

Management Summary

Inhalt

[ETIC2 zur Verwaltung von Ventiltests 6](#_Toc491525510)

[1 Einleitung 6](#_Toc491525511)

[2 Grundlagen 7](#_Toc491525512)

[2.1 Situation 7](#_Toc491525513)

[2.1.1 Controller 7](#_Toc491525514)

[2.1.2 Firmware 8](#_Toc491525515)

[2.1.3 Test 8](#_Toc491525516)

[2.2 Testumgebung 9](#_Toc491525517)

[2.2.1 Ventil Hardware 9](#_Toc491525518)

[2.2.2 Test Hardware (NI PXI) 10](#_Toc491525519)

[2.2.3 Test 10](#_Toc491525520)

[2.2.4 ParameterStructBuild 11](#_Toc491525521)

[2.2.5 TestUpdateFirmware 12](#_Toc491525522)

[2.2.6 TTIC2 (Test Tool) 12](#_Toc491525523)

[2.2.7 SoftwareVersionsDatabase 14](#_Toc491525524)

[2.2.8 Firmware Database 14](#_Toc491525525)

[3 Zielsetzung 16](#_Toc491525526)

[3.1 Ziele zum Startzeitpunkt 16](#_Toc491525527)

[3.2 Ziele nach der ersten Analyse 17](#_Toc491525528)

[3.3 Quantitative Ziele 18](#_Toc491525529)

[3.4 Qualitative Ziele 18](#_Toc491525530)

[3.5 Aufgabenbegrenzung 18](#_Toc491525531)

[4 Methodik 19](#_Toc491525532)

[4.1 Vorgehensweise 19](#_Toc491525533)

[4.2 Hilfsmittel 20](#_Toc491525534)

[4.2.1 SoftwareVersionsDatabase 21](#_Toc491525535)

[4.2.2 Test, TestUpdateFirmware, TTIC2 22](#_Toc491525536)

[4.2.3 ETIC2 23](#_Toc491525537)

[5 Auswertung der Testresultate 23](#_Toc491525538)

[5.1 SoftwareVersionsDatabase 23](#_Toc491525539)

[5.1.1 Modellierung Firmware Informationen 23](#_Toc491525540)

[5.1.2 Modellierung Testinformationen 26](#_Toc491525541)

[5.1.3 Modellierung Testresultate 27](#_Toc491525542)

[5.1.4 Modellierung Firmware Bugs 29](#_Toc491525543)

[5.2 Test 30](#_Toc491525544)

[5.2.1 Abspeicherung Testinformationen 30](#_Toc491525545)

[5.2.2 Testinformationen an TestUpdateFirmware über Umgebungsvariablen 32](#_Toc491525546)

[5.3 TestUpdateFirmware 33](#_Toc491525547)

[5.3.1 Anweisung Update Testinformationen 33](#_Toc491525548)

[5.3.2 Abfrage Testinformationen 34](#_Toc491525549)

[5.3.3 Testinformationen in SoftwareVersionsDatabase schreiben 34](#_Toc491525550)

[5.4 TTIC2 34](#_Toc491525551)

[5.4.1 Auslesung der Testinformationen 35](#_Toc491525552)

[5.4.2 Hinterlegung des Grundzustandes 35](#_Toc491525553)

[5.4.3 Abspeicherung der Testresultate 35](#_Toc491525554)

[5.5 ETIC2 36](#_Toc491525555)

[5.5.1 Codierung nach MVVM 36](#_Toc491525556)

[5.5.2 Anbindung SoftwareVersionsDatabase 36](#_Toc491525557)

[5.5.3 Ausgabe Bericht 36](#_Toc491525558)

[6 Ergebnisse 37](#_Toc491525559)

[6.1 Quantitative Ziele 37](#_Toc491525560)

[6.2 Qualitative Ziele 37](#_Toc491525561)

[7 Diskussion 38](#_Toc491525562)

[8 Ausblick 39](#_Toc491525563)

[8.1 Umsetzung Überarbeitung SoftwareVersionsDatabase 39](#_Toc491525564)

[8.2 Integration Buglist in ETIC2 39](#_Toc491525565)

[9 Verzeichnisse 40](#_Toc491525566)

[Literaturverzeichnis 40](#_Toc491525567)

[Abkürzungsverzeichnis 40](#_Toc491525568)

[Abbildungsverzeichnis 41](#_Toc491525569)

[Tabellenverzeichnis 41](#_Toc491525570)

[Glossar 42](#_Toc491525571)

[A. Projektmanagement 43](#_Toc491525572)

[A.1. Zeitplan 43](#_Toc491525573)

[B. Testumgebung 44](#_Toc491525574)

[B.1. TTIC2 44](#_Toc491525575)

[C. Hilfsmittel 45](#_Toc491525576)

[C.1. CVI 45](#_Toc491525577)

[C.1.1. Umgebungsvariable 45](#_Toc491525578)

[C.1.1.1. Gegenüberstellung zu C Libary Umgebungsvariable 45](#_Toc491525579)

[C.2. SQL Toolkit 46](#_Toc491525580)

[C.2.1. Funktionsumfang 46](#_Toc491525581)

[C.2.2 Code Ausschnitt um einen neuen Eintrag zu erstellen 47](#_Toc491525582)

[Selbständigkeitserklärung 49](#_Toc491525583)

# ETIC2 zur Verwaltung von Ventiltests

# Einleitung

# Grundlagen

Das Grundlagenkapitel ist in zwei wichtige Elemente unterteilt. Einerseits wird die Umgebung, die Situation und der aktuelle Stand sozusagen die Grundvoraussetzungen beschrieben. Darauf folgt andererseits die Hauptaufgabe dieser Masterarbeit die Erweiterung der Testumgebung, dabei wird beschrieben was schon entwickelt wurde, wo es Anpassungen geben wird und wie das neue Produkt Evaluation Tool Integrierter Controller 2 (ETIC2) angedacht ist.

