Definition und Automatisierung von Equipment Test Cases für ein IMA System

PROJEKTARBEIT

des Studienganges Mobile Informatik

an der Dualen Hochschule Baden-Württemberg Ravensburg

von

Dominik Bernhardt

##.##.####

Bearbeitungszeitraum	12 Wochen
Matrikelnummer, Kurs	7434619, TIM 17
Ausbildungsfirma	Konzept Informationssysteme GmbH, Meersburg
Betreuer der Ausbildungsfirma	Dipl.-Ing. Jochen Zwick
Gutachter der Dualen Hochschule	Anne Hoffmann

Konzept
Informationssysteme GmbH

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Erklärung

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2015.

Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema:

Definition und Automatisierung von Equipment Test Cases für ein IMA System

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Meersburg, den 28. September 2018

Dominik Bernhardt

Unterschrift

Abstract

Die vorliegende Projektarbeit befasst sich mit dem Definieren und Implementieren von automatisierten Equipment Test Cases für ein Integrated Modular Avionics (IMA) System. Diese Test Cases werden in der Programmiersprache Python realisiert. Anhand der formulierten Requirements im Equipment Requirements Documents (ERD), sollen zunächst die Test Cases definiert werden und in einer Equipment Verification Specification (EVS) dokumentiert werden. Entsprechend der formulierten Spezifikation werden anschließend die Test Skripte implementiert und ausgeführt. Neben der Implementierung der Test Skripte muss zugleich die vorhandene Testanlage an die entsprechenden Test Cases angepasst werden. Die Versionsverwaltung des Softwareprojektes wird über GitLab gesteuert.

The present project is dealing with the definition and implementation of automatic equipment test cases for an Integrated Modular Avionics (IMA) System. These test cases are being generated in the Python programming language. The requirements, which are located in an Equipment Requirements Document (ERD) are supposed to be defined and documented in an Equipment Verification Specification (EVS) and implemented in Python. Simultaneously the testing facility has to be configured and managed beside the implementation of the actual test cases. The project repository is being managed with the web-application GitLab.

A

C

D

E

F

H

I

L

N

Q

R

S

U

Konzept
Informationssysteme GmbH

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Abbildungsverzeichnis

Tabellenverzeichnis

Konzept
Informationssysteme GmbH

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Inhaltsverzeichnis

1 Einleitung

1.1 Firma

Abbildung 1 Konzept Informationssysteme, Meersburg

„Ihr starker Partner, heute und morgen“

Seit Anfang Oktober des Jahres 2017 stehe ich bei der Firma Konzept Informationssysteme GmbH als dualer Student, des Studiengangs Mobile Informatik, unter Vertrag.

In den Bereichen Softwareentwicklung, Systems Engineering und Qualitätssicherung hat sich die Firma Konzept Informationssysteme GmbH erfolgreich im süddeutschen Raum, beziehungsweise seit 2010 auch in der Schweiz etabliert.

An mehreren Standorten in Meersburg, als eingetragener Hauptsitz, Ulm, München und Hünenberg in der Schweiz beschäftigt die Firma rund 150 Mitarbeiter.
Die Firma bearbeitet Projekte aus den Bereichen Avionik, Automotive, Raumfahrt, Energiesysteme, Produktion und Logistik, Medizintechnik sowie Verteidigungstechnik.
Für meine erste Projektarbeit werde ich, als Teil eines Teams, zusammen mit vier Kollegen einen Projektauftrag einer Firma im Avionik-Bereich bearbeiten.

1.2 Projektbeschreibung

Für die Firma sollen automatisierte Tests, auf Python Basis, geschrieben und ausgeführt werden in welchen ein neues IMA System auf die Einhaltung von definierten Anforderungen (engl. „Requirements“) geprüft wird.

IMA bezeichnet einen distributiven Echtzeitcomputer an Bord eines Luftfahrzeuges, mit standardisierten Komponenten und Schnittstellen zur Kommunikation zwischen den unterschiedlichen Systemen.

Es wurde im Vorfeld an das Projekt ein Equipment Requirements Document (ERD) formuliert, in welchem alle System (Black Box) Requirements enthalten sind. In dem Dokument wird ebenfalls zwischen den verschiedenen Typen von Requirements unterschieden. Die Requirements sind aufgeteilt in Analysis, Performance und Functional. Eine Art der Verifikation der jeweiligen Requirements ist die Formulierung von Test Cases, in welchen die Verhaltensweise der UUT nach Requirements verifiziert wird. Mit Hilfe von Test Cases lassen sich Functional und Performance Requirements prüfen.

Zunächst sollen über einen Zeitraum von drei Monaten Requirements der beiden CAN und A429 Interface Implementationen überprüft werden. In den Requirements sind Übertragungen von Daten Typen, sowie Zustände und Signale beschrieben, in denen sich die UUT befinden, beziehungsweise senden soll, wenn bestimmte Fälle eintreten. Anhand dieses Dokumentes werden vom Team passende Test Procedures und Test Cases formuliert und ausgeführt, in denen geprüft wird ob die vorher gegebenen Requirements erfüllt werden.

Das Projekt wurde rechtzeitig zu meinem Einstieg in die Praxisphase Anfang Juli gestartet. Für das neue Produkt gibt es zur Zeit der Studienarbeit noch keinen Hard-

und Software Prototypen, weswegen unser Team die Requirements mit Hilfe eines älteren RDC Systems überprüft. Die beiden Systeme besitzen einen ähnlichen Interfaceaufbau, weswegen beim späteren Testen des eigentlichen Produktes nur minimale Anpassungen durchgeführt werden müssen.

Ein RDC an Bord eines Luftfahrzeuges besteht aus einem Input/Output Interface um Verbindungen zu einem oder mehr Input/Output Geräten und einem Netzwerk-Interface um eine Verbindung zu einem Fernprozessor zu ermöglichen.

