrierung der manuellen Tests in dieses Werkzeug ist die Anforderung automatisch erfüllt.

Da es vorkommt, dass neue Versionen eines Produkts einen Testfall überflüssig machen, gibt es im Testfall die Möglichkeit die Eigenschaft „Veraltet seit“ mit einer Version zu versehen. Für Versionen nach dieser kann dann keine neue Konfiguration mehr angelegt werden. Veraltete Testfälle werden in der Baumstruktur als solche markiert.

(Anf. 4.7) Werden für verschiedene Produkte unterschiedliche Konfigurationen benötigt, so gibt es die Möglichkeit einem Testfall beliebig viele Konfigurationen zuzuordnen (ebenfalls im Baum dargestellt). Eine Konfiguration kann Eigenschaften des Testfalls überschreiben und ist mit einem Produkt und einer Version verknüpft. Der Übergeordnete Testfall dient in dem Fall dazu, das allgemeine Vorgehen bei dem Test zu dokumentieren. Die Konfigurationen passen dieses Vorgehen für eine Version eines Produktes an. Solange es nur eine Konfiguration für einen Testfall gibt, wird dieses Element nicht benötigt, sondern es reicht aus nur mit dem Testfall selbst zu arbeiten.

Zur Übersicht kann die Baumstruktur nach Produkt, Version sowie Umsetzungsstatus der Anforderungen und Spezifikationen gefiltert werden. Wird nach Produkt und Version gefiltert, so werden sämtliche Konfigurationen von Testfällen ausgeblendet und in der Ansicht des Testfalls direkt der Inhalt der gefilterten Version angezeigt. Die Tatsache, dass es weitere Konfigurationen zu dem Testfall gibt, wird durch eine kurze Information sichtbar gemacht.

Sind zu einer Spezifikation vorerst alle Testfälle erstellt, so kann der Umsetzungsstatus der Spezifikation auf „Testfall spezifiziert“ gesetzt werden. Sind alle Spezifikationen innerhalb einer Anforderung testfallspezifiziert, so wird der Status der Anforderung ebenfalls auf „Testfall spezifiziert“ gesetzt.

Ein Testfall hat einen Umsetzungsstatus, weil ein manueller Test im Normalfall nach der Spezifikationen noch verfeinert werden muss und weil ein automatisierbarer Testfall evtl. noch automatisiert wird. Erst wenn fest steht, dass der Testfall so getestet werden soll, wird der Status auf „Abgeschlossen“ gesetzt.

Testautomatisierer

Der Testautomatisierer bekommt eine Ansicht mit einer Baumstruktur gleich der des Testfallspezifizierers, mit dem Unterschied, dass der Testautomatisierer Testfälle nicht bearbeitet, sondern primär anschaut. Er kann aber ein Testskript mit einem Testfall oder einer Konfiguration verknüpfen und den Automatisierungsgrad auf „Automatisiert“ setzen. In dem Fall wird der Umsetzungsstatus automatisch auf „Abgeschlossen“ gesetzt. Für den Fall, dass noch keine Konfiguration angelegt wurde, aber eine benötigt wird, kann der Testautomatisierer selbst eine

Konfiguration anlegen.

Testplaner

Der Testplaner benötigt folgende Ansichten:

- 1. Übersicht über die aktuell vorhandenen Testpläne mit dem neusten an erster Stelle. Der aktive Testplan wird gut sichtbar markiert und kann hier gesetzt werden. Neue Testpläne können von hier aus angelegt werden.
- 2. Einstellungsansicht zu einem Testplan. Produkte und Version werden hier ausgewählt. Es gibt eine Übersicht über die Tester und deren verfügbare Zeit.
- 3. Zuordnung der Testfälle zu den Testern mit Baumstruktur der Testdaten die entsprechend des Testplans gefiltert werden.

Zu Ansicht 1:

Um den Testern die Auswahl des Testplans abzunehmen, legt der Testplaner fest welcher Plan aktuell getestet werden soll. Wird ein neuer Testplan angelegt, wird vorgeschlagen diesen als den aktuellen Testplan zu speichern.

Zu Ansicht 3:

Zur Zuordnung der Testfälle erhält der Testplaner eine Ansicht mit einer Baumstruktur, welche alle relevanten Testfälle und Konfigurationen anzeigt. Konfigurationen werden nur explizit angezeigt, wenn es für einen Testfall mehr als eine gibt die zum Testplan passt. Ansonsten wird sie anstelle des Testfalls angezeigt. Gibt es offene Anforderungen, Spezifikationen oder Testfälle, dann werden diese gut sichtbar markiert.

Zur Verplanung von Arbeitskräften benötigt der Testplaner Schätzwerte für die Dauer einer Durchführung. Diese Schätzwerte werden in den Testfällen gespeichert und können von Konfigurationen überschrieben werden. Zusätzlich wird die Anzahl der bisher durchgeführten Tests gespeichert um Schätzwerte besser anpassen zu können. Gibt es noch keine Daten von vorherigen Durchführungen, dann bekommt ein manueller Testfall den Anfangswert 15 Minuten. Ein automatisierter Testfall 5 Minuten. Zusammen mit der Zahl der bisherigen Durchführungen kann der Schätzwert bei jeder Durchführung an die durchschnittliche Dauer angepasst werden und nähert sich mit der Zeit einem praxisnahen Wert.

Ziel der Testplanung ist, einen Plan zu erstellen, der Testern Testfälle zuordnet. Grundlage für die Planung ist die verfügbare Arbeitszeit des Testers welche mündlich eingeholt wird und bei dem jeweiligen Tester gespeichert werden kann, sowie die geschätzte Dauer von Testfällen. Daher wird direkt in der Baumstruktur die geschätzte Dauer mit angezeigt. Elemente die kein Blatt im Baum sind, zeigen die summierte Dauer all ihrer Kindelemente. Einem ausgewählten Element kann nun ein Tester zugewiesen werden. Die Auswahl für die Tester zeigt die bereits zugeteilte geschätzte Testdauer, sowie die verbleibende Zeit mit an, sodass auf einen Blick klar ist welcher Tester noch Zeit für weitere Tests hat. Tester mit ausreichend freier Zeit werden farblich hervorgehoben.

Da die Einteilung der Testfälle nicht auf Testfallebene geplant werden muss, sondern auch ganze Spezifikationen und Anforderungen ausgewählt werden können, kann der Testplaner ganze Blöcke gleichzeitig zuteilen und spart dadurch Aufwand.

Zur besseren Unterstützung der Planung kann in der Baumstruktur nach noch nicht zugeteilten Testfällen gefiltert werden. Zudem ist es möglich offene Testfälle auszublenden.

Ein Testplan wird immer in Zusammenhang mit einer Version gespeichert. Eine Historie vorheriger Testdurchführungen bleibt daher erhalten. Dafür muss ein Testplan neben den gewählten

Produkten und Versionen die Zuordnung von Testfall und Tester speichern, sowie die verfügbare Zeit der Tester. Schätzwerte der Durchführungsdauer sind nicht abhängig von einem Testplan. Anhand der Historie ist es möglich einen vorherigen Testplan als Grundlage für einen neuen Testplan zu verwenden. Der neue Testplan muss dann nur noch um die neuen Testfälle ergänzt werden und Zuordnungen entsprechend der aktuell verfügbaren Zeit der Tester angepasst werden.

Tester

Ein Tester bekommt als erstes eine Baumstruktur, die passend zum aktuellen Testplan und seinen zugewiesenen Testfällen gefiltert wird. Der Automatisierungsgrad wird im Baum farblich hervorgehoben, damit der Tester einen ersten Eindruck bekommt. Es kann auch danach gefiltert werden. Zusätzlich kann nach Konfigurationen gefiltert werden, sofern denn mehrere zur Auswahl stehen. Zur aktuellen Auswahl wird stets eine übersichtliche, für den Test angepasste, Zusammenfassung des Inhalts angezeigt. Enthalten sind z.B. Anzahl der Testfälle und Gesamtdauer der Testfälle.

Aus dieser ersten Ansicht heraus können Testdurchläufe gestartet werden. Dabei gibt es folgende Möglichkeiten:

- Komplettdurchlauf des gesamten Testplans
- Komplettdurchlauf fortsetzen
- Durchlauf des gewählten Teilbaums

Anschließend erscheint eine neue Ansicht, bei der es noch folgende Optionen bezüglich Automatisierungsgrad und Konfigurationen gibt (falls mehr als eine Konfiguration vorhanden):

- Filterung nach Automatisierungsgrad
- Falls automatisiert und nicht automatisiert: Wahl eines Modus
  - Alle manuellen Tests zuerst
  - Reihenfolge behalten
  - Dynamisch: Während ein manueller Test durchgeführt wird, kann ein nicht abhängiger automatisierter Test bereits gestartet werden.
- Filterung nach Konfiguration
- Falls nicht nach Konfiguration gefiltert: Wahl eines Modus für Konfigurationen
  - Kompletter Durchlauf einer Anforderung pro Konfiguration
  - Kompletter Durchlauf einer Spezifikation pro Konfiguration
  - Kompletter Durchlauf nur für die erste Konfiguration. Anschließend minimaler Durchlauf.