## Situation

Dieses Unterkapitel beschreibt die zentrale Steuereinheit für die die Testumgebung entwickelt wird, diese wird benötigt, um ein Ventil steuern zu können. Das Verhalten des Ventils wird durch die Software auf dem Controller übernommen, welche auf Anweisungen des Benutzers wartet. Diese Verhalten werden durch Tests überprüft und erhöhen die Software Qualität.

### Controller

Die Firma VAT stellt Vakuumventile her für die Halbleiter- und Medizinalindustrie, die Forschung und Entwicklung sowie für die Automobilindustrie (VAT Group AG, 2017). VAT ist im Bereich der Herstellung von Vakuumventilen mit einem Marktanteil von über 40% klarer Weltmarktführer. Die Firma ist bekannt für ihre Regelventile, bei denen ein Controller die Steuerung dieser Vakuumventile übernimmt (siehe Abb. 1). Dieser Controller ist modular aufgebaut und besteht grob gesagt aus drei Komponenten. Die wichtigste Komponente ist das Masterboard, die mit den zentralen Elementen bestückt ist. Dieses ist unerlässlich und wird jeweils an die gewünschte Ventilhardware angepasst. Das Herzstück des Controllers ist der Mikrocontroller, für den VAT eine eigene Firmware entwickelt hat. Dazu ist oder sind Motorbausteine nötig, die eine weitere Firmware von externen Lieferanten benötigt. Die zweite Komponente ist das Interface Board. Dieses wird nach Kundenwunsch angefertigt. Falls der Kunde mit einem Feldbus System arbeitet, wird eine Interface Firmware nötig. Der Nutzen von Feldbus Systemen liegt darin, dass von einem Host aus mehrere Teilnehmer angesprochen werden können. Die dritte Komponente ist die Option Unit, die Zusatzfunktionen nach Wunsch beinhaltet. Der Kunde kann mit Hilfe des Controllers seine Sensoren direkt speisen. Eine weitere Option ist, dass bei Spannungsabfall das Ventil eine vordefinierte Position einnimmt. Verwendet der Kunde kein Feldbus System, so kann er mit Hilfe eines Cluster Systems oder über die RS485 Interface Schnittstelle mehrere Ventile ansprechen. (Marugg, 2010).



Abbildung : Basiskonzept Ventil Controller (Marugg, 2010)

### Firmware

Wie bereits im vorherigen Kapitel erwähnt, ist das Masterboard mit einem Mikrocontroller bestückt. Die Gründe für den Einsatz des Mikrocontrollers liegen bei den niedrigen Anschaffungskosten und den fundierten Programmierkenntnissen der Mitarbeitenden mit der Programmiersprache C. Auf dieser läuft die von VAT entwickelte Firmware, welche ohne Betriebssystem auskommt. Die Firmware wartet ständig auf neue Befehle des Benutzers und führt diese nach Ablauf der Zykluszeit immer wieder aus. Um den Anwendern das Leben zu erleichtern, wurde eine Parameterbaumstruktur eingefügt, bei welcher grob gesagt jeder Parameter eine Funktionalität des Ventils wiederspiegelt. Dabei kann es sich um Parameter handeln, die gesetzt werden oder die reine Status Informationen abfragen können. Dies passiert entweder über den Service (USB) oder über den Interface Kanal. Die Parameter sind in vier Ebenen unterteilt, wie sie Abbildung 2 zeigt. Das soll den Anwender bei der Suche nach der gewünschten Einstellung helfen.

Abbildung : Parameterbaumstruktur der Software

### Test

Aufgrund der Komplexität dieser Firmware und der damit verbundenen zunehmenden Fehlerquote, wurde ich von der Firma VAT eingestellt und beauftragt die Qualität der Firmware sicherzustellen. In der Vergangenheit wurde diese Aufgabe vom Software Entwickler selbst übernommen. Dabei kam es immer wieder vor, dass die Zeit für die Überprüfung aus Zeitgründen sehr schmal ausfiel. Die Basis bildete Excel Listen mit Ventilbefehlen, welche sequentiell abgespielt werden und Überprüfkommandos beinhalteten. Die Schwachstellen dieses System lagen in der Wartung sowie in den begrenzten Funktionsmöglichkeiten, welche durch visuelle Überprüfung des Testers übernommen werden musste.

Um diese Probleme für die neue integrierte Controller Generation zwei (IC2) zu lösen, musste ein neues System gefunden werde. Das primäre Ziel war es völlig automatische Tests zu entwickeln, welche keinen Einfluss des Testers benötigten. Dadurch war schnell klar, um Berechnungen durchzuführen und um die Firmware Funktionalität zu prüfen, wird der Einsatz einer Programmiersprache nötig. Zudem soll eine schnelle und einfache Auswertung der Tests möglich sein. Dies erfordert nicht nur den Einsatz eines Tests, sondern einer ganzen Umgebung, welches im nächsten Unterkapitel näher beschrieben wird.

## Testumgebung

Im Unterkapitel Testumgebung werden die einzelnen Elementen der Testumgebung näher beschrieben, welche in der Abbildung 3 ersichtlich sind. Aus Gründen der Leserfreundlichkeit, wird die Beschreibung der Elemente in drei Teile aufgeteilt, und zwar in Ist-Zustand, Stärken und Schwächen sowie Problemstellung.

Abbildung : Ist-Zustand der Testumgebung

### Ventil Hardware

#### Ist-Zustand

Mit Hilfe der Testumgebung werden unterschiedliche Ventil Hardware getestet. Dabei wird der Controller der jeweiligen Ventil Hardware Anforderungen angepasst. Diese beinhaltenen Unterschiede in der Bestückung des Masterboards zum Beispiel an die Anzahl Motoren oder deren Leistung. Das bedeutet, dass es unterschiedliche Controller Ausführungen gibt. Der Grundgedanke am Aufbau des Controllers mit den drei Boards bleibt jedoch gleich wie auch die Struktur der Ventil Firmware. Um die Hardware Eingänge des Controllers überprüfen zu können, wird eine Test Hardware benötigt, mit welcher analoge und digitale Signale erzeugt und ausgelesen werden können.