Das neue Produkt soll im Gegensatz zum älteren RDC System, welches proprietär entwickelt wurde und nur in bestimmten Flugzeugen zum Einsatz kommt, eine breite Masse an potentiellen Kunden zur Verfügung stehen.

Abbildung 2 Beispiel eines RDC[1]

[1] Quelle: https://www.diehl.com/aviation/en/diehl-aviation/portfolio/portfolio-avionics

1.3 Verwendete Technologien

IMA-System

Integrierte Modulare Avionik bezeichnet eine modulare Rechner-Elektronikeinheit zur Kommunikation zwischen verschiedenen Systemen in einem Luftfahrzeug. Das IMA Konzept ersetzt separate Prozessoren und LRU[2]s mit wenigen zentralisierten Prozessoreinheiten, wodurch eine Gewichtseinsparung, sowie eine einfachere Wartung resultiert.

ARINC 429

(Quelle: AIM ARINC429 Specification Tutorial)

Der A429 ist ein, wenn nicht sogar der, herkömmliche Datenbus für kommerzielle Luftfahrzeuge. Zusätzlich wird zwischen einem Highspeed- und einem Lowspeed-Bus unterschieden. Bei dem Highspeed Bus werden Daten mit einer Rate von 100 kBit/s übertragen, während beim Lowspeed Bus die Daten nur mit einer Rate von 12,5 kBit/s übertragen werden.

Die Datenübertragung erfolgt grundsätzlich über einen Rahmen von 32 Bit. Die Übertragung beginnt mit Bit 1, allerdings werden die Bits umgekehrt von Bit 32 aus beschrieben.

Der Rahmen wird in fünf Bereiche unterteilt:

[2] Bauteil oder Baugruppe das bei Wartung oder Instandsetzung auswechselbar ist

Tabelle 1 A429 Frame

Bit	Funktion
32	Paritätsbit
31 - 30	Sign/Status Matrix (SSM), in welchem verschiedene Zustände codiert werden: - Normal Operation (NO) - Functional Test (FT) - No Computed Data (NCD) - Failure Warning (FW)*
29 - 11	Datenfeld
10 - 9	Source/Destination Identifier (SDI), in welchem entweder das Empfangsmodul oder das Sendemodul definiert ist
8 - 1	Label, welches den Datentyp der zu übertragenden Daten beschreibt

*

Normal Operation (NO)	Daten sind korrekt
Functional Test (FT)	Daten dienen zu Testzwecken
No Computed Data (NCD)	Daten werden als fehlerhaft eingeschätzt z.B. außerhalb des erwarteten Wertebereiches
Failure Warning (FW)	Daten werden aufgrund einer internen Fehlfunktion als fehlerhaft eingeschätzt

ARINC 664 / AFDX

(Quellen: AIM AFDX Tutorial; Abaco Systems AFDX/ARINC 664 Protocol)

Die Punkt-zu-Punkt Verbindung des A429, welcher aufgrund dessen eine enorme Kabelmenge beanspruchte und damit auch viel Gewicht, sollte durch ein neues Flugfahrzeug Datennetzwerk ersetzt werden.

„Avionic-Full-Duplex Switched Ethernet“ (AFDX), auch bezeichnet als ARINC 664, ist eine Spezifikation eines deterministischen Luftfahrzeug Datennetzwerkes, welches in Flugzeugen, Eisenbahnen und in militärischen Systemen angewendet wird.
Das System basiert auf der IEEE 802.3 Ethernet Technologie, durch welche die allgemeinen Kosten und die Wartung, durch die herkömmlichen Ethernet Komponenten, gesenkt werden und zusätzlich auch eine schnellere Systementwicklung gewährleistet wird. Das A429 Bus ist noch auf spezifizierte Komponenten angewiesen.

AFDX/A664 erweitert den Ethernet Standard mit einem deterministischen Verhalten mit einer garantierten Bandbreite sowie Quality of Service (QoS). Durch QoS können systemkritische Übertragungen priorisiert werden um die Latenzzeiten zu reduzieren.

Im Gegensatz zum A429 lässt sich der Aufbau eines AFDX/A664 Frames konfigurieren, weshalb eine bitgenaue Beschreibung nicht eindeutig anfertigen lässt.

Der Frame beginnt aus einem Ethernet Header, in welchem jedoch die Empfangs- beziehungsweise Sende-IP-Adresser durch einen jeweiligen virtuellen Link (engl. „Virtual Link“) ersetzt wird und einem AFDX/A664 Header.

Die zu übertragenden Daten werden als FDS Paar versendet, in welchem jeder zu übertragende Datensatz einen funktionellen Status gesetzt bekommt, wobei dieser auch mehreren Datensätzen zugeteilt werden kann.

Der funktionelle Status kann mehrere Zustände annehmen:

Normal Operation (NO)	Daten sind korrekt
Functional Test (FT)	Daten dienen zu Testzwecken
No Computed Data (NCD)	Daten sind als fehlerhaft eingeschätzt
No Data (ND)	Keine Daten vorhanden

CAN

(Quelle: https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en

Der Controller Area Network (CAN) Bus ist ein robustes Nachrichten basiertes serielles Bussystem, mit dessen Hilfe es möglich ist, bei Embedded Systems, Kabelbäume zu vermindern und somit Kosten und Gewicht zu sparen. Damals entwickelt hauptsächlich für die Autoindustrie, wird der Bus heutzutage in nahezu jedem Fortbewegungsmittel aufgrund der genannten Vorteile eingesetzt.

Der CAN-Bus besitzt trotz einer geringen Datenübertragungsrate von 1 Mbit/s kleinere Latenzzeiten, eine schnellere Fehlererkennung und eine schnellere Fehlerbehebung als beispielsweise eine Übertragung über das TCP/IP Protokoll.