Im nächsten Schritt wird eine lineare Zusammenfassung des Testdurchlaufs angezeigt. Alle Testfälle und Konfigurationen werden in der Reihenfolge angezeigt wie sie tatsächlich durchgeführt werden sollen. Der Automatisierungsgrad wird farblich hervorgehoben. Anforderungen und Spezifikationen werden unauffälliger gestaltet, da sie hier eher unwichtig sind. Es gibt eine Zusammenfassung der Gesamtdauer, des gesamten Automatisierungsgrades, der Anzahl der Unterbrechungen von automatisierten durch manuelle Tests, falls zutreffend der Einsparung durch paralleles starten von manuellen und automatisierten Tests. Unterbrechungen von automatisierten Tests durch manuelle Tests werden zudem zusammengefasst und zur entsprechenden Stelle in der eigentlichen Ansicht verlinkt.

Nach dem Betrachten der Zusammenfassung wird der Testdurchlauf gestartet. Die Zeit für jeden Testfall wird nebenher ermittelt und gespeichert. Manuelle Tests werden durch jeweilige Anzeige einer Zusammenfassung des Testfalls sowie dessen Umgebung (Anforderung, Spezifikation, Konfiguration, vorheriger und nächster Testfall) und der Möglichkeit das Ergebnis abzuspeichern unterstützt. Eine Fortschrittsanzeige mit Restzeitanzeige verschafft dem Tester
(Anf. 3.12) einen besseren Überblick. Ergebnisse werden mit einem Testfall oder, falls vorhanden, einer Testfallkonfiguration verknüpft. Teil des Ergebnisses ist eine Veränderung des Umsetzungsstatus des Testfalls oder der Konfiguration. Er kann auf „Getestet“ oder „Getestet mit Fehler“ geändert werden, was automatisch anhand des Testergebnisses geschieht.

Ein Durchlauf kann unterbrochen werden. Es ist nicht notwendig, den Durchlauf mit Einstellungen und aktuellem Stand explizit speichern zu können, da die Anwendung sich Einstellungen des Benutzers grundsätzlich merken sollte. Ein Fortsetzen kann so ohne weiteres neu berechnet werden.

Nach dem Durchlauf wird eine Testauswertung mit allen Ergebnissen angezeigt und abgespeichert.

Testauswerter

Zur Auswertung der Tests benötigt der Testauswerter eine Übersicht über alle durchgeführten Tests zu einem Testplan. Es wird der vom Testplaner eingestellte aktuelle Testplan ausgewählt. Ein Wechsel auf andere Testpläne ist aber möglich.

Die Übersicht enthält eine Baumstruktur mit allen enthaltenen Anforderungen, Spezifikationen, Testfällen und Konfigurationen. Es kann nach Produkt und Testergebnis gefiltert werden. Zu der jeweiligen Auswahl im Baum wird stets die passende Zusammenfassung aus dem Testdurchlauf angezeigt.

Migration

Die Migration der Daten für die Testverwaltung muss bisherige Daten aus drei Werkzeugen zusammenführen. Daten von Focal Point und Requisite Pro müssen von Hand migriert (neu eingetragen) werden. Dabei sollte eine überarbeitete Datenstruktur bereits beachtet werden. Für die Migration der Daten aus ClearQuest kann ein grafisches Hilfswerkzeug erstellt werden welches die Migration plant und den Benutzer dabei unterstützt. Anhand dieses Plans kann die Migration innerhalb von wenigen Minuten automatisiert durchgeführt werden um einen Reibungslosen Übergang im Betrieb zu ermöglichen.

5.2.2 Automatisierte Tests

An die Werkzeuge für Testautomatisierung haben wir im Rahmen der Fachstudie eigentlich keine Anforderungen. Relevant ist aber die Anbindung an die Testverwaltung.

Bei der Ideallösung gibt es an Werkzeuge zur Testautomatisierung folgende Anforderungen:

Datenstruktur

Durch Richtlinien wird eine einheitliche Datenstruktur eingehalten. Das Vorkommen von Versionsnummern in der Datenstruktur ist auch hier zu vermeiden.

Implementierung

Code und Kommentare die aus den Daten der Testverwaltung generiert werden können, sollen auch generiert werden.

Aufruf

Der Aufruf eines Testskripts ist durch die Testverwaltung im Rahmen einer Testdurchführung möglich. Die Auswahl der zu testenden Testfälle ist auf genau dieser Ebene durch die Testverwaltung möglich. Eine statische Bündelung von Tests in einem einzigen Skript ist nicht vorgesehen.

Auswertung

Nach einer Durchführung steht das Testergebnis, die Ausführungsdauer sowie in einem Fehlerfall eine möglichst genaue Fehlerbeschreibung zur Verfügung und kann durch die Testverwaltung abgespeichert werden.

Migration

Ein Austausch der Werkzeuge ist nicht gefordert, da das den Rahmen der Fachstudie sprengt. Eine Migration von Daten wird daher nicht besprochen. Da bisherige Daten aber sehr spezifisch für ein Werkzeug sind, ist es kaum möglich diese sinnvoll zu migrieren.

6 Wirtschaftlicher Lösungsvorschlag

Grundlage des wirtschaftlichen Lösungsvorschlags ist eine subjektive Bewertung aller Anforderungen nach Kosten und Nutzen.

Darauf aufbauend setzt sich der wirtschaftliche Lösungsvorschlag aus der Verbesserung der Organisation und des Prozesses, sowie der Verbesserung der Werkzeuge oder deren Austauschs zusammen. Im Bereich der Werkzeuge zeigen wir FleetBoard bewusst mehrere umsetzbare Lösungsvarianten auf und treffen keine Entscheidung, welche der Lösungen die beste ist. Wir machen Verbesserungsvorschläge zum Umgang mit den bestehenden Werkzeugen und erörtern die Vor- und Nachteile des teilweisen Austauschs bestehender Werkzeuge durch eine integrierte externe Lösung im Vergleich zu einer flexiblen Eigenentwicklung.

6.1 Bewertung der Anforderungen

Nicht jede unserer Anforderungen ist gleich wichtig. Außerdem gibt es starke Unterschiede in den geschätzten Umsetzungskosten. Daher haben wir alle Anforderungen rein subjektiv nach Nutzen und Kosten bewertet und dann eine sinnvolle Auswahl getroffen. Das Ergebnis ist folgende Liste.

- Hohe Priorität:
  - 2.1 Person, die sich den Aufgaben gemäß LL07, Kap. 13.1.3 widmet. Klar definierte Aufgaben und festgelegte Zeit zur Umsetzung.
  - 3.1 Dokumentation, die einen Überblick über den Testprozess vermittelt. Rollen und Artefakte bereits eingeführt.
  - 3.4 Zusätzliche Zeit zur Automatisierung von Tests. Am besten je Sprint eine fest definierte Menge.
  - 3.7 FleetBoard-weites Begriffslexikon. Verzicht auf Abkürzungen.
  - 3.11 Kundenakzeptanztests erfolgreich einführen.
  - 3.12 Exaktere Testergebnisse durch Anhängen an einen Prüfpunkt. Speicherung des Testlogs für Ursachenermittlung.
  - 4.5 Ordnung und Abhängigkeiten zwischen Testfällen
  - 4.11 Layout- und Inhaltsvorlagen zum Anlegen und Verändern von Testfällen
  - 4.20 Ein aktueller Styleguide mit Code- und Qualitätsrichtlinien für Testskripte sowie eine Schritt-für-Schritt-Anleitung für ein gutes Testskript sind notwendig
  - 4.21 Kontrolle (z.B. ein Review) der Umsetzung der Richtlinien für Testskripte
  - Die folgenden Anforderungen sind nur gemeinsam durch Ablösung von Manual Tester möglich:

        * 4.25 Werkzeug zur Unterstützung manueller Tests mit kurzen Antwortzeiten und Copy & Paste Möglichkeit.
        * 4.26 Richtige vollständige Beschriftung von Buttons usw.
        * 4.27 Werkzeug mit erkennbarem Vorteil und echter Unterstützung des manuellen Tests

    - Mittlere Priorität:
        - 2.2 Mehr Mitarbeiter für die Tests.
        - 3.2 Die benötigte Zeit abschätzen und ausreichend Mitarbeiter für die Testspezifikation und -automatisierung zur Verfügung stellen
        - 3.9 Qualitativ hochwertige Dokumentation, die ausreichend ins Detail geht um die tägliche Arbeit erledigen zu können, sich aber trotzdem auf das Wesentliche konzentriert.
        - 4.3 Klare Vorgaben zur Platzierung und Ordnung von Testfällen in ClearQuest und Überwachung deren Einhaltung. Die vorhandenen Daten müssen neu strukturiert werden.
        - 4.12 Testskripte und manuelle Tests müssen aus dem Testfallverwaltungswerkzeug gestartet werden können.
        - 4.14 In ClearQuest werden nur die in der Praxis verwendeten Formularfelder und Daten angezeigt.
        - 4.15 Die Dokumentation des ITT-Testmanagements muss sich an den tatsächlichen Aufgaben des Testbetriebs orientieren, Informationen über Begrifflichkeiten beinhalten und im Wiki zu finden sein
        - 4.17 Klare eindeutige Namensgebung der einzelnen Seiten des ITT-Testmanagements und deren Links
        - 4.18 Die Übersichtsseite des ITT-Testmanagements muss die Ziele, für die das ITT entwickelt wurde, übersichtlich als solche darstellen. Eine optische Einteilung (Kategorien) in unterschiedliche Benutzergruppen ist wünschenswert
        - 4.24 Dokumentation zu Manual Tester muss angelegt werden
        - 4.28 Genügend Lizenzen für das Werkzeug zur Unterstützung der manuellen Tests müssen vorhanden sein.