#### Problemstellung

Über den Lebenszyklus eines Controllers werden immer mehr Ventil Hardware unterstützt. Dies hat zur Folge, dass unterschiedliche Controller Ausführungen erstellt werden müssen. Um diesem Umstand gerecht zu werden, ist eine flexible Testumgebung zwingend. Es müssen alle Controller Typen mit den Tests qualifiziert werden können.

### Test Hardware (NI PXI)

#### Ist-Zustand

Der Controller hat mehrere Schnittstellen zum Hostsystem des Vakuumkammer Betreibers. Einerseits hat jeder Controller einen USB Stecker, welcher verwendet wird, um mit einer Service Applikation das Ventil in Betrieb zu nehmen. Im Betrieb wird das Ventil vom Host über die Interface Schnittstelle angesprochen. Immer häufiger spricht der Host seine Komponenten mit Hilfe einer Feldbus Lösung an.

Für den Low-end Markt wird der Controller mit einer Logik Interface ausgerüstet. Dabei steuert der Host das Ventil mit Hilfe von analogen und digitalen Signalen. Wie auch die Status Informationen des Ventils werden über analoge und digitale Signale an den Host zurückgeliefert.

Um das Ventil wie auch den Controller betreiben zu können, braucht es einen Power Stecker. Dieser Stecker bietet zusätzlich eine Safety Möglichkeit, mit dieser kann das Ventil durch die Hardware geöffnet und geschlossen werden.

All diese Funktionalitäten müssen mit Hilfe der Test Hardware emuliert und geprüft werden können. Dabei bietet National Instruments (NI) eine flexible Kartenbestückungslösung, um über eine grafische Programmiersprache (LabVIEW) oder einer textbasierten Programmiersprache (CVI) das PXI System ansprechen zu können. Dieses System ermöglicht eine kurze Entwicklungszeit der Prüfsoftware für die Controller Hardware, da es sehr viele Hilfsfunktionalitäten zur Verfügung stellt. Dabei gibt es zwei Einschränkungen bezüglich der Speisung und der Unterstützung der CCLink Interface Anbindung. Die Speisung wird mit Hilfe eines separaten Speisegeräts per Software (LabVIEW) gesteuert. Die Kommunikation über CCLink mit dem Controller wird über einen Umweg eines Drittanbieter Produktes erreicht.

#### Stärken

* Schnelle Inbetriebnahme der Hardware über die NI Programmierumgebung
* Fast alle Hardware Schnittstellen können über die NI Hardware getestet werden
* Das System kann solange erweitert werden, bis alle 8 Steckplätze besetzt sind
* Engineering Support durch NI

#### Schwächen

* Jährliche Lizenzkosten für die Programmierumgebung
* Hohe Anschaffungskosten

#### Problemstellung

Die Test Hardware wurde zu einem früheren Zeitpunkt evaluiert und angeschafft. Da noch nicht alle Steckplätze mit Karten ausgestattet sind, kann es noch zu Erweiterungen kommen. Der Grund liegt in der Vergrösserung des Portfolios an Ventilen, welche mit dem IC2 Controller angesprochen werden. Diese einzelnen Controller unterscheiden sich in Bezug auf den Power Stecker. Somit steigt der Bedarf an analogen- und digitalen Karten an.

### Test

#### Ist-Zustand

Bei den geschriebenen Tests handelt es sich um automatisierte Anwendertests, welche mit der CVI Programmierumgebung erstellt sind. Hierin werden die Ventil Funktionalitäten im laufenden Betrieb geprüft. Diese Tests werden eingesetzt, um eine Ventil Firmware zu qualifizieren. Dabei werden diese meistens übers Wochenende oder über Nacht ausgeführt, was zur Folge hat, dass der Anwender nicht mehr aktiv in die Ausführung eingreifen muss. Die Tests werden bei der zugehörigen Ventil Firmware in der SVN hinterlegt, umso bei allfälligen Fehler im Feld eine Analyse durchführen zu können. In den Tests werden die Anforderungen bezüglich der Ventil Hardware definiert. Wichtig ist dabei, dass die Tests möglichst universell eingesetzt werden können.

Im Test wird mit Parameternamen gearbeitet, um dem Code einfacher lesen zu können. Die Identifikationsnummer kann sich so ändern, ohne dass eine Anpassung des Tests notwendig wird. Diese Informationen werden durch die erzeugten Files des ParameterStructBuild Programms bezogen. Die Testfunktionalitäten werden in Files definiert, welche in allen Tests verwendet werden. Im Hauptfile wird der Ablauf eines Tests definiert.

#### Stärken

* Vollautomatische Tests
* Testfunktionen nur in einem File, der Rest sind Library Files
* In einer Struktur sind alle Ventil Parameter hinterlegt sowie seine Enum Werte

#### Schwächen

* Testinformationen werden in einem Textfile hinterlegt
* In den Testinformationen können nicht mehrere Werte für ein Attribut definiert werden

#### Problemstellung

Im Test werden einerseits Informationen zur Kennzeichnung des Tests hinterlegt. Zu diesen gehören eine Beschreibung, die aktuelle Version, der Autor und das Erstellungsdatum. Anderseits wird definiert für welche Ventil Hardware der Test ausführbar ist. Diese Informationen werden im gleichen Ordner, in welchem sich der Test befindet, in einem Textfile hinterlegt. Dieses File wird erzeugt, wenn das Test Tool Integrierter Controller 2 (TTIC2) Programm kein Textfile im Testordner finden kann. So kann der Fall auftreten, dass die Informationen im Test im Vergleich zum Textfile nicht übereinstimmen.