Jede CAN Nachricht beginnt mit einem dominanten Bit, welches auf 0 gesetzt wird um den ECUs zu signalisieren, dass eine CAN Nachricht folgt.

Anschließend folgt ein CAN-Identifier, welcher zwischen jeder CAN Nachricht von Bit 2 bis Bit 30 eindeutig differenziert. Mit dem Remote Transmission Request (RTR) Bit lassen sich Nachrichten von anderen ECUs anfordern.

Vor dem eigentlichen Datenfeld, welches zwischen 0 und 64 Bit lang sein kann, wird im Control Bereich des Frames die Länge des zu übertragenden Datenfeldes in Byte angegeben.

Auf das Datenfeld folgen 16 CRC-Bit, mit denen die Integrität der Daten gewährleistet wird und das Resultat der CRC Überprüfung wird im Acknowledge Bereich angegeben.

Am Ende der Übertragung wird das End Of Frame (EOF) gekennzeichnet.

Abbildung 3 CAN Frame[3]

[3] Quelle: https://canlogger1000.csselectronics.com/img/CAN-Bus-Dummies-Simple-Message-Overview-SOF-RTR-CRC-ACK.png

2 Projektumgebung

2.1 GitLab

GitLab ist eine webbasierte Anwendung um Softwareprojekte auf Basis von git zu verwalten.

Branches

Es ist möglich in GitLab das Projekt in mehrere Branches aufzuteilen, somit ist es möglich das Projekt auf einem master-Stand zu behalten und neue Funktionen in lokalen Branches zu bearbeiten. Somit ist das Gesamtprojekt nicht gefährdet, wenn eine neue implementierte Funktion nicht so funktioniert wie sie angedacht war, sondern dass lediglich der Fehler im Branch behoben werden muss, der master[4] Branch zur selben Zeit jedoch fehlerlos vorliegt.

Ist ein Entwicklungsbranch vollständig, so kann dieser wieder zurück in den master Branch gemerged werden.

Diese Arbeitsweise anhand von verschiedenen Branches, erhöht die Projektübersicht, sowie die Fehlerrate wird reduziert, da die Funktion unabhängig vom master entwickelt werden kann und bis zum finalen Merge mögliche Fehler gar nicht erst in den master kommen.

[4] Permanenter Haupt-Branch

Issues

In GitLab können Issues angelegt und verwaltet werden. Dieses Fehler- und Change-Management ermöglicht es Prozesse über längere Zeit zu verwalten, Fehler hervorzuheben und deren Ausbesserung zu dokumentieren.

Bei einer neuen Idee, kann ein solcher Issue angelegt werden, Arbeitskollegen ist es möglich über diesen Issue bei der Entwicklung der Idee oder des Prozesses auf dem Laufenden zu bleiben, Kommentare zu hinterlassen sowie ist es möglich Commits[5], welche in Folge des Prozesses getätigt wurden in dem Issue zu hinterlegen, welches die Rückverfolgung eines speziellen Commits erleichtert.
Issues können individuell auf dem Board verwaltet werden. Hierzu zählt die Zuweisung an einzelne Kollegen, beziehungsweise Gruppen, das Setzen von Flags, sprich dem Markieren des Issues anhand der aktuellen Lage (in Bearbeitung, Abgeschlossen, …) und dem dokumentieren der geleisteten Arbeitszeit.

Am Ende jedes Arbeitstages werden pro Mitarbeiter die geleisteten Arbeitsstunden den verschiedenen Issues zugewiesen. Dies sorgt für einen Stundengenauen Überblick über die geleisteten Arbeiten am Projekt, jedoch dient es eher zur Auswertung. Für künftige Projekte können anhand der Daten leichtere Prognosen betreffend der benötigten Zeit für spezielle Arbeiten getroffen werden, welches das planen von Projekten präziser gestaltet.

[5] bestätigende Freischaltung einer Änderung

CI / CD

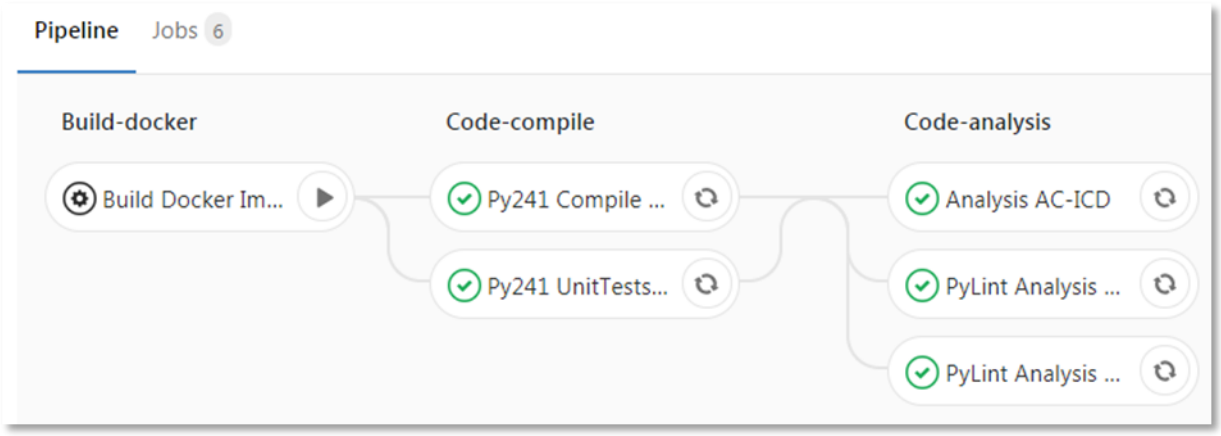

Abbildung 4 Pipeline

In GitLab wird die kontinuierliche Integration durch Pipelines repräsentiert, in welchen über definierte stages mehrere Jobs ausgeführt werden können, um eine dauerhaft erhöhte Qualität des Programmcodes zu garantieren. In der Pipeline werden mehrere Jobs definiert, welche wiederrum in verschiedene stages unterteilt sind (siehe Abbildung 4 Pipeline).