    - Niedrige Priorität: Alle Übrigen. Deren Einschätzung und Umsetzung überlassen wir FleetBoard. Die vollständige Liste mit allen Prioritäten ist als Anhang beigefügt.

6.2 Verbesserung von Organisation und Prozess

Unsere Ideallösung hierzu beinhaltet im Wesentlichen Finetuning an Organisation und Prozess und verursacht somit nur den Aufwand für die eigentliche Umstellung sowie zusätzliche Personalkosten. Teure Datenmigrationen fallen nicht an.

Daher schlagen wir vor, die Ideallösung für die Organisation und den Prozess, so wie sie formuliert ist, umzusetzen.

6.3 Verbesserungen im Umgang mit vorhandenen Werkzeugen

6.3.1 ClearQuest

ClearQuest bietet aus zwei Gründen das größte Potential für Verbesserungen. Zum einen ist es das zentrale Werkzeug der Testverwaltung, zum anderen kann es mit dem ClearQuest Designer an individuelle Bedürfnisse angepasst werden, wenn auch nur in begrenztem Maße. Es ist möglich, neue Datentypen anzulegen, diese mit Werten und Verknüpfungen zu anderen Datentypen auszustatten, sowie einfache Oberflächen für diese Typen zu erstellen.

Trotz dieser Möglichkeiten bleibt die Usability der erstellten Oberflächen weit hinter den theoretischen Möglichkeiten zurück. So ist bei den Vorschlägen zu ClearQuest zu überlegen, ob diese nicht besser mit Hilfe eines neuen Tools umgesetzt werden. Im Abschnitt 6.4 werden zwei Alternativen für ein neues Werkzeug vorgestellt.

6.3.2 Functional Tester

Erstellung von Code- und Qualitätsrichtlinen für Testskripte

Um das Problem der schwankenden Qualität von Testskripten in den Griff zu bekommen, empfehlen wir, dass das schwarze Team zusammen mit einigen Testskriptentwicklern Richtlinien für Functional Tester Skripte aufstellt. Diese Richtlinien sollten eine Liste von zu verwendenden und zu vermeidenden Strukturen enthalten, also eine Liste von „Do“s and „Don’t“s. Ebenso sollte die Unabhängigkeit von nicht einfach reproduzierbaren Testflotten vorgeschrieben werden. Darüber hinaus sollten alle Skripte eines Dienstes wenn möglich ohne Flottenwechsel auskommen und vom Tester wenig Insiderwissen verlangen.

Die Einhaltung dieser Richtlinien sollten beim Durchführen der Regressionstests vom schwarzen Team geprüft werden. Ein Regressionstester muss dann ein Skript als nachbesserungsbedürftig empfehlen und zum Entwickler zurückgeben können. Die Kosten einer zeitnahen Korrektur sind gering im Vergleich zu den Kosten, die entstehen, wenn ein Problem beim Test des nächsten Releases entdeckt werden.

6.4 Alternativen zu bestehenden Werkzeugen

Die Analyse hat ergeben, dass die Wahl der Werkzeuge nicht optimal war. Sie hat aber auch ergeben, dass ein Austausch eines Werkzeugs ohne automatisierte Migration nahezu ausgeschlossen ist, da bereits zu viel Arbeitszeit in die Erstellung der Daten geflossen ist. Zudem gab es vor drei Jahren bereits eine aufwändige Migration, die man nun ungern wiederholen möchte. Diesen Aspekt haben wir bei unserem Vorschlag beachtet.

6.4.1 Quality Manager

Der IBM Rational Quality Manager baut auf das Jazz-Projekt auf und ist das aktuelle Werkzeug von IBM zur Testfallverwaltung, aber auch Anforderungserhebung, Testplanung, Testauswertung, Fehlerverwaltung und Durchführung manueller Tests. Es würde damit ClearQuest, ITT-Testmanagement und Manual Tester ersetzen und käme der Ideallösung in vielen Bereichen nahe.

Folgende Anforderungen wären damit abgedeckt: 3.13, 4.9, 4.13, 4.14, sowie die Anforderungen zum ITT-Testmanagement und Manual Tester.

Als problematisch sehen wir die Migration der Daten. Grundsätzlich können alle Standarddaten aus anderen Rational-Produkten importiert werden. Allerdings gibt es momentan etliche selbstprogrammierte Erweiterungen, die nicht ohne Weiteres übernommen werden können. Die tatsächliche Migrierbarkeit lässt sich für uns nur schwer abschätzen, ohne einen Versuch mit echten Daten zu starten. Dazu wäre eine Kopie des Produktivsystems mit Adminrechten, eine freie Lizenz jedes verwendeten Rational-Produkts, sowie Kenntnis über die Serverarchitek-

tur bei FleetBoard notwendig. Wir überlassen nähere Untersuchen daher den Mitarbeitern von FleetBoard.

Ebenfalls FleetBoard überlassen möchten wir die Abwägung zwischen dem Vorteil einer integrierten externen Lösung und einer flexiblen Eigenimplementierung, die die bisherigen Daten weiter verwenden und Stück für Stück die alten Werkzeuge ablösen kann.

6.4.2 Eigenentwicklung

Um den zahlreichen, nicht durch Tools von der Stange abdeckbaren, Anforderungen des Qualitätsicherungsprozesses bei FleetBoard gerecht zu werden und um aufwendige Datenmigrationen zu vermeiden, bietet sich der Einsatz eines selbst entwickelten Werkzeugs an.

Bei einer Eigenentwicklung ist es möglich, sich an der vorgeschlagenen Ideallösung zu orientieren, mit dem Unterschied, dass die bisherige Datenbank weiter verwendet wird und deren Einschränkungen beachtet werden müssen. Die Frage, ob eine Webanwendung oder ein Fat Client entwickelt werden soll, wird leicht beantwortbar, wenn man überlegt, dass die Anwendung eine enge Anbindung an andere lokal laufende Anwendungen benötigt. Das ist mit einer Webanwendung kaum umsetzbar, weswegen aus technischen Gründen ein Fat Client notwendig ist.

Die Einführung einer Eigenentwicklung wird dadurch erleichtert, dass keine Migration und kein Toolwechsel auf einen Schlag nötig sind. Die Funktionen des im folgenden skizzierten Programms lassen sich schrittweise implementieren, wodurch der alte ClearQuest-Client schleichend abgelöst werden kann. Da keine Kompatibilitätsprobleme verursachende Änderungen vorgeschlagen werden, kann in der Übergangszeit und auch später der normale ClearQuest-Client weiter verwendet werden.

Die Unterscheidung der Rollen in der Ideallösung muss auch in der tatsächlichen Umsetzung beachtet werden. Der Einstieg in die Anwendung (Übersichtsseite) sollte sich daran orientieren. Da eine umfassende Lösung mit allen Rollen nicht nur für die Fachstudie, sondern auch für die Umsetzung bei FleetBoard sehr umfangreich ist, haben wir die Rollen priorisiert und drei davon ausgearbeitet.

Wir schlagen folgende Reihenfolge für die Umsetzung der Ansichten für die Rollen vor:

1. Testfallspezifizierer
2. Tester
3. Testautomatisierer
4. Testauswerter
5. Testplaner
6. Spezifizierer

Eine neue Anwendung für den Testfallspezifizierer bringt unserer Meinung nach den größten Nutzen. Der Tester kommt direkt im Anschluss, da wir dringend zur Abschaffung von Manual Tester raten. Testauswerter und Testplaner kommen später, weil sie mit dem ITT-Testmanagement bereits eine gute Werkzeugunterstützung haben. Sollten die Funktionen dieser beiden Rollen aus dem ITT-Testmanagement in das neue Tool migriert werden, schlagen wir eine Orientierung an den Vorschlägen der Ideallösung vor. Die Ansicht des Spezifierers hat die geringste Priorität, weil nicht klar ist, ob die Spezifikation überhaupt aus Requisite Pro ausgelagert werden sollte. Da dies aber eine Möglichkeit ist, wird es hier mit aufgeführt, ohne dass eine genaue Beschreibung dieser Ansicht folgt.

Die Anforderungen an die Ansichten der höchst priorisierten drei Rollen werden im Folgenden skizziert. Eine detaillierte Spezifikation sprengt hier den Rahmen.

Testfallspezifizierer

(Anf. 4.2) Der Testfallspezifizierer bekommt eine Ansicht, die links eine Baumstruktur (ähnlich wie ClearQuest) anzeigt. Diese stellt die Datenstruktur der ClearQuest-Datenbank dar. Beachtet werden muss, dass unnötige Elemente (siehe Anforderung 4.2) nicht angezeigt und Elemente mit gleicher Funktion (Strukturierung) gleich dargestellt werden. So werden Asset-Registries und Testpläne gleich dargestellt. (Asset-Registries sind auch nur eine Gruppierung für Testfälle und damit mit Testplänen, die auch keine weitergehende Funktion haben, gleichwertig.) Unnötige Zwischenelemente aus dem ClearQuest-Client werden nicht übernommen.