### ParameterStructBuild

#### Ist-Zustand

Dieses Programm liest die aktuellen Parameter der Firmware aus und speichert diese in Strukturform ab. Daraus resultiert ein Header File, in welchem die Struktur definiert wird. Als zweites wird ein Source File generiert, welche den Inhalt der Struktur füllt. Diese mehrdimensionale Struktur spiegelt den Parameterbaum nach. Wichtige Elemente bilden dabei die Identifikationsnummer, welche aus einer achtstelligen Nummer besteht. Dabei werden jeweils zwei Stellen für eine Ebene verwendet. Anschliessend werden die einzelnen Ebenen in Textform definiert. Den Datentyp, der Zugriff sowie den Wertebereich des Parameters wird ausgelesen. Dazu enthalten einige Parameter Enum Werte, welche den Parameterwert in Textform beschreibt. Diese wird auch in dieser Strukturform hinterlegt. Diese Informationen werden später in den Tests genutzt, sodass der Test Schreiber sieht, welche Parameter von der Firmware unterstützt werden. Er muss keine Kenntnisse über die Identifikationsnummer haben. Die Enum Werte erlauben zudem eine Auflistung aller zulässigen Parameterwerte und eine einfachere Lesbarkeit.

#### Stärken

* Automatische Erstellung einer Struktur mit allen Parameter Informationen
* Kann für alle IC2 Generation Controller verwendet werden

#### Schwächen

* Es braucht eine aktive Verbindung über USB mit dem Controller
* Weitere Applikationen nötig um die zyklischen und azyklischen Parameter der Feldbussysteme auslesen zu können

#### Problemstellung

Durch das das Programm sehr weit ausgereift ist, gibt es prinzipiell kein Problem ausser kleinere Anpassungen, wie z.B. die Feldbus Objekte zu integrieren. Dies wird zum entsprechenden Zeitpunkt realisiert, wenn das Interface fertig entwickelt ist.

### TestUpdateFirmware

#### Ist-Zustand

Die TestUpdateFirmware ruft zuerst das ParameterStructBuild auf, um die Parameterstruktur Files zu erstellen. Anschliessend werden die alten Parameterstruktur Files mit dem neuen ersetzt. Diese Files sind in allen Tests sichtbar, da alle Library Files wie auch die Parameterstruktur Files in einem gemeinsamen Ordner liegen. Der Benutzer kann jetzt angeben, in welchem Testordner alle Tests mit den neuen Parameter Informationen neu kompiliert werden sollen. Wenn alle Tests kompiliert wurden, erscheint ein Fenster mit der Auflistung der Tests, welche nicht ohne Fehler kompiliert wurden. Weiter gibt der Benutzer auch den Ordner an, wo das TTIC2 Programm hinterlegt ist, um auch hier die neusten Parameterstrukturen im Programm zu verwenden.

#### Stärken

* Die Parameter Files werden automatisch ersetzt
* Es werden automatisch alle Tests mit den neusten Parameter Informationen kompiliert
* Auflistung über Tests, welche nicht ohne Fehler kompiliert werden konnten

#### Schwächen

* Die Tests werden auf der Konsole kompiliert -> Arbeit am PC wird eingeschränkt
* Mit der wachsenden Anzahl von Tests braucht der Vorgang seine Zeit

#### Problemstellung

Aktuell werden immer alle Tests mit den entsprechenden Parameter Informationen neu kompiliert. Besitzt das Ventil nicht den gleichen Controller Typ, so kann der Test nicht ausgeführt werden. Der Grund liegt darin, dass die Firmware nur Parameter besitzt, welche durch die Controller Hardware ausgeführt werden. Somit sind die Controller spezifischen Parameter, welche im Test gebraucht werden, bei der Auslesung der Firmware nicht mehr definiert. Aus diesem Grund, müssen diese Parameter im Test ausgeblendet werden, um einen Fehler bei der Kompilation des Tests zu vermeiden.

### TTIC2 (Test Tool)

#### Ist-Zustand

Wird das TTIC2 gestartet, so wird der Benutzer als erstes aufgefordert den Pfad auszuwählen, aus welchem die Test geladen werden sollen. Dabei werden nur Tests in der linken Spalte aufgelistet, welche mit dem aktuellen angeschlossenen Controller ausgeführt werden können (siehe Abbildung 4). Nun können die gewünschten Tests ausgewählt werden und in die rechte Spalte verschoben werden, um die Tests später auszuführen. Unter dem Punkt Selection gibt es verschiedene Filtermöglichkeiten. Es können auch mehrere Filter gesetzt werden. Wird ein Filter gesetzt, so wird der linken Spalte die Auflistung der zu verfügbaren Tests angepasst. Weiter kann unter My Test Collection eine vorher abgespeicherte Test Kollektion geladen werden oder eine neue erstellt werden. Dabei werden alle Tests in der rechten Spalte zu einer Kollektion zusammengefasst. Zusätzlich können spezifische Testeinstellungen wie beispielsweise «soll der Test bei einem Fehler abgebrochen werden», «wie viele Informationen soll der Report liefern» vor der Ausführung definiert werden. Mit dem Drücken des Run Knopfes werden die Tests gestartet.



Abbildung : Ansicht der TTIC2 Oberfläche für die Auswahl der Testkollektion

Anschliessend wird geprüft, ob alle Test mit der angeschlossenen Hardware ausgeführt werden kann. Der Benutzer kann den entscheiden, ob bei einer negativen Überprüfung die Tests trotzdem gestartet werden soll oder die Test Kollektion nicht gestartet werden soll.

Nach der Überprüfung wird ein Reportfenster geöffnet. Dieses zeigt bei der Ausführung der Test Kollektion, den aktuellen Test sowie die bereits ausgeführten und die anstehenden. Eine Ansicht des Reportfensters ist im Anhang zu finden (Testumgebung/TTIC2). Weiter wird jedes Testergebnis notiert. Wird ein Fehler detektiert, so wird dieser rot hervorgehoben. Der Anwender hat die Möglichkeit die Test Kollektion abzubrechen.