Die erste Stage beinhaltet neben Unit-Tests noch eine versionsbedingte Syntaxüberprüfung des Programmcodes, in welchem überprüft wird, ob die Syntax des Programmcode auf die von uns verwendete Python 2.4 Version passt.

Die zweite Stage beinhaltet Jobs, welche ebenfalls den Programmcode auf Syntaxfehler überprüfen, jedoch wird der Code hierbei mit PyLint überprüft, welches mit der Python Version 2.7 ausgeführt wird.

Die einzelnen Jobs werden vom GitLab-Server auf Clients (Runner) verteilt, in welchem die Jobs innerhalb eines Dockers ausgeführt werden.

Am Ende der Ausführung einer jeden Pipeline wird ein Ergebnis ausgegeben, entweder ist die Pipeline erfolgreich ausgeführt worden (Passed) oder es wurden Fehler gefunden, in welchem Fall der Status auf Failed gesetzt wird. Das Ziel ist es zu jederzeit die Pipeline erfolgreich durchlaufen zu lassen, um sicher zu stellen, dass der Code nicht nur funktioniert, sondern ebenfalls auch leserlich und nachvollziehbar geschrieben ist.

Da unsere Test-Skripte in Python formuliert sind, bei denen die einzelnen Programmteile erst bei der Ausführung kompiliert werden, wird in unserer Pipeline auf eine Überprüfung der Kompilierung verzichtet.
Da in unserem Projekt keine Anwendung kontinuierlich ausgeliefert werden muss, wie beispielweise eine Webseite oder Firmeninterne Software wird in unserem Projekt das Prinzip der Kontinuierlichen Bereitstellung nicht bewusst angewandt.

Wiki

Innerhalb der Projektumgebung auf GitLab ist eine Dokumentationsfunktion implementiert, in der berechtigte Anwender wichtige Informationen oder Prozesse des Projektes dokumentieren können. Dies kann zum Beispiel als Einsteiger Anleitung dienen, sodass falls ein neuer Entwickler dem Projekt beitritt, dieser alle wichtigen Informationen zur Einrichtung direkt verfügbar hat.

Charts

In GitLab ist es dem Anwender möglich das Projekt mithilfe von gesammelten Daten und generierten Graphen zu analysieren. Zum einen ist es möglich das Projekt auf Commits zu überprüfen, diese werden als Kreisdiagramm und in zeitlicher Abfolge als Balkendiagramm. Zum anderen ist es möglich die ausgeführten Pipeline Jobs zu analysieren, hier wird zusätzlich zu den zeitlich angeordneten Graphen auch die

Passed zu Failed Rate angezeigt, welche aufzeigt wie sorgfältig im Projekt anhand von den Anforderungen gearbeitet wurde. Eine Teaminterne Zielvorgabe war es bei den Commits eine Passed zu Failed Rate von etwa 80 % zu erreichen, dies würde dafür sorgen, dass das Projekt von Beginn an sauber geführt wird.

2.2 Testanlage

Die UUT befindet sich in beim Kunden in einem Labor, in welchem mehrere Testschränke, genannt ATE, aufgebaut sind. In einer dieser ATE Anlagen, oder auch Bench genannt, ist unser Testobjekt montiert.

Die ATE ist mit zwei Rechnern verbunden, einen um die Testanlage zu steuern und einen für Data Loading. Über unsere virtuellen Maschinen können wir uns zusätzlich über eine weitere Remoteverbindung auf den ATE Rechner anmelden. Zum einen befindet sich hier der Daedalos Server, zum anderen lassen sich hier mit dem ADS-2 Session Manager neue Konfigurationen laden oder es werden weitere Debugging Anwendungen bereitgestellt.

Die Testanlage verfügt über eine Netzwerkgesteuerte Stromversorgung, mit der wir über den ATE Rechner einzelne Plätze oder die gesamte Stromversorgung steuern können.
Ebenfalls in die Testanlage integriert, sind mehrere Messgeräte, mit deren Hilfe vor Ort nahezu alle wichtigen Daten zu messen sind.

2.3 Jetbrains PyCharm

PyCharm ist eine Integrierte Entwicklungsumgebung, der tschechischen Firma JetBrains, welche sich auf Python spezialisiert. PyCharm bietet eine hervorragende Coding Assistenz und Analyse, sowie eine ebenso hervorragende Projekt- und Code Navigation.

Änderungen, welche entweder vom Server geholt, beziehungsweise zum Server gesendet werden müssen lassen sich von PyCharm aus tätigen. Die Möglichkeit in der Programmieroberfläche schnell die Branches zu wechseln, Änderungen zu verwerfen oder auf die Seite zu legen machen das Arbeiten mit der Entwicklungsumgebung sehr intuitiv und Verzögerungsfrei.

2.4 VMware

Beim Kunden ist für die UUT eine Testanlage vorhanden, dabei handelt es sich um einen etwa 2m hohen Schrank, in welchem diverse Testsysteme verbaut sind. Da neben uns auch Mitarbeiter des Kunden bei Bedarf auf der Testanlage arbeiten, befindet sich die Testanlage nicht in der Firma Konzept. An der Testanlage ist ein Computer angeschlossen, über welchen die Testanlage gesteuert wird. Dieser Computer ist mit Hilfe von virtuellen Maschinen, welche wir zur Verfügung gestellt bekommen von der Ferne aus zu erreichen. Der Zugang auf die VM erfolgt über den Horizon Client von VMware, mit Hilfe eines RSA Security Tokens.