(Anf. 4.5, 4.11) Die Ansicht ist grundsätzlich zweigeteilt. Während links stets der Baum angezeigt wird, werden rechts im Falle des Testfalls die aus ClearQuest bekannten Felder für Daten angezeigt. Unnötige Felder werden weggelassen. Darunter werden die Prüfpunkte des ausgewählten Testfalls in einer Tabelle angezeigt. Bei der Tabelle sollte besonders auf Usability wert gelegt werden. Häufige Wechsel zwischen Tastatur und Maus müssen vermieden werden. Copy & Paste und Drag & Drop müssen möglich sein. Neben der Tabelle sind zum schnellen Einfügen über Buttons konfigurierbare Vorlagen verfügbar, um Anforderung 4.11 zu erfüllen. Zwischen den Prüfpunkten kann eine Ordnung angegeben werden. Die Reihenfolge der Prüfpunkte kann nachträglich verändert werden.

(Anf. 3.13) Ein Testfall wird hat initial den Umsetzungsstatus „Offen„. Zusätzlich zum Umsetzungsstatus gibt es einen Automatisierungsgrad, welcher bei der Testfallspezifikation angegeben werden kann. Initial hat er den Wert „Automatisiert (nicht implementiert)“. Für manuelle Tests kann dieser Grad direkt bei der Testfallspezifikation auf „Manuell“ gesetzt werden. Durch die komplette Integrierung der manuellen Tests in dieses Werkzeug ist die Anforderung 3.13 automatisch erfüllt.

Da es vorkommt, dass neue Versionen eines Produkts einen Testfall überflüssig machen, gibt es im Testfall die Möglichkeit die Eigenschaft „Veraltet seit“ mit einer Version zu versehen. Für Versionen nach dieser kann dann keine neue Konfiguration mehr angelegt werden. Veraltete Testfälle werden in der Baumstruktur als solche markiert. Hierfür muss die Datenstruktur der ClearQuest-Datenbank mit Hilfe des Designers angepasst werden.

(Anf. 4.7) Werden für verschiedene Produkte unterschiedliche Konfigurationen benötigt, so gibt es die Möglichkeit, einem Testfall beliebig viele Konfigurationen zuzuordnen (ebenfalls im Baum dargestellt). Eine Konfiguration kann Eigenschaften des Testfalls überschreiben und ist mit einem Produkt und einer Version verknüpft. Der übergeordnete Testfall dient in dem Fall dazu, das allgemeine Vorgehen bei dem Test zu dokumentieren. Die Konfigurationen passen dieses Vorgehen für eine Version eines Produktes an. Solange es nur eine Konfiguration für einen Testfall gibt, wird dieses Element nicht benötigt, sondern es reicht aus nur mit dem Testfall selbst zu arbeiten.

Zur Übersicht kann die Baumstruktur nach Produkt, Version sowie Umsetzungsstatus der Anforderungen und Spezifikationen gefiltert werden. Wird nach Produkt und Version gefiltert, so werden sämtliche Konfigurationen von Testfällen ausgeblendet und in der Ansicht des Testfalls direkt der Inhalt der gefilterten Version angezeigt. Die Tatsache, dass es weitere Konfigurationen zu dem Testfall gibt, wird durch eine kurze Information sichtbar gemacht.

Ein Testfall hat einen Umsetzungsstatus, weil ein manueller Test im Normalfall nach der Spezifikationen noch verfeinert werden muss und weil ein automatisierbarer Testfall evtl. noch automatisiert wird. Erst wenn fest steht, dass der Testfall so getestet werden soll, wird der Status auf „Abgeschlossen“ gesetzt.

Tester

Auch die zweite Ansicht für den Tester zeigt links den Testfallbaum an. Der Testfallbaum kann gefiltert werden. Jetzt sind allerdings Testpläne und Testfälle darin markierbar. Rechts des Bau-

mes kann mit Hilfe eines Buttons die Testdurchführung gestartet werden. Dabei werden manuelle Testfälle direkt zur Bearbeitung angeboten, in dem die Prüfpunkte nacheinander angezeigt werden und mit „Ok“, „Nicht Ok“, „Ok mit Anmerkung“ oder „Nicht ausgeführt“ quittiert werden können. Die Ergebnisse werden direkt in die ClearQuest-Datenbank eingetragen. Automatisierte Testfälle, also solche, die eine Verknüpfung mit Functional Tester haben, können durch Klick auf einen Button automatisch von Functional Tester ausgeführt werden. Zusätzlich kann ein automatischer Modus aktiviert werden, bei dem aufeinanderfolgende Functional Tester Skripte ohne weitere Bestätigung nacheinander ausgeführt werden.

Während der Testdurchführung wird die benötigte Zeit zur Ausführung der Testfälle erfasst und für die Testplanung und Restaufwandsschätzung gespeichert. Die hier beschriebene einfache Durchführung von manuellen Tests ist der aktuellen Lösung mit Manual Tester um einiges voraus. Bei der Ablösung von Manual Tester entsteht allerdings Migrationsaufwand, weil Manual Tester Testbeschreibungen nicht automatisiert in ClearQuest-Prüfpunkte überführt werden können. Dem gegenüber steht der Vorteil, sämtliche Testbeschreibungen in einer Datenbank zu haben und nicht wie bisher auf ClearQuest und Manual Tester verteilt.

Testautomatisierer

Der Testautomatisierer bekommt im Wesentlichen die gleiche Ansicht wie der Testfallspezifizierer. Im Gegensatz zu diesem sieht der Testautomatisierer generierte Prüfpunktkommentare und Methodenrümpfe, wie es aus dem bisherigen ClearQuest-Client bekannt ist. Da der Testautomatisierer die Prüfpunkte nicht verändern muss, ist seine Ansicht im Nur-Lesen-Modus. Die Ansicht des Testautomatisierers unterstützt eine kompakte Darstellung der Prüfpunkte des ausgewählten Testfalls. Diese Ansicht kann in ein extra Fenster verlegt werden, das auf „immer im Vordergrund“ eingestellt werden kann. So kann der Testautomatisierer während seiner Arbeit im Functional Tester stets ohne Platz- und Zeitverschwendung seine Aufgabe sehen.

A

Begriffslexikon

Begriff **DispoPilot**
Bedeutung Der DispoPilot ist ein Handgerät mit Barcodescanner und Navigationsfunktion, das im LKW für die Disposition eingesetzt wird.

Begriff **Konfigurierter Testfall**
Bedeutung Ein Konfigurierter Testfall ist ein Kindelement des Testfalls in ClearQuest. Er dient zur Anpassung des Testfalls an spezifische Hardware- oder Softwarekonfigurationen, die für die Ausführung des Testfalls verwendet werden. Functional Tester Skripte werden mit Konfigurierten Testfällen verknüpft und Testlogs werden an sie angehängt.

Begriff **Prüfpunkt**
Bedeutung Ein Prüfpunkt ist ein Kindelement des Testfalls in ClearQuest. Ein Testfall enthält 0-n Prüfpunkte. Der Prüfpunkt ist ein selbst entwickeltes Element und enthält Felder für Vorbedingung, Aktion, Nachbedingung und Automatisierungsstatus. Damit entspricht er dem „Lehrbuch-Testfall“.

Begriff **Schwarzes Team**
Bedeutung Das schwarze Team ist ein Scrum-Team, das die Integrationstests leitet und koordiniert. Zu diesen Aufgaben zählen nicht nur die Durchführung im Rahmen von Regressionstests, sondern auch die Entwicklung der Testinfrastruktur.
Synonym Black Team, Integrationstestteam

Begriff **Scrum Team**
Bedeutung Alle Entwicklungsteams, die nicht das schwarze Team sind, werden bei FleetBoard einfach „Scrum-Team“ genannt.
Synonym Entwicklungs-Scrum-Team, Entwicklungsteam

Begriff **Telematikplattform**
Bedeutung Die Telematikplattform ist ein Gerät, das in die LKW der Kunden eingebaut wird und dort Fahrzeugdaten vom CAN-Bus zusammenträgt und diese über GPRS an FleetBoard-Server schickt.
Abkürzung TP

Begriff **Testfall**
Bedeutung Ein Testfall bei FleetBoard ist ein Container, der Prüfpunkte enthält. Er entspricht nicht dem „Lehrbuch-Testfall“. Seine Verwendung bei FleetBoard unterscheidet sich kaum von der des Testplans.
Querverweis Prüfpunkt

B Fragenkatalog

Es folgt der Fragebogen, den wir für die Interviews bei FleetBoard verwendet haben.

B.1 Einleitung

- Ziel der Fachstudie, Zweck des Interviews.
- Was ist deine Aufgabe bei Fleetboard?
- Was ist deine Aufgabe in Bezug auf Tests? (Spezifizierst du Testfälle?, Implementierst du Testfälle?, Verwaltest du Testfälle?, Führst du Tests durch?)
- Mit welcher Art von Testfällen hast du zu tun? (GUI, Betrieb, Oberfläche, Last, „Kundenakzeptanz“, ...)
- Welche Größenordnung von Testfällen verwaltest du?

B.2 Werkzeuge

- Welche Werkzeuge setzt du ein?