Ist die Test Kollektion ausgeführt worden, so kann der User den Report einem gewünschten Ort abspeichern. Wird diese Option nicht gezogen, so wird dieser wiedergelöscht. Für jeden erfolgreichen Test wird ein Zertifizierungsfile erstellt. Bei allen fehlerhaften Tests wird ein Diagnostikfile erstellt.

#### Stärken

* Alle Funktionalitäten sind auf der Oberfläche ersichtlich (keine Verschachtelungen)
* Die verfügbaren Tests können nach Hardware Eigenschaften gefiltert werden
* Wird ein einzelner Test angewählt, so wird eine Beschreibung des Tests sowie die Hardware Anforderungen angezeigt
* Es können Testkollektionen abgespeichert werden
* Der fortlaufende Report wird auf der Oberfläche angezeigt und im Hintergrund in einem Textfile hinterlegt
* Die automatische Generierung von Zertifizierungsfiles

#### Schwächen

* **Das Programm wird auf mehreren Rechnern ausgeführt**
  + Erschwerte Auswertung der Tests
  + Die abgespeicherten Testkollektionen sind nur auf dem jeweiligen Rechner sichtbar
* **Keinen Verlauf der Testergebnisse der verschiedenen Ventil Firmwaren ersichtlich**
  + Letzter Reportfile wird im SVN abgelegt
  + Keine schnelle Suche, ob der Testfehler schon einmal aufgetreten ist
  + Fehlermeldung nur im Reportfile ersichtlich
* Keine Sicherstellung des Grundzustandes

#### Problemstellung

Mit dem Programm TTIC2 entwickelte ich eine Testoberfläche, welche einzelne Tests in einer Kollektion zusammenfasst und nacheinander ausführt.

Das Problem ist, dass aktuell nach der Ausführung der Testkollektion, das entstandene Report File manuell und zeitaufwändig nach fehlerhaften Testdurchläufen durchsucht werden muss. Das Reportfile enthält zudem alle Testschritte und erreicht dadurch eine sehr grosse Datenmenge. Aus diesem Grund wird nur der letzte Report bei einer Ventil Firmware Freigabe in der SVN abgelegt. Dies erschwert die Auswertung der Tests enorm. Zusätzlich werden auch Zertifizierungsfiles abgelegt, diese erhöhen die Datenmenge weiter und kommen deshalb nochmals erschwerend hinzu.

Beim Start des TTIC2 Programms liest die Oberfläche die Testinformationen über ein Textfile aus. Wie schon unter dem Punkt 2.2.3 Test erwähnt, können diese Informationen nicht immer aktuell sein.

### SoftwareVersionsDatabase

#### Ist-Zustand

Ist eine SQL Datenbank, welche aktuell genutzt wird um die Firmware Informationen der IC2 Controller Generation abzuspeichern. Der Grund liegt darin, dass die VAT SQL Server im Einsatz hat.

Dabei werden verschiedene Typen von Firmwaren hinterlegt, wie z.B. Ventil, Motion Controller sowie auch Interface. Dabei wird auch definiert, welche Ventil Firmware mit welchem Motion- und Interface Firmwaren lauffähig ist.

#### Stärken

* Firmware Informationen sind für viele Benutzer ersichtlich
* Firmware Informationen können editiert werden

#### Schwächen

* In der Modellierung wurden die Regeln der Normalform nur teilweise umgesetzt
* Ungeeignete Namensgebung für Zwischentabellen

#### Problemstellung

Die Datenbank enthält aktuell die Firmware Informationen. Die Modellierung beachtet nicht alle Regeln zur Normalform und es werden Funktionen der Modellierung durch die Firmware Database Applikation übernommen. Das Modell soll angepasst sowie die aktuellen Daten migriert werden. Weiter wird eine Erweiterung nötig, um die Testresultate, Testinformationen und Firmware Bugs darin ablegen zu können.

### Firmware Database

#### Ist-Zustand

Die Firmware Database Applikation ist eine in C# realisierte Oberfläche, mit welcher die Firmware Informationen gelesen und bearbeitet werden. Die Oberflächenansicht besteht im Grunde aus zwei Teilen. Im oberen Bereich kann ein einzelner Eintrag bearbeitet werden. Im unteren Bereich werden alle Firmware Einträge aufgelistet.

#### Stärken

* Viele Such- und Filtermöglichkeiten der einzelnen Firmware Einträgen
* Viele Firmware Einträge besitzen vordefinierte Auswahlfelder

#### Schwächen

* Noch keine Benutzerverwaltung (Diese gebraucht, um gewisse Informationen zu verstecken)
* Keine Refresh Möglichkeit (Applikation muss neu gestartet werden)

#### Problemstellung

Die Applikation übernimmt aktuell Funktionen der Datenbankmodellierung. Weiter werden auch die Datenbanktypen in der Applikation verwendet. Werden Firmware Einträge geändert, so sind diese nur in dieser Applikation ersichtlich. Jedoch wird diese Applikation von mehreren Usern genutzt, welche die Applikation meistens geöffnet lassen.

# Zielsetzung

Im ersten Unterkapitel werden die Ursprungsziele der Masterarbeit aufgezeigt. Im zweiten Teil werden die Ziele nach der ersten Analyse revidiert, da bei der Modellierung der SoftwareVersionsDatabase auffiel, dass zusätzlich die Testinformationen abgespeichert werden sollen. Der Grund ist im Kapitel 2.2.2.4 näher beschrieben. Somit fiel der Entscheid die Arbeit auf die gesamte Testumgebung zu erweitern.

## Ziele zum Startzeitpunkt

Abbildung : Ursprüngliches Konzept Masterarbeit

Zielsetzung nach Stand der Disposition besteht darin die bestehende SoftwareVersionsDatabase Datenbank so zu erweitern, dass die Testresultate vom TTIC2 abgespeichert werden. So soll auch der Grundzustand, welcher vor der Ausführung der Testkollektion definiert wird, hinterlegt werden.