2.5 DOORS

Bei DOORS handelt es sich um eine Client-Server Anwendung von der IBM Tochtergesellschaft Rational Software. DOORS ermöglicht eine strukturierte Aufzeichnung und Verwaltung von Requirements, welche jeweils als eigene Objekte dargestellt werden und mit Attributen versehen werden können. In tabellarischer Ansicht werden neben dem Objekt auch die Attribute angezeigt, je nachdem wie man die Anzeige konfiguriert. Objekte können ebenfalls miteinander verlinkt werden, um die Verfolgung von Anforderungen (Tracing) im Projektablauf zu ermöglichen.

2.6 Daedalos

Daedalos ist eine Client-Server basierte kundenspezifische Anwendung zur Organisation von Test Skripten im Avionik Bereich. Über mehrere Virtuelle Maschinen ist es über die Anwendung möglich gleichzeitig Test Skripte an die Testanlage zu übermitteln und die Reihenfolge der Ausführung zu organisieren.

2.7 Arbeitspakete

Konzept Informationssysteme hat für die Bearbeitung des Auftrages mehrere Arbeitspakete zugewiesen, von denen zwei während meiner Studienarbeit abgedeckt werden. In den jeweiligen Arbeitspaketen werden automatisierte Tests formuliert um den CAN und den A429 Standard auf die korrekte Erfüllung der gegebenen Requirements zu überprüfen. In erster Linie werden Functional und teilweise Performance Requirements getestet, währenddessen die Analyse Anforderungen innerhalb eines späteren Projektabschnittes bearbeitet werden.

Innerhalb des CAN Arbeitspaketes sollen innerhalb von neun erstellen Test Skripten zwanzig Requirements überprüft werden. Für das A429 Interface wurden ebenfalls

zwanzig Requirements formuliert, welche von uns innerhalb von sechs Test Skripten abgedeckt werden.

Beide Arbeitspakete wurden jeweils auf einen Zeitraum von einem Monat angesetzt und sollten nach Lieferung der Test Cases vom Kunden validiert werden.

Das erste Arbeitspaket sollte ursprünglich den A429 Bus verifizieren, allerdings wurde dieses aufgrund einer Konfigurationsstörung auf der Testanlage einen Monat nach hinten verschoben und es wurde zunächst das CAN Arbeitspaket bearbeitet.

2.8 Projektaufgaben

Meine Aufgaben innerhalb des Projektes setzen sich aus verschiedenen Anforderungen zusammen, welche erfüllt werden müssen um das neue IMA System auf seine korrekte Ausführung zu prüfen.

- Gruppierung von Requirements für einen gemeinsamen Test Case
- Definition und Dokumentation von Test Cases
- Implementation Test Skripte

Zunächst müssen aus dem ERD Anforderungen zusammengefasst und einzelnen Tests zugewiesen werden, sodass ähnliche Anforderungen innerhalb eines Testes überprüft werden können.

Sind die Anforderungen auf die Tests verteilt, werden die einzelnen Test Skripte konstruiert. Die Konstruktion erfolgt in der DOORS Anwendung. Hier wird ein Objekt für den jeweiligen Test angelegt und die Parameter werden ausgefüllt, hierzu zählen die allgemeine Beschreibung des Testes, die spezifische Beschreibung der einzelnen

Test Schritte, die Auflistung der abgedeckten Requirements sowie die Zuweisung der abgedeckten Requirements auf die einzelnen Test Schritte und das Einfügen eines Links, in welchem das Test Skript später direkt aufrufbar ist.

Als nächstes müssen die ADS-2 Projekte konfiguriert werden. Aufgrund von Performanceproblemen, welche auftreten, wenn zuviele Current Value Table (CVT, siehe 3.3 Testimplementierung)-Punkte innerhalb einer Konfiguration geladen werden müssen. Aus diesem Grund werden für die verschiedenen Test Cases eigene Konfigurationen erstellt, in denen nur die jeweilig genutzten CVT-Punkte definiert werden.

3 Projektdurchführung

Den Ausgangspunkt des Projektes bildet das ERD in DOORS, in welchem die Requirements, welche wir abdecken sollen, aufgelistet und beschrieben sind. Die erste Aufgabe besteht aus dem Zusammenfassen von einzelnen Requirements, sodass innerhalb von einem Test Skript ähnliche Requirements abgedeckt werden können.

Das Ziel hierbei ist es, Überlappungen der abgedeckten Requirements in den Test Cases zu vermeiden. Hierbei muss bereits abgeschätzt werden, wie die einzelnen Requirements mit Hilfe eines Python Skriptes abgedeckt werden können, um so die Komplexität der Test Skripte einschätzen zu können.

Innerhalb des ERD, werden im Anschluss die Gruppierungen eingetragen und den Mitarbeitern zugewiesen um unnötige Redundanz zu verhindern.

3.1 Equipment Verification Specification

Im Anschluss werden die einzelnen Tests geplant und theoretisch konstruiert. Mit Hilfe der DOORS Anwendung werden nun die Test Skripte dokumentiert. Hier wird ein Objekt für den jeweiligen Test angelegt und die definierten Parameter werden ausgefüllt, hierzu zählen die allgemeine Beschreibung des Testes, die spezifische Beschreibung der einzelnen Test Schritte, die Auflistung der abgedeckten Requirements sowie die Zuweisung der abgedeckten Requirements auf die einzelnen Test Schritte und das Einfügen eines Links, in welchem das Test Skript später direkt aufrufbar ist.

Somit sind die einzelnen Tests theoretisch konstruiert und können im Review, hierbei gemeint ist eine interne Qualitätskontrolle, insofern geprüft werden, ob die Tests die jeweiligen Requirements auch vollständig testen.