Für jedes Werkzeug:

- Allgemeinbewertung: Was hältst du von diesem Werkzeug auf einer Skala von 1-10 (1 ungenügend, 10 sehr gut)
- Welche Probleme gibt es mit diesem Werkzeug? Was könnte man an diesem Werkzeug verbessern?
- Was findest du gut an diesem Werkzeug?
- Kennst du ein anderes Werkzeug, welches die Aufgabe besser lösen kann? Was konkret ist dort besser?

B.3 Testprozess

- Beschreibe den Testprozess aus deiner Sicht
- Was hältst du vom Testprozess auf einer Skala von 1-10?
- Brauchst du Informationen oder Daten von jemand anderem? Von wem? Was?
- Woher weißt du, was du zu testen hast?
- Wer bekommt die Ergebnisse deiner Arbeit?
- Wie funktioniert die Testaufwandsplanung? Was kann man daran verbessern?
- Enthält der Testprozess unnötige Schritte?
- Fehlen im Testprozess deiner Meinung nach erforderliche Schritte?
- Was gefällt dir am Testprozess?
- Was stört dich am Testprozess?
- Wer legt fest / Wo steht was überhaupt getestet wird? (alle Features?, eine Teilmenge? welche?)

B.4 Testfälle

B.4.1 Spezifikation

- Unterscheiden sich deine Testfälle in irgendeiner Form von „anderen“ Testfällen bei FleetBoard? Haben sie spezielle Merkmale? Benötigst du eventuell andere Eingabefelder, kannst du mit vorhandenen Feldern nichts anfangen?
- Welche Daten benötigst du dabei und wo nimmst du diese her?
- Wie beurteilst du die Qualität dieser Daten auf einer Skala von 1-10? (Sind die Testfälle verständlich?, eindeutig formuliert?)
- Wie sind die Testfälle strukturiert? Wie ist die Gliederung in ClearQuest? Gibt es dafür eine Vorgabe?
- Wie sind die einzelnen Felder zu benutzen? Werden sie auch so benutzt? Wenn ja, warum nicht? Machen „die anderen“ das auch richtig?
- Gibt es eine Priorisierung von Testfällen?
- Wird der benötige Aufwand für die Durchführung abgeschätzt? Wie?
- Wie kann man die Aufwandsabschätzung verbessern?
- Problematik des Testfalls als Gruppierung für Prüfpunkte ansprechen (Zuordnung von Prüfpunkten zu Testfällen möglicherweise nicht sinnvoll). Wie siehst du das? Entstehen daraus deiner Meinung nach Probleme?

B.4.2 Implementierung

- Welche Daten benötigst du dabei und wo nimmst du diese her?
- Wie beurteilst du die Qualität dieser Daten auf einer Skala von 1-10? (Sind die Testfälle verständlich?, eindeutig formuliert?)
- Wie findest du die Verbindung zwischen ClearQuest und Functional Tester (Werkzeug A und B falls andere Tools bei z.B. den Hardware-Leuten).
- Was stört? Was lässt sich besser machen?
- Was ist gut?
- Was lässt sich eventuell noch automatisieren? Welchen Aufwand würde das bedeuten? (Wie viel Zeit spart man durch die Automatisierung? Wie lange dauert die Automatisierung? (Wartung der Automatisierung?))

B.4.3 Durchführung

- Wie funktioniert die Testdurchführung auf deinem Arbeitsgebiet? Welche Schritte sind dafür notwendig?
- Welche Daten benötigst du dabei und wo nimmst du diese her?
- Wie beurteilst du die Qualität dieser Daten auf einer Skala von 1-10? (Sind die Testfälle verständlich?, eindeutig formuliert?)
- Wo gehen die Ergebnisse hin? (Personen)
- Welche Schritte werden dabei manuell erledigt und könnten unabhängig vom verwendeten Werkzeug automatisiert werden? (Theoretisch automatisierbar) (Wie viel Zeit spart man durch die Automatisierung? Wie lange dauert die Automatisierung? (Wartung der Automatisierung?))
- In welcher Form liegen die Testergebnisse vor und wie werden diese weiterverarbeitet? Falls komplett manuell: Gibt es eine Form-Vorgabe? (Formular)
- Hältst du diese Form für sinnvoll? (Was kann man besser machen?)
- Falls manuelle Ergebnisse: Wie kommen diese in die ClearQuest-Datenbank?
- Was passiert, wenn ein Fehler gefunden wird, der weiteres Testen unmöglich macht, bzw. wenn wesentliche Features nicht funktionieren?
- Was ist das Testendekriterium?

B.4.4 Auswertung

- In welcher Form werden dir die Testergebnisse präsentiert? Welches Tool? Wo kommen die Daten her?
- Was ist daran schlecht?
- Was könnte verbessert werden?
- Was ist gut?

C Priorisierung der Anforderungen

In dem Lösungsvorschlag konzentrieren wir uns nur auf die wichtigsten Anforderungen. Der Rest bleibt dort teilweise unbeachtet. Der Auswahl liegt eine subjektive Priorisierung unserseits zugrunde, die wir der Vollständigkeit halber hier noch mit einfügen.

Bei der Priorisierung handelt es sich um eine Tabelle, welche alle Schwachstellen und dazugehörige Anforderungen auflistet und jeweils einen geschätzten Wert für den Nutzen (N) und für die Kosten (K) bekommen hat. Da wir die Organisatorische Ebene grundsätzlich für wichtiger halten als des Rest, bekommt jeder Nutzen-Wert zwei hinzu addiert. Der Prozess bekommt plus eins. Dies ist eine Gewichtung (GN) nach Kategorie. Auf Grund dieser Schätzungen gibt es eine Empfehlung (E) zur Umsetzung der Anforderung.

Zu den Skalen: Der Nutzen wurde auf einer Skala von 1 bis 5 geschätzt, wobei 5 der höchste Nutzen ist. Durch die Gewichtung hat der gewichtete Nutzen eine Skala von 1 bis 7. Einen Nutzen von 5 und höher haben wir grün (sehr hoch) markiert. 4 ist gelb (hoch) markiert und 3 rot (mittel). 2 und 1 sind nicht markiert (niedrig). Die Kosten haben ebenfalls eine Skala von 1 bis 5, wobei 5 sehr teuer ist. Wir haben zur Übersicht auch hier Farben verwendet: Grün für 1 (sehr billig), Gelb für 2 (billig), Rot für 3 (mittel).

Zu der Empfehlung: Es handelt sich dabei um eine sehr subjektive Einschätzung. Grundsätzlich wurden Nutzen und Kosten abgewägt. Es fließen aber auch Wünsche der Mitarbeiter mit ein. Vergeben wurde stark (Grün), mittel (Gelb) und niedrig (Rot) als empfohlene Priorisierung. Es ist natürlich trotzdem sinnvoll nicht priorisierte Anforderungen umzusetzen. Wir geben lediglich eine Hilfestellung zur Selektion, falls aus Zeitmangel nicht alle umgesetzt werden können.

Gewichtung der Anforderungen und unsere Empfehlung

Anf.	Schwachstelle	N	GN	K	E
2.1	Probleme auf Organisatorischer Ebene	5	7	3	Stark
2.2	Zu wenig Tester	5	7	5	Mittel
3.1	Fehlender Überblick über den Testprozess	3	4	1	Stark
3.2	Entwickler testen ihren eigenen Code	4	5	3	Mittel
3.3	Kommunikationsprobleme bei verschiedenen Versionen der eigenen Software	1	2	1	Niedrig
3.4	Zu wenig Zeit zur Automatisierung von Tests	5	6	2	Stark
3.5	Problematische Übergänge zwischen den Phasen des Testprozesses	3	4	3	
3.6	Vermeidung von Redundanz	3	4	1	
3.7	Terminologie	4	5	2	Stark
3.8	Verbesserung der Testinfrastruktur	2	3	3	
3.9	Unzureichende Qualität der Dokumentation	4	5	3	Mittel
3.10	Nicht einheitliche Verwaltung der verschiedenen Dokumentationen	3	4	3	
3.11	Einführung der Kundenakzeptanztests	4	5	4	Stark
3.12	Mangelhafte Verknüpfung von Testergebnissen mit den Tests	5	6	2	Stark
3.13	Manuelle Tests sind zu vielschichtig	4	5	4	
4.1	Unzureichende Dokumentation zu ClearQuest	2	2	2	
4.2	Unnötige Elemente im Testfallbaum	3	3	5	
4.3	Unkontrollierte Struktur des Testfallbaums	5	5	3	Mittel
4.4	ClearQuest-Testfall ist kein Testfall	2	2	4	
4.5	Aktuelle Methode zur Testfallspeicherung erlaubt keine Ordnung	4	4	2	Stark
4.6	Soll-Umfang und Soll-Inhalt eines Testfalls ist nicht definiert	1	1	1	Niedrig
4.7	Uneinheitliche Verwendung von Konfigurierten Testfällen	2	2	3	
4.8	Schwer bedienbare Suchfunktion	3	3	5	
4.9	Klick- und fensteranzahlintensive Bedienung	5	5	5	Niedrig
4.10	Schlechte Unterstützung der Strukturierung von Testfällen und Strukturierungselementen	4	4	5	
4.11	Fehlende Templates für Texte	5	5	4	Stark
4.12	Tests können nicht aus dem Testfallverwaltungswerkzeug gestartet werden	4	4	2	Mittel
4.13	Generierung der Testfallzusammenfassung nicht vollautomatisch	3	3	2	
4.14	Überflüssige Formularfelder	3	3	1	Mittel
4.15	Fehlende einheitliche und gute Dokumentation	3	3	1	Mittel
4.16	Umständliche Bedienung der Oberfläche	3	3	2	
4.17	Konsistenz bei den Seitennamen	1	1	1	Mittel
4.18	Feature-Orientierung	4	4	2	Mittel
4.19	Skalierbarkeit der Darstellung	2	2	2	
4.20	Mängel in der Dokumentation	4	4	2	Mittel
4.21	Fehlende Kontrolle der nach Anforderung 20. eingeführten Richtlinien	5	5	1	Stark
4.22	Mergeprobleme	4	4	3	
4.23	Riesige globale Properties-Datei	2	2	2	
4.24	Fehlende Dokumentation	3	3	1	Mittel
4.25	Ineffizient und langsam	4	4	5	Stark
4.26	Schlechte Usability	1	1	mit 4.25	Stark
4.27	Unzufriedenheit der Mitarbeiter	5	5	mit 4.25	Stark
4.28	Fehlende Lizenzen	4	4	2	Mittel