Die TTIC2 Oberfläche soll so angepasst werden, dass der Benutzer vor dem Start einer Testkollektion ein Grundzustand definieren muss. Dieser kann nur aus bereits hinterlegten Informationen in der SoftwareVersionsDatabase erstellt werden. Diese Einträge werden in der Firmware Database Oberfläche erstellt.

Das Hauptaugenmerk der Arbeit liegt in der Erstellung des ETIC2. Die Applikation gibt die generierten Testresultate vom TTIC2 wieder. Die Oberfläche ist in drei Ebenen gegliedert. Die höchste Ebene unterteilt die Einträge nach den unterschiedlichen Grundzuständen. Unter dieser Ebene folgt die Testkollektion. Hier wird die Anzahl fehlerhaften Tests angezeigt. In der untersten Ebene werden alle Tests mit ihren allfälligen Fehlermeldungen zu der jeweiligen Testkollektion aufgelistet. Über alle Ebenen ist eine Suche nach bestimmten Suchwörtern möglich.

Durch die Angabe eines Grundzustandes, welche aus einzelnen Elementen besteht, die in der SoftwareVersionsDatabase hinterlegt sind, werden alle Testresultate in einem Report aufgelistet. Speziell dabei ist, dass nicht alle Elemente des Grundzustandes definiert werden müssen. Somit kann z.B. geschaut werden, ob eine Firmware Version mit mehreren Motoren Firmwaren geprüft wurde und somit mehrere Versionen kompatibel ist.

In der ersten Phase der Arbeit dem Erweitern der SoftwareVersionsDatabase fiel auf, dass es Sinn macht nicht nur die Testergebnisse abzuspeichern, sondern auch die Testinformationen. Der Grundgedanke dahinter ist, dass die Testinformationen immer aktuell sind. Als Weiteres wurde auch eine Erweiterung der Datenbank mit den Firmware Bugs Informationen als sinnvoll erachtet. Da die Firmware Bugs aus den Testresultaten ersichtlich werden.

Das Resultat nach der ersten Analyse machte klar, dass das primäre Ziel sein muss eine lauffähige Testumgebung als Resultat der Arbeit liefern zu können. Und nicht eine abgegrenzte Arbeit, in welchem die Testresultate angezeigt werden.

## Ziele nach der ersten Analyse

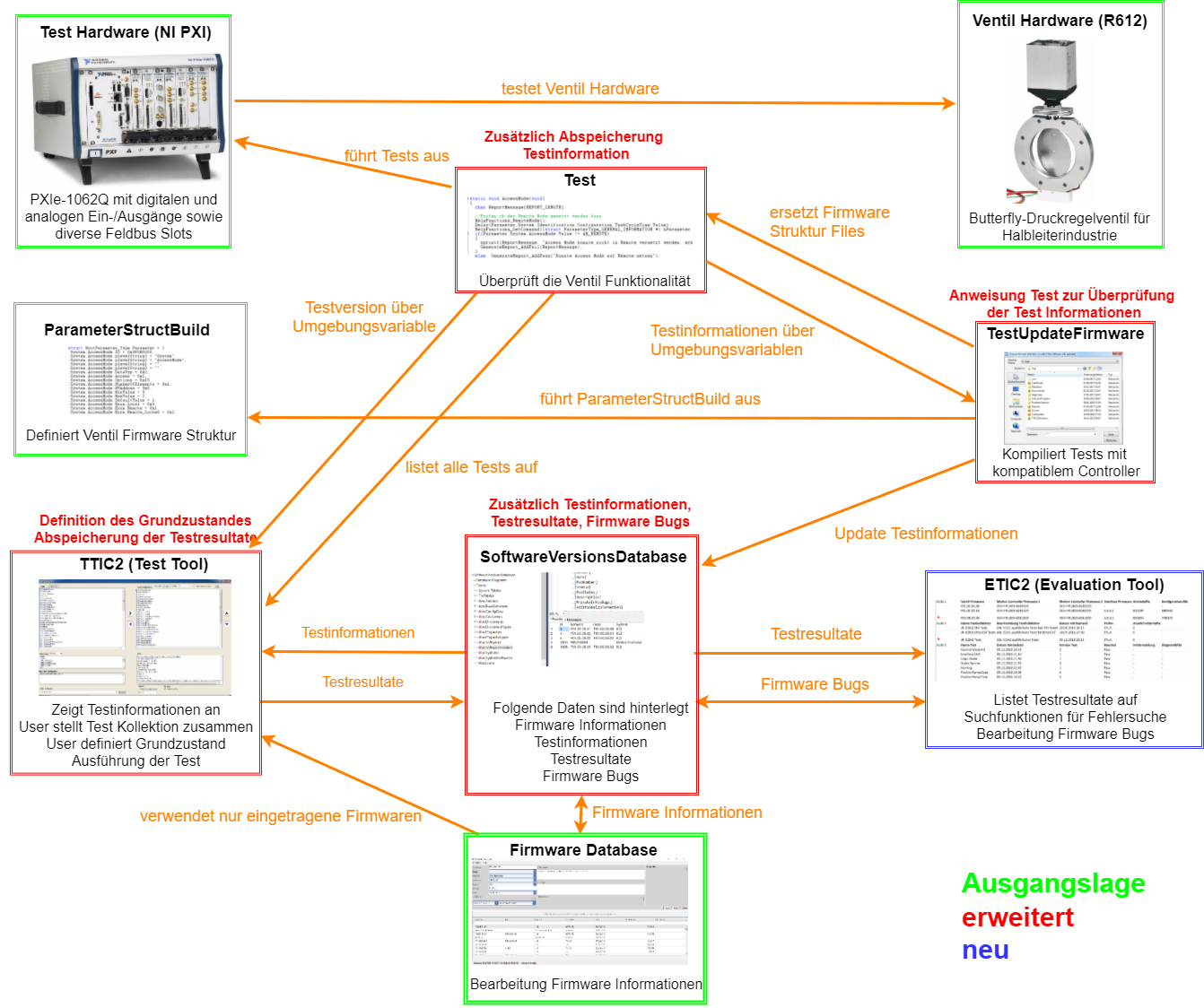
Aus der Abbildung 6 geht hervor, welche Elemente in der Testumgebung gleichbleiben und welche angepasst oder neu erstellt werden. Der nächste Abschnitt erklärt die Ziele der einzelnen Anpassungen.