Das Review der Spezifikation erfolgt durch den Projektleiter oder durch einen erfahrenen Kollegen innerhalb des Teams.

3.2 ADS-2 Projekt

Ist die Spezifikation erfolgreich durch das Review gekommen, muss im Vorfeld auf die Python Implementierung das ADS-2 Testanlagen Projekt konfiguriert werden. Da jeder Test Skript unterschiedliche Anforderungen überprüft, muss das ADS-2 Projekt spezifisch auf das jeweilige Test Skript angepasst werden. Innerhalb der ADS-2 Projekte können alle definierten Übertragungskanäle individuell angepasst werden.

Das ADS-2 Projekt wird mit Hilfe des AC-ICDs angepasst, in welchem für jeden Input und jeden Output Frame wichtige Daten angegeben werden, darunter zählen Länge der Nachricht, der assoziierte Port der Übertragung, Benennung des korrespondierenden CVT-Punktes (s.3.3 Testimplementierung) und des zu übertragenden Datentyps, Definition der Position beziehungsweise Bit- /Bytelänge der verschiedenen Nachrichtenbereiche (Funktioneller Status, Datenfeld…), Übertragungsrate, Angabe des Least Significant und des Most Significant Bit sowie zahlreichen anderen Parametern.

Die Konfiguration des ADS-2 Projektes ist sehr schwierig, da trotz zahlreicher Anhaltspunkte wie sich die Testanlage verhalten soll, zusätzlich noch einige Einstellungen vorhanden sind, welche im schlimmsten Fall mit der Erstellung des Python Skriptes erst getätigt werden können, da sich das ADS-2 Projekt beim Ausführen eines Testes immer noch nicht so verhält, wie gewünscht.

3.3 Testimplementierung

Ist die ADS-2 Konfiguration abgeschlossen, wie bereits gesagt können immer noch Fehler aufkommen, wird letztendlich mit Hilfe der Anwendung PyCharm das jeweilige Test Skript implementiert.

Bei der Erstellung des Test Skriptes ist die oberste Priorität die Dokumentation, in DOORS und im Test Skript selber, sowie die Leserlichkeit des Test Skriptes, sodass dritte Personen, ohne das Test Skript lange nachvollziehen zu müssen, den Ablauf des Skriptes einfach und schnell verstehen können. Um dies zu gewährleisten wird auf Redundanz Optimierung verzichtet, da sonst die Skripte zu Umfangreich werden.

```
14 from iom_evs_ivvp_0009 import IomEvsIvvp0009
15 from iomtecl.iomconst import POS_01
16
17 CAN_BUS_INSTANCE = '01'
18 POSITION = POS_01
19 CAN_SPEED = '83K'
20
21 IOM_EVS_IVVP_0009 = IomEvsIvvp0009(bus_instance=CAN_BUS_INSTANCE,
22                                    position=POSITION)
23 IOM_EVS_IVVP_0009.run()
```

Abbildung 5 Instanziierung

Jeder Test Case besteht aus einem zentralen generischen Skript sowie aus verschiedenen Instanziierungsskripten, welche das generische Skript mit unterschiedlichen Parametern ausführen können. Beispielsweise können über verschiedene Instanziierungsskripte verschiedene Übertragungsgeschwindigkeiten definiert werden, da hier die Testanlage auf die bestimmte Übertragungsgeschwindigkeit angepasst werden muss, sowie das UUT in verschiedenen Positionen innerhalb des Luftfahrzeuges simuliert werden kann.

Diese Auftrennung resultiert aus der Tatsache, dass die momentane UUT noch ein älteres RDC System ist und sobald der erste Prototyp des neuen Produktes vorhanden ist es nur noch Anpassungen innerhalb des Instanziierungsskriptes gemacht werden müssen. Beispielsweise besitzt die UUT zur Zeit der Projektarbeit nur zwei A429 Instanzen, währenddessen das neue Produkt bis zu 15 Instanzen bereitstellen soll. Somit müssen nur neue Instanziierungsskripte angelegt werden, anstatt das generische Skript überarbeiten zu müssen.

Innerhalb des generischen Test Skriptes wird der eigentliche Test formuliert. Anfangs muss zunächst die Session erstellt und die jeweiligen CVT Punkte geladen werden. Mit Hilfe der konfigurierten CVT Punkte wird der Datenaustausch zwischen dem Python Test Skript und dem ADS-2 Projekt gewährleistet. Von Python Seite aus werden die jeweiligen Daten, zum Beispiel ein zu überprüfender Wert, in eine Python Variable geschrieben, welche den CVT Punkt repräsentiert. Innerhalb der ADS-2 Konfiguration ist definiert, wie die Testanlage den CVT Punkt zu interpretieren hat, also die Positionierung der einzelnen Daten innerhalb des CVT Punktes, sodass diese korrekt interpretiert werden, sowie andersherum die Daten wieder korrekt in die CVT-Punkte geschrieben werden.

Anschließend wird die entsprechende Konfiguration über HPP aus einem Multi-Position-Load gewählt / gestartet. Je nach Pin Belegung wird hier aus einer Konfiguration der jeweilige Speicherbereich aus dem Flashspeicher in den RAM des Gerätes geladen. Ist die UUT erfolgreich gestartet, werden die einzelnen Test Schritte ausgeführt, hierbei ist es wichtig das Gerät in dem richtigen Modus zu setzen. Die UUT kann verschiedene Zustände annehmen, in welchem er sich unterschiedlich verhält. Zum Beispiel gibt es die beiden Zustände on_Ground und in_Flight, welche respektive angeben ob sich das Luftfahrzeug und somit auch das Gerät in der Luft oder am Boden befindet. Befindet sich das Luftfahrzeug am Boden, wird beispielsweise bei einem

Anschaltvorgang, hierbei egal ob durch normales Anschalten, einen Neustart oder einen Reset, eine Selbstüberprüfung der einzelnen Busse durchgeführt, welche bis zu zwanzig Sekunden dauert. Diese Selbstüberprüfung sollte, jedoch nicht stattfinden, wenn sich das Gerät in der Luft befindet, da in der Zeit die UUT keinerlei andere Funktionen ausführen kann, was in einem kritischen Moment in der Luft nicht passieren darf, sodass dieser in_Flight innerhalb von zwei Sekunden wieder Einsatzbereit ist.