D Beispiele

Es folgen einige Beispieldaten für die Strukturierung der Testfälle und für Prüfpunkte, um dem Leser ein Gefühl für die Daten bei FleetBoard zu geben. Die Daten wurden ausgewählt, um verschiedene Arten und Ausprägungen von Testfällen und deren Strukturierung im Ist-Zustand zu zeigen. Die gezeigten Prüfpunkte stammen aus verschiedenen Testfällen und wurden ausgewählt, um die Verschiedenheit der Prüfpunkte zu demonstrieren. Sie entsprechen dem Originalzustand der Darstellung in der Ansicht „Testfallspezifikation“ in ClearQuest. Diese Ansicht bildet auch für die Entwickler die Grundlage zum Implementieren von Testfällen. Inhalt und Formatierung wurden daher nicht verändert. Kürzungen sind durch das Wort „gekürzt“ in eckigen Klammern gekennzeichnet.

D.1 Betrieb

```
ID: 2140        Kurzbeschr.: Batchjob - beenden
        Automatisierung: N

[ Vorbedingung ]
Batchjob muss gestartet sein.

[ Benutzeraktion ]
Die Batchjobs beenden sich nach abarbeitung der aktuellen Flotte.

[ Soll-Ergebnis ]
Batchjobs wurden korrekt beendet.
=====================================

ID: 2141        Kurzbeschr.: Batchjobs - neu anlegen
        Automatisierung: N

[ Vorbedingung ]
Batchjobs müssen beendet sein.

[ Benutzeraktion ]
Die Batchjobs werden auf einem andern WAS-Member neu angelegt.

[ Soll-Ergebnis ]
Batchjobs wurden auf einem anderen Member neu angelegt.
=====================================

ID: 2142        Kurzbeschr.: Batchjobs - starten
        Automatisierung: N

[ Vorbedingung ]
Batchjobs müssen auf einem WAS-Member angelegt sein.

[ Benutzeraktion ]
Die Batchjobs starten automatisch an den angegebenen Zeitpunkten.

[ Soll-Ergebnis ]
Alle Batchjobs wurden gestartet.
=====================================

ID: 2143        Kurzbeschr.: Prüfen der Logfiles
        Automatisierung: N

[ Vorbedingung ]
Logfiles müssen vorhanden sein.
```

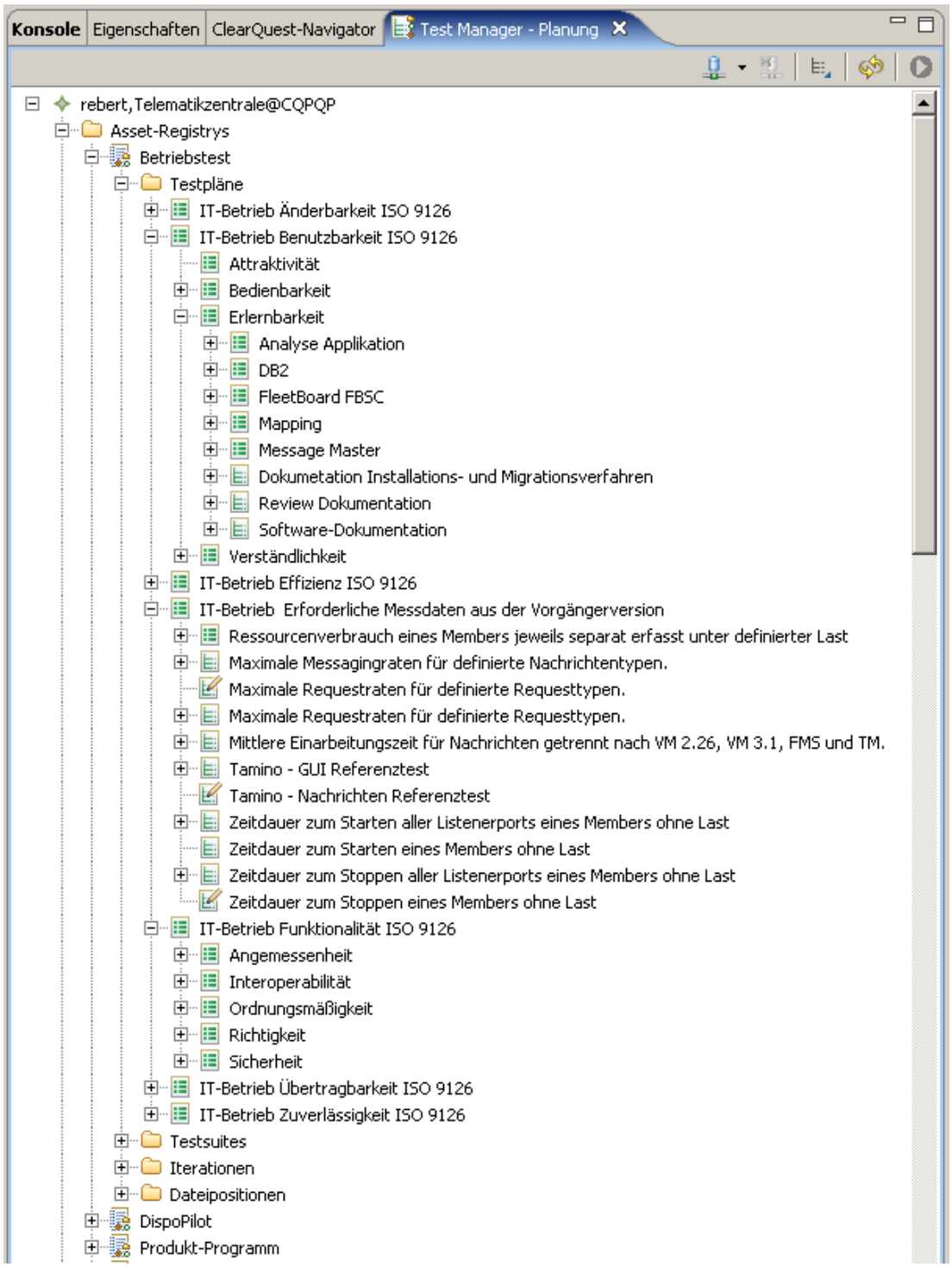


Abbildung D.1: Testfallstrukturierung der Betriebstests

```
[ Benutzeraktion ]
Prüfung ob die zum Abbruchzeitpunkt des Batchjobs aktive Flotte
    ordnungsgemäß abgearbeitet wurde.

[ Soll-Ergebnis ]
Keine Fehler in der Logfile.
```

D.2 Telematikplattform-Test

```
ID: 6754        Kurzbeschr.: query_all
        Automatisierung: N

[ Vorbedingung ]
Setup_Testcase_IES executed.

[ Benutzeraktion ]
1)      Send the following dynamic telediagnose frame query to TP

        Reference DS#19 (dynamic telediagnose frame query):

        Source # 19                                                 0x13
        Length                                                      0x008B

        Number of Queries PreDoOnes     0x02
        ECU address                                                 0x01
        Query Protocol Version(IES)             0x00
        Query length (1.1)                              0x14
        Query frame (1.1)                               0x06 00 11 04 98
             22 20 20 98 22 20 21 98 22 20 22 98 22 20 23
        ECU address                                                 0x02
        Query Protocol Version(IES)             0x00
        Query length (1.2)                              0x14
        Query frame (1.2)                               0x06 00 11 04 98
             22 20 20 98 22 20 21 98 22 20 22 98 22 20 23

        .... [ gekürzt ] ....


        2)      Simulate the positive response on CAN to the dynamic
            telediagnose frame from 1.)

        Reference positive response for every query to a ECU using IES:
        06002408986220203937323008986220213231314
            B089862202237323930089862202335 37FFFF

[ Soll-Ergebnis ]
1) TP sends CAN messages, corresponding to the query protocol, to the
    ECUs in the folowing order:
                ECU1(query 1.1)
                ECU2(query 1.2)
                ECU1(query 2.1+2.2)
                ECU2(query 2.1+2.2)
                ECU1(query 3.1)
                ECU2(query 3.2)

        2)TP replies to DS#19 containing positive and negative response
            for the query

        Source # 19                                                 0x13
        Length                                                      0x01F9

        Number of Queries PreDoOnes     0x02
        ECU address                                                 0x01
        Query Protocol Version(IES)             0x00
        Query length (1.1)                              0x14
        Query frame (1.1)                               0x06 00 11 04 98
             22 20 20 98 22 20 21 98 22 20 22 98 22 20 23
        Response length (1.1)                   0x27
        Response frame (1.1)                    0
            x06002408986220203937323008986220213231314B0898622022373
            23930089862202335 37FFFF
        ECU address                                                 0x02
        Query Protocol Version(IES)             0x00
        Query length (1.2)                              0x14
        Query frame (1.2)                               0x06 00 11 04 98
             22 20 20 98 22 20 21 98 22 20 22 98 22 20 23
        Response length (1.2)                   0x27
        Response frame (1.2)                    0
            x06002408986220203937323008986220213231314B0898622022373
            23930089862202335 37FFFF
```