Abbildung : Konzept Masterarbeit Testumgebung komplett

Die Testinformationen bezüglich des Tests sowie die Anforderungen an die Ventil Hardware soll neu in der SoftwareVersionsDatabase hinterlegt werden.

Die TestUpdateFirmware Applikation soll nur noch die Tests kompilieren, die mit dem Controller der Ventil Hardware lauffähig sind.

Die Testinformationen bezüglich des Tests und der Ventil Hardware soll neu im TTIC2 aus der SoftwareVersionsDatabase ausgelesen werden. Neu soll der Grundzustand vor der Ausführung der Testkollektion definiert werden und zwar nur aus eingetragenen Werten in der SoftwareVersionsDatabase. Sind diese Werte noch nicht auf der Ventil Hardware eingestellt werden diese, bevor die Tests ausgeführt werden, aktualisiert. Nach der Ausführung der Test Kollektion werden die Resultate in der SoftwareVersionsDatabase hinterlegt.

Die bereits existierende SoftwareVersionsDatabase soll so erweitert werden, dass zusätzlich noch Werte bezüglich Testinformationen, Testresultate sowie Firmware Bugs hinterlegt werden können.

Die neue ETIC2 Oberfläche soll eine schnelle Auswertung der Testresultate ermöglichen. Weiter soll ein Report zu einem bestimmten Grundzustand mit allen Testresultaten per Knopfdruck erzeugt werden können.

## Quantitative Ziele

* Die SoftwareVersionDatabase muss gleichzeitig Schreibanfragen von vier Benutzern bearbeiten können.
* Jede einzelne Testkollektion muss im ETIC2 zu einem Grundzustand zugeordnet werden.
* Der Grundzustand kann nur mit bereits vorhandenen Einträgen in der SoftwareVersionsDatabase definiert werden.

## Qualitative Ziele

* Das ETIC2 soll sich durch seinen einfachen und stabilen Aufbau, verbunden mit der raschen Auswertung, ob ein Fehler in der ausgeführten Testkollektion aufgetreten ist, auszeichnen.
* Eine ausgeprägte Suchfunktion soll ein Bestandteil des ETIC2 sein, welche eine schnelle Suche nach Fehlermeldungen erlaubt.
* Mit dem ETIC2 soll das Resultat der ausgeführten Testkollektion unmittelbar und einfach ersichtlich sein.
* Auf Knopfdruck sollen die Testresultate exportiert werden können.
* (optional) Das TTIC2 soll für die Anzeige der Testinformationen immer auf die aktuellsten im Test definierten Beschreibungen zugreifen. Auch die Anforderungen an die Ventil Hardware, welche in den Tests definiert werden, sollen auf dem aktuellsten Stand im TTIC2 sein. (Bei einem Release Prozess)
* (optional) Das TTIC2 muss vor Ausführung der Test Kollektion automatisch die definierten Firmwaren und Parameterfiles (Drive- und Configuration File) auf die Ventil Hardware updaten.

## Aufgabenbegrenzung

* Die Ventil Hardware ist als Verständnishilfe im Konzept zu finden und nicht Teil der Arbeit
* Die Test Hardware wird verwendet um die Ventil Hardware anzusprechen, ist aber nicht Teil dieser Arbeit.
* Durch das ParameterStructBuild Programm können die Parameter Informationen der Firmware ausgelesen werden, was den Anforderungen entspricht.
* Anpassungen an den einzelnen Tests gehören nicht zur Arbeit
  + Ausnahme: Testinformationen in SoftwareVersionsDatabase zu hinterlegen
* Die Weiterentwicklung der TTIC2 Applikation ist nicht Teil der Masterarbeit. Ausnahmen sind:
  + Auslesung der Testinformationen aus der SoftwareVersionsDatabase
  + Definition des Grundzustandes vor der Ausführung Test Kollektion
  + Abspeicherung der Testresultate in der SoftwareVersionsDatabase
* Die SoftwareVersionDatabase wird erweitert aber die bestehenden Attribute und Inhalte werden nicht bearbeitet.
* Die Firmware Database ist bereits im Einsatz um die Firmware Informationen einzusehen, was aktuell ausreicht.
* ETIC2 wird für den internen VAT Verwendungszweck entwickelt und nicht für den kommerziellen Gebrauch konzipiert.

# Methodik

Das Kapitel Methodik umfasst zwei Teile, einerseits die Vorgehensweise in der Arbeit. Diese inkludiert das angewendete Konzept der Software Entwicklung sowie den chronologischen Ablauf der Arbeitsschritte. Anderseits werden die verwendeten Hilfsmittel erläutert.

## Vorgehensweise

In der Software Entwicklung gibt es hauptsächlich zwei Vorgehensweisen nämlich eine agile Methode oder das Wasserfall-Modell. Die agile Variante zeichnet sich durch ihre variable Änderung der Anforderungen aus, wohingegen die Wasserfallmethode durch ihre festen Anforderungen und klaren Phasenabgrenzungen auszeichnet. (Pavlov, 2014)

Mit der Erstellung von automatischen Ventiltests sowie der dazu benötigten Ablaufsteuerung durch das TTIC2 befasse ich mich bereits seit einem Zeitraum von drei Jahren. In der Anfangszeit wurde die Ablaufsteuerung parallel mit immer neuen Tests entwickelt. Im letzten Jahr wird der Hauptfokus auf der Erweiterung der Testabdeckung gelegt. In dieser Zeitspanne wurde das TTIC2 bereits produktiv eingesetzt und die Schwachstellen wurde von meiner Seite evaluiert, da es keine weiteren beteiligten Personen gibt. Daraus resultierten genaue Anforderungen an eine Erweiterung des Systems. Daher kann davon ausgegangen werden, dass sich die Anforderungen während der Arbeit sich nicht verändern. Weiter sind auch die verwendeten Technologien und Hilfsmittel vor Beginn der Arbeit definiert worden (siehe Kapitel 4.2). Die Arbeit wird nur durch eine Person ausgeführt, somit muss das Knowhow nicht ausgetauscht werden.