Weiterführend kann die UUT neben den beiden Zuständen, noch mehrere konfigurierte Status annehmen in denen definiert ist, wie sich die UUT bei der Übertragung verhält. Im Total_Bus_Silent und Configured_Bus_Silent werden beispielsweise keine Frames versendet, weshalb sich die UUT bei Übertragungen nicht in den Modi befinden sollte.

Der benötigte Zustand der UUT kann je nach Test Schritt, auch innerhalb eines Test Skriptes angepasst werden, muss jedoch im Zuge dessen auch dokumentiert sein. Für die Dokumentation eines Test Schrittes gibt es in unserem Projekt eine erstellte Vorlage (siehe Abbildung 6 Test Beschreibung).

```
# ------------------------------------------------------------------------------#
# Step Number:          0002                                                    #
# Step Short Text:      Encoding Float                                          #
# ------------------------------------------------------------------------------#
# Covered Requirements:                                                         #
#  - IOM-ERD-REQ-0230                                                           #
#  - IOM-ERD-REQ-0488 Float                                                     #
# ------------------------------------------------------------------------------#
# Step description:                                                             #
# This step is done first with FS set to NO and second FS set to FT.            #
# 1) Verify lower limit value is encoded correctly                              #
# 2) Verify upper limit value is encoded correctly                              #
# 3) Verify quiet value is encoded correctly                                    #
# 4) Verify signalling value is encoded correctly                               #
# 5) Verify positive infinity value is encoded correctly                        #
# 6) Verify negative infinity value is encoded correctly                        #
# ------------------------------------------------------------------------------#
# Used values:                                                                  #
# FLOAT_MIN         = 0x00800000                                                #
# FLOAT_MAX         = 0x7F7FFFFF                                                #
# FLOAT_QUIET       = 0x7FFFFFFF                                                #
# FLOAT_SIGNAL      = 0x7FBFFFFF                                                #
# FLOAT_POS_INF     = 0x7F800000                                                #
# FLOAT_NEG_INF     = 0xFF800000                                                #
# ------------------------------------------------------------------------------#
```

Abbildung 6 Test Beschreibung

Neben der Angabe der Nummerierung und einer kurzen prägnanten Bezeichnung, werden alle abgedeckten Requirements angegeben sowie ein schrittweiser Ablauf des Test Schrittes. Die benutzten Werte des Test Schrittes werden ebenfalls angegeben, da beispielsweise die Übertragung von Opaque oder Bitfeldern mit binären Sequenzen getestet wird, welche nicht direkt nachvollziehbar sind. Angegeben werden unsere Test Werte im hexadezimalen System, um Test Skripte übergreifend gleich zu implementieren.

Die unterschiedlichen Test Skripte werden so gut möglich nach demselben Schema aufgebaut. Am Anfang werden in die entsprechenden Frames über CVT-Punkte die Test Werte gesetzt, welche auf der gegenüberliegenden Seite verifiziert werden können. Beispiel:

Im Test „IOM_EVS_IVVP_0007“ (DOORS-ID) soll die korrekte Dekodierung von CAN Frames überprüft werden. Um dies zu verifizieren, wird das ADS-2 Projekt auf eine Übertragung von einem CAN Bus auf einen A664 Bus konfiguriert. In dem man in über einen CAN Input Frame einen Test Wert auf einen A664 Output Frame überträgt, kann so verifiziert werden, dass der CAN Frame korrekt dekodiert wird, da der Wert korrekt im A664 System ankommt.

Für jeden im Requirement vorhandenen Datentyp wird ein Test Schritt implementiert. Einer der verlangten Datentypen wäre zum Beispiel ein 8-Bit Integer Wert (signed). Der entsprechende Test Schritt soll mit ausgewählten Werten die korrekte CAN Dekodierung verifizieren. Dies soll wie beschrieben so einfach und so leserlich wie möglich geschehen. Numerische Datentypen können relativ einfach und schnell auf korrekte Dekodierung überprüft werden, da hier die einzelnen Grenzwerte überprüft werden können. Wenn die Grenzwerte korrekt dekodiert werden, werden auch alle anderen möglichen Werte korrekt dekodiert.

Bei einem signed 8-Bit Integer Datentyp sind die entsprechenden Grenzwerte zum einen 0x7F (dez. 127, upper limit value) und zum anderen 0x80 (dez. -128, lower limit value[6]). Die negativen Zahlen werden mit dem Zweierkomplement gebildet.

[6] Unterer Grenzwert

Um die korrekte Dekodierung zu verifizieren, werden die beiden Grenzwerte nacheinander über den entsprechenden 8-Bit Integer CAN CVT-Punkt in den CAN Input Frame geschrieben und können nach dem jeweiligen Setzen über einen A664 CVT-Punkt aus dem A664 Output Frame gelesen und verglichen werden. Wenn die Werte übereinstimmen ist die korrekte Dekodierung des jeweiligen Datentyps verifiziert. Im A664 Standard muss ebenfalls nach dem korrekten funktionellen Status überprüft werden (siehe 1.3 Verwendete Technologien - ARINC 664 / AFDX).