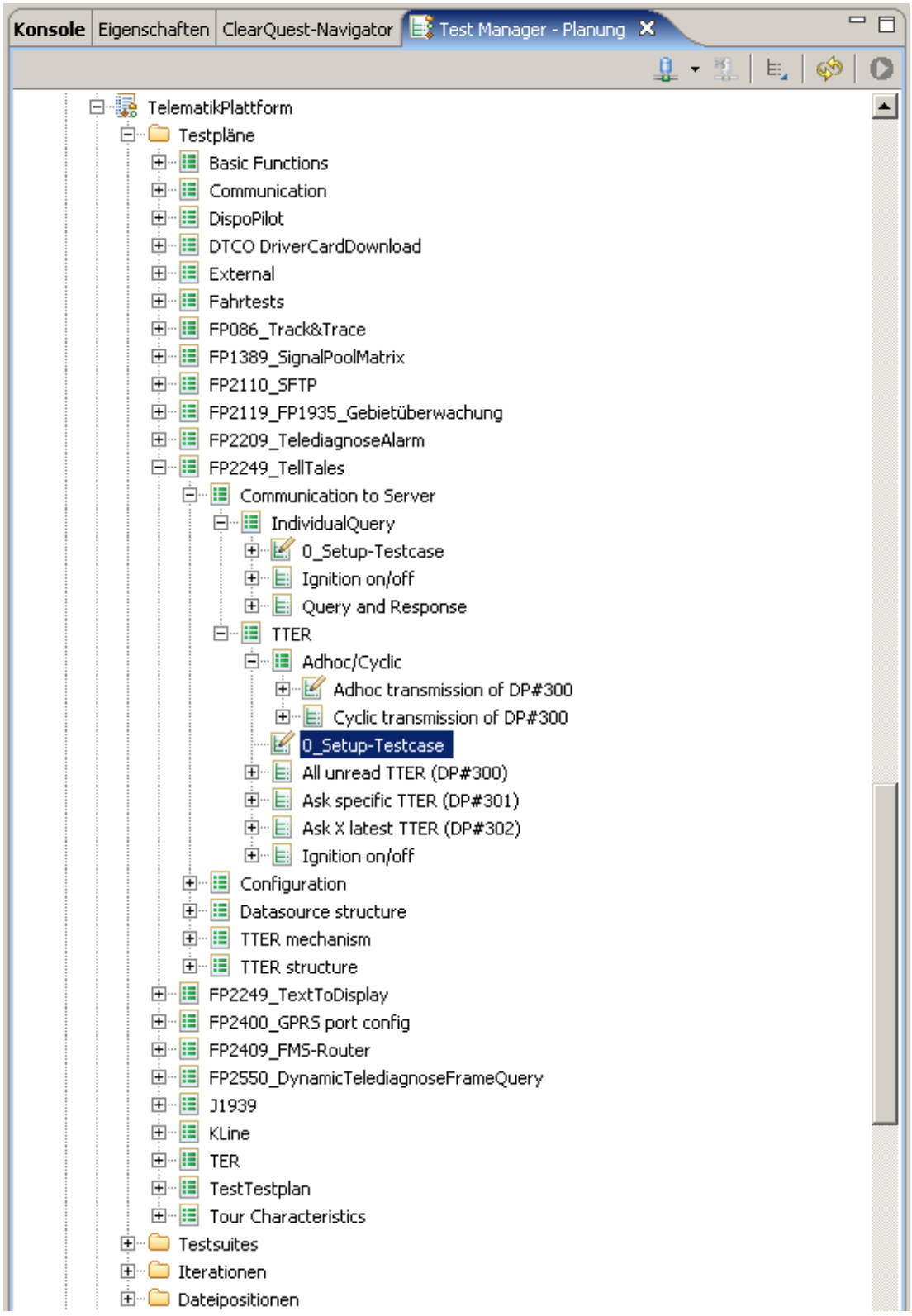


Abbildung D.2: Testfallstrukturierung der TP-Tests

```
        Number of Queries Do4All                0x04
        ECU address                                                 0x01
        Query Protocol Version(IES)             0x00
        Query length (2.1)                                  0x14
        Query frame (2.1)                                   0x06 00 11 04 98
             22 20 20 98 22 20 21 98 22 20 22 98 22 20 23
        Response length (2.1)                   0x27
        Response frame (2.1)                    0
            x06002408986220203937323008986220213231314B0898622022373
            2393008986220233537FFFF
        ECU address                                                 0x01

        .... [ gekürzt ] ....
=====================================

ID: 5035        Kurzbeschr.: Event "Yellow"
        Automatisierung: N

[ Vorbedingung ]
Setup testcase executed

[ Benutzeraktion ]
1. Set status of tell-tale ID 1 to "Info". (ref_telltaleStat01=03)
2. Wait for longer than the time given in TP-config Parameter 247 coded
    in third byte. (see Setup-TestCase)
3. Set status of tell-tale ID 1 to "Yellow". (ref_telltaleStat01=02)  -
    (time t1)
4. Wait till the threshold given in TP-config Parameter 247 coded in
    third byte is overcome. (see Setup-TestCase)
5. Verify recieved DS#100.

[ Soll-Ergebnis ]
5. DP#300 is sent adhoc from TP to FBSC, containing DS#100 with TTER3 "
    Yellow" which tell-tale ID is 1 an the timestamp is equal to t1. (
    Perhaps TTER3 "Info" is included)
=====================================

ID: 6349        Kurzbeschr.: Configuration
        Automatisierung: N

[ Vorbedingung ]
SFTP dashboard prepared.

TP installed and activated.

[ Benutzeraktion ]


[ Soll-Ergebnis ]
```

D.3 DispoPilot-Test

```
 ID: 3073       Kurzbeschr.: Einstellungen-Navigation-Kartenausrichtung
        Automatisierung: J

[ Vorbedingung ]


[ Benutzeraktion ]
Es wird gebprueft ob folgende Buttons  bzw. Listenelemente angezeigt
    werden:
Nordausrichtung
Fahrtausrichtung
Back
Ok

[ Soll-Ergebnis ]
Folgende Buttons  bzw. Listenelemente werden angezeigt:
Nordausrichtung
Fahrtausrichtung
Back
Ok
=====================================

ID: 3074        Kurzbeschr.: Einstellunge-Navigation-Sonderziel Radius
        Automatisierung: J

[ Vorbedingung ]
```

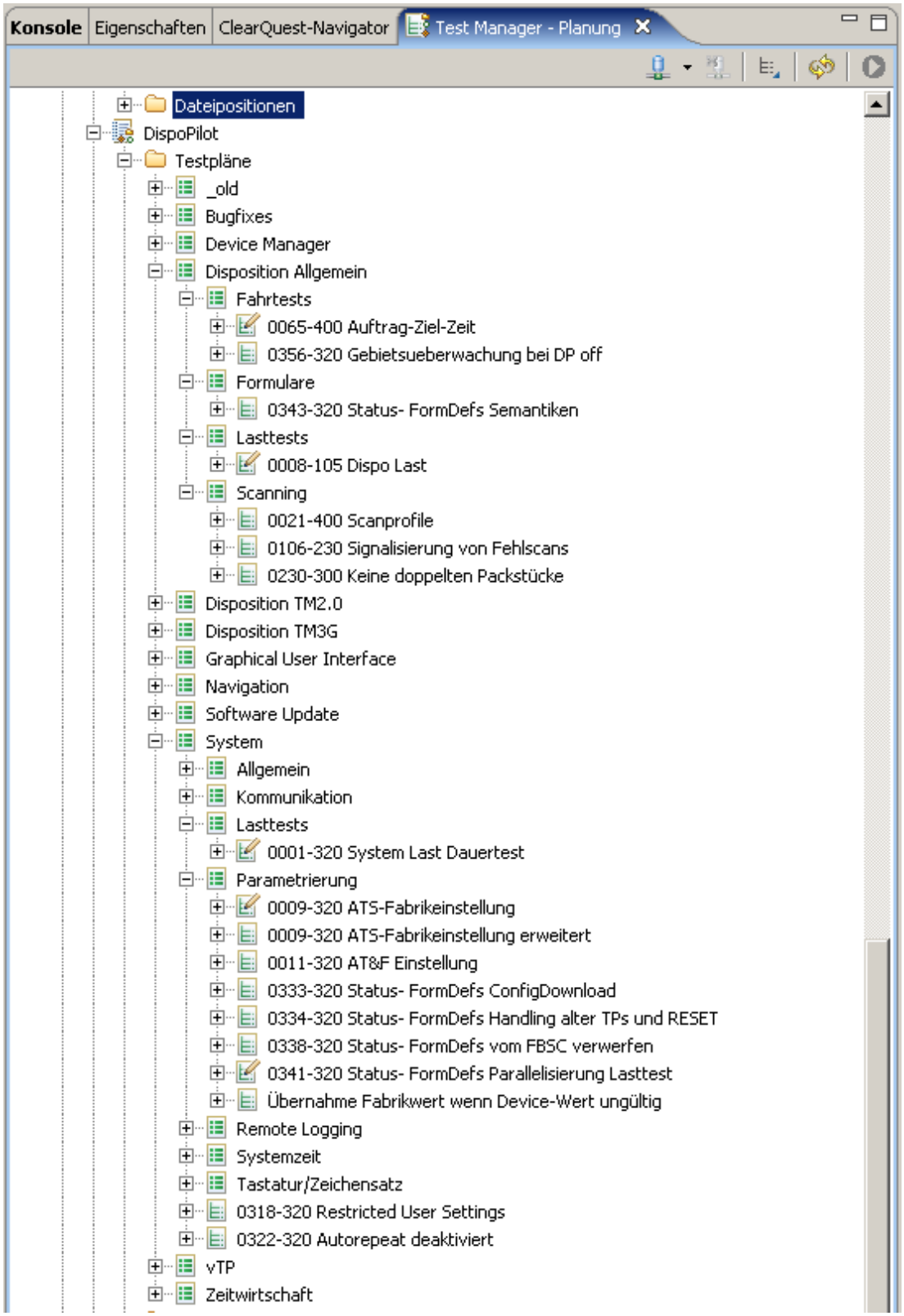


Abbildung D.3: Testfallstrukturierung der DispoPilot-Tests

```
[ Benutzeraktion ]
Es wird gebprueft ob folgende Buttons bzw. Listenelemente angezeigt
    werden:
3 km
5 km
10 km
15 km
25 km
Back
OK

[ Soll-Ergebnis ]
Folgende Buttons und Listenelemente werden angezeigt
3 km
5 km
10 km
15 km
25 km
Back
OK
=====================================

ID: 3075        Kurzbeschr.: PLZ Meldung 'Mind. 2 Zeichen'
        Automatisierung: J

[ Vorbedingung ]


[ Benutzeraktion ]
Eine Ziffer eingeben (default=1) in PLZ Eingabefeld

[ Soll-Ergebnis ]
Suche wird abgelehnt mit der Meldung 'Bitte geben Sie mindestens 2
    Zeichen ein!' ab 2.3.0 Navi2G
```

D.4 Server- und Soap-Schnittstellen-Test

```
 ID: 7709       Kurzbeschr.: Parameterkombinationen
        Automatisierung: J

[ Vorbedingung ]
Nutzer ist angemeldet (hat Session)
Nutzer hat Recht, SOAP-Methode getTm3GMessageIn auszuführen

[ Benutzeraktion ]
1) SOAP Aufruf von getMessageIn mit gültiger Tm3gReferenceNo und
    VehicleID
2) SOAP Aufruf von getMessageIn mit gültiger Tm3gReferenceNo und
    VehicleID = null
3) SOAP Aufruf von getMessageIn mit gültiger VehicleID und
    Tm3gReferenceNo = null

[ Soll-Ergebnis ]
1) kein SOAP Fault - Im Response-Dokument sind alle Messages mit der
    abgefragten Tm3GReferenceNo enthalten
2) kein SOAP Fault - leerer Response
3) kein SOAP Fault - leerer Response
=====================================

ID: 2863        Kurzbeschr.: Phase 1 (Fahrzeug in MF)
        Automatisierung: M

[ Vorbedingung ]
Ein Fahrzeug muß in der Masterflotte MF vorhanden sein.

[ Benutzeraktion ]
Mapping in MF öffnen und eine Positionsabfrage ausführen

[ Soll-Ergebnis ]
Positionsabfrage erfolgreich ausgeführt
=====================================

ID: 2864        Kurzbeschr.: Phase 2 (an PF1 verliehen)
        Automatisierung: M

[ Vorbedingung ]
Masterflotte (MF) zu Partnerflotte (PF1) Beziehung erstellen und ein
    Fahrzeug an PF1 verleihen.

[ Benutzeraktion ]
```

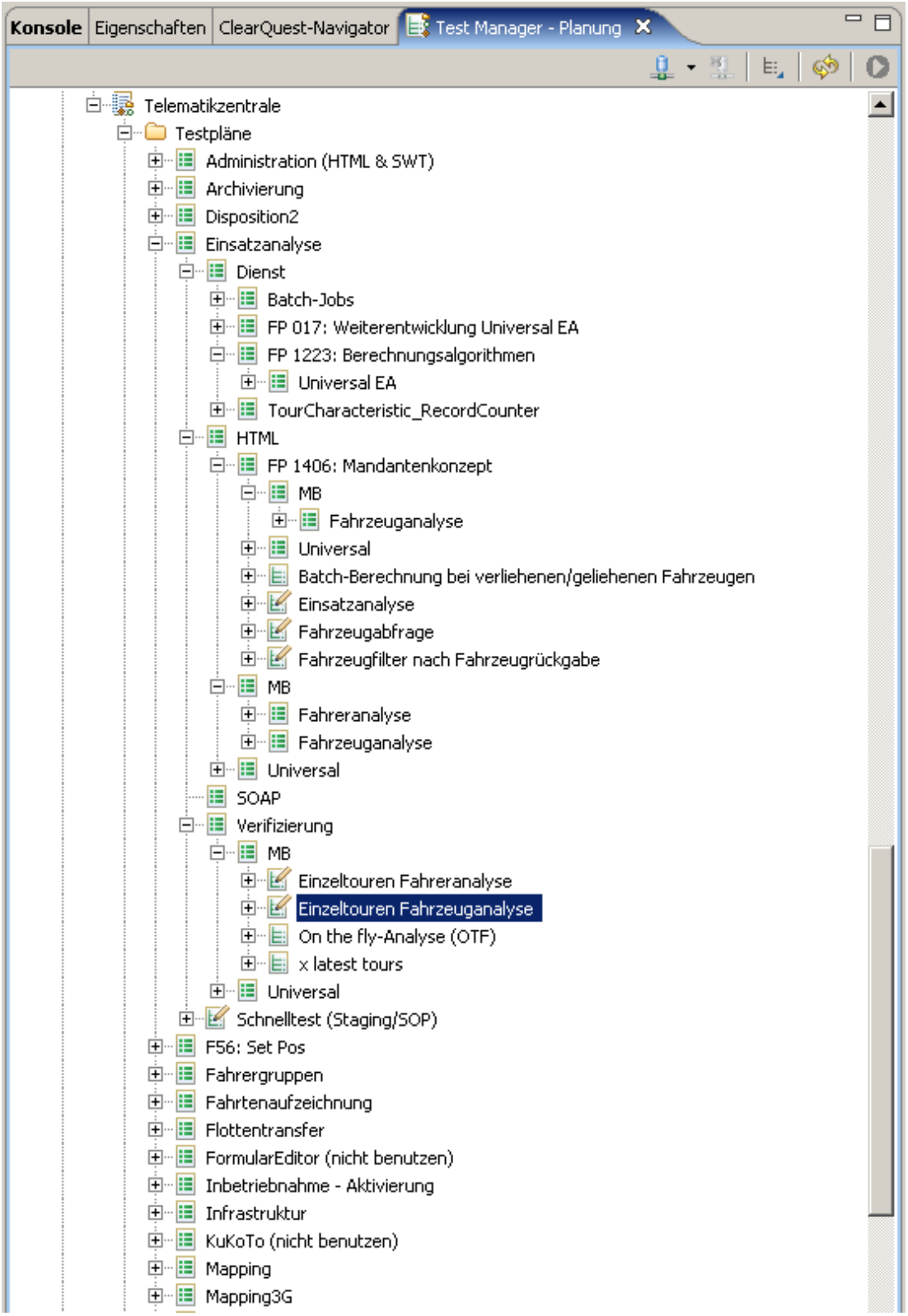


Abbildung D.4: Testfallstrukturierung der Server- und GUI-Tests

Mapping in MF und PF1 öffnen und jeweils eine Positionsabfrage ausführen

[Soll-Ergebnis]
Positionsabfrage jeweils erfolgreich ausgeführt

==========

ID: 3643 Kurzbeschr.: Einzeltouren (Fahrzeug-/Fahreranalyse)
Automatisierung: N

[Vorbedingung]

[Benutzeraktion]
Verifizierung Zeitstempel Tourbeginn / Tourende in GUI mit Tourbeginn/ Tourende-Zeitstempel des Fahrprotokolls (SB-Wechsel bei MB-Einzeltouren und DriverCard-Wechsel bei Universal-Einzeltouren)

[Soll-Ergebnis]

==========

Literaturverzeichnis

[LL07] LUDEWIG, Jochen ; LICHTER, Horst: *Software Engineering*. 1. Auflage. Heidelberg : dpunkt.verlag, 2007