Wird der Wert falsch dekodiert, kann das verschiedene Gründe haben. Der Fehler könnte im Test Skript liegen, z.B. falsche Überprüfung der Werte, er könnte im ADS-2 Projekt liegen, z.B. falsche Konfiguration der Positionierung innerhalb eines Frames oder aber die UUT befindet sich beispielsweise aufgrund eines vorherigen Tests in einem falschen Zustand. Wenn trotz der richtigen Konfiguration aller genannten Parameter die Verifizierung nicht funktioniert, ist es ebenfalls möglich, dass innerhalb unserer Library ein Fehler war oder aber mit der UUT, letzteres aber eher unwahrscheinlich.

Wenn das gesamte Test Skript erfolgreich durchläuft und somit alle zu überprüfenden Requirements verifiziert, findet das Review in Form eines Vier-Augen-Prinzips statt zusammen mit dem Kunden statt.

Eventuell auftretende Fehler werden in Form von Findings gekennzeichnet, geprüft, bewertet und gegebenenfalls korrigiert. Erst wenn der Test keine Findings mehr aufweist, wird dieser abgeschlossen.

Im Vorfeld wurde mit dem Kunden vereinbart, dass zu einem gewissen Meilenstein die ersten Tests weiter geleitet werden, dass das formale Review durchgeführt werden kann.

4 Fazit

Das Projekt war in vielerlei Hinsicht äußerst interessant. Zunächst einmal bin ich das erste Mal in ein Projekt eingestiegen, welches mit mir zusammen neu begonnen hat, da ich in der ersten Praxisphase in der Weiterentwicklung der firmeninternen Verwaltungsanwendung beteiligt war. Es war sehr interessant zu sehen, wie ein Projekt von neuem begonnen wurde. Mit GitLab habe ich bereits in meiner ersten Praxisphase gearbeitet und war somit auch schon mit den zahlreichen Funktionen vertraut. In der Theoriephase hatten wir ebenfalls mit GitHub gearbeitet, in welchem einige Funktionen, wie zum Beispiel Issues, Pipeline… zwar nicht verfügbar sind, allerdings der Grundaufbau derselbe ist.

Da ein Kollege aus unserem Team bereits an der Entwicklung eines Schwestermodells gearbeitet hatte, war es seine Aufgabe die Grundstruktur des Projektes anzulegen. Generell war es sehr hilfreich jemanden im Team zu haben, der bereits mit der verwendeten Technik vertraut ist, da vor Allem in den ADS-2 Projektkonfigurationen mehrmals Fragen aufgetaucht sind und ich so auch sehr guten Input erhalten konnte. So war es möglich, mit Hilfestellungen, die auftretenden Probleme so gut es ging selbstständig zu bearbeiten, möglichst ohne den Projektleiter oder Vollzeitmitarbeiter dauerhaft zu beschäftigen.

Bei der Durchführung des Projektes habe ich ebenfalls viel Neues gelernt, da ich noch nicht die Möglichkeit hatte Anforderungen eines Firmenkunden zu verwalten, sprich die Requirements die wir erhalten haben selbst zu bearbeiten. Der Ausgangspunkt war hierbei größtenteils das ERD des Kunden. Ausgehend von diesem Dokument mussten wir im Team die Requirements zunächst komplett verstehen, da diese sehr speziell formuliert sind und entsprechend von Ähnlichkeiten untereinander bestmöglich in Tests miteinander zu verknüpfen und abzuarbeiten.

Die Analyse der Requirements und die Definition von den einzelnen Test Skripten war eine anspruchsvolle und gewinnbringende Aufgabe.

Die letztendliche Programmierung der Python Test Skripte war persönlich etwas anstrengend, nicht weil diese schwierig zu Programmieren waren, sondern weil diese möglichst einfach sein sollten. Als Programmierer war jeweils der erste Reflex die Test Skripte mit so wenig Redundanz und so kurz wie möglich zu formulieren, jedoch sollen diese Test Skripte so einfach wie möglich sein und sollen in Folge dessen haufenweise Redundanz tatsächlich aufweisen. Der Hintergrund, dass auch Personen, welche nicht alltäglich programmieren, die Tests nachvollziehen und verifizieren können ist durchaus plausibel, hat anfangs allerdings meinem ersten Reflex widersprochen. Aus diesem Grund gab es auch Tests, welche im Review mit zahlreichen Findings zurückkamen, da diese zwar funktionell richtig waren, allerdings deutlich zu viele Kontrollstrukturen aufwiesen, welche den Programmcode zu kompliziert gestaltet haben.

Der einzige Punkt, der mir im Laufe des Projektes Probleme gemacht hat, war die Tatsache, dass ich erst im letzten Drittel der Praxisphase die Möglichkeit hatte die Testanlage in der Realität zu sehen, vorher hatte ich zwar einen Überblick über die Testanlage, allerdings nur über Skizzen und mündlichen Erklärungen, somit hat mir ein wenig der Bezug zur tatsächlichen Testanlage gefehlt.

5 Quellen

Informationsquellen:

Abaco Systems (2016) (abgerufen am 24.09.2018):
AFDX/ARINC 664 Protocol

AIM (2010) (abgerufen am 24.09.2018):
AFDX Tutorial
ARINC429 Specification Tutorial

CSS electronics (2018) (abgerufen am 24.09.2018):
https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en

Bildquellen:

***Bilder ohne Quellenangabe stammen aus dem internen Bereich der Konzept Informationssysteme GmbH oder wurden persönlich erstellt*

Remote Data Concentrator:
https://www.diehl.com/aviation/en/diehl-aviation/portfolio/portfolio-avionics

CAN Frame:
https://canlogger1000.csselectronics.com/img/CAN-Bus-Dummies-Simple-Message-Overview-SOF-RTR-CRC-ACK.png