Abbildung : Wasserfallmodell (QMETHODS, 2017)

Aus diesen Gründen wird die Arbeit prinzipiell mit dem Wasserfall Modell gelöst. Die Abbildung 7 zeigt die einzelnen Phasen des Wasserfallmodells. In der Tabelle 1 wird der Ort der einzelnen Phasen des Wasserfallmodells aufgezeigt.

|  |  |  |  |
| --- | --- | --- | --- |
| **Wasserfall Phase** | **Kapitel** | **Name des Kapitels** | **Inhalt** |
| Anforderungserhebung | 2.2 | Testumgebung | Auflistung Schwachstellen |
| Analyse | 3.2 | Ziele nach der ersten Analyse | Ziele abgeleitet |
| Entwurf | 5 | Auswertung der Testresultate (erstes Unterkapitel) | Grobes Konzept |
| Implementierung | 5 | Auswertung der Testresultate (zweites Unterkapitel) | Beschreibung der Umsetzung |
| Test | 6 | Ergebnisse | Ziele erreicht wurden |

Tabelle : Phasen des Wasserfallmodells in Dokumentation

Die Phase des Betriebs wird erst nach Beendigung dieser Arbeit in Kraft treten. Aus diesen Erfahrungen wird dann später gewisse Wartungsaufgaben auftreten.

Der nächste Abschnitt beschreibt den chronologischen Ablauf der Arbeit.

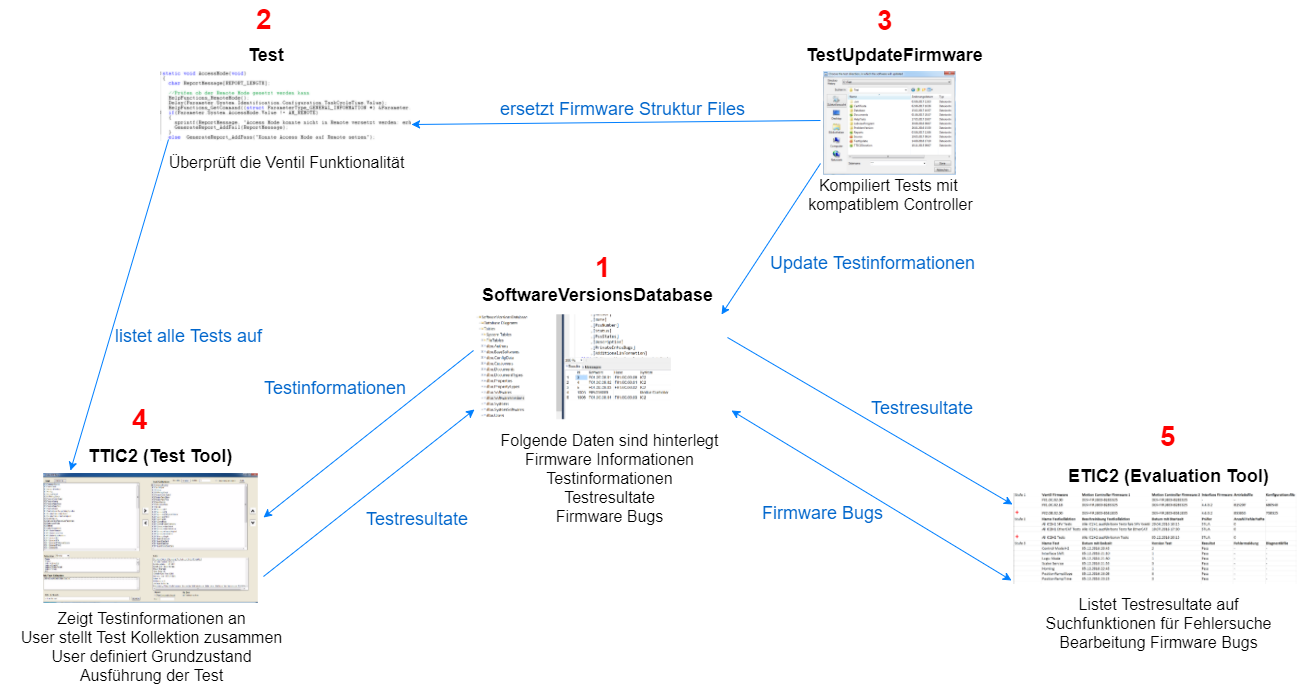


Abbildung : Vorgehensweise Testumgebung

Aus der Abbildung 8 ist die Reihenfolge die einzelnen Testumgebungselemente zu sehen, welche in der Arbeit behandelt werden. Als erstes wird die bereits existierende SoftwareVersionsDatabase erweitert um Testinformationen, Testresultate sowie Firmware Bugs Informationen abzuspeichern. Als nächstes werden die Tests so angepasst, dass die Testinformationen in der Datenbank abgelegt werden. Das TestUpdateFirmware Programm wird so angepasst, das nur noch Tests kompiliert werden, bei welchem der angeschlossene Controller Type übereinstimmt. Anschliessend wird das TTIC2 so erweitert, dass sie erstens die Testinformationen von der Datenbank bezieht und zweitens die Testresultate hinterlegt. Zudem ist ein Ziel das TTIC2 dem Benutzer die Möglichkeit bieten den Grundzustand zu definieren. Als letztes werden die Testresultate im ETIC2 in einfacher Form wiedergegeben.

In dieser Arbeitsphase wird das Wasserfallmodell nicht streng befolgt. Es wird jeweils ein Testumgebungselement genommen und es durchläuft den Entwurf, die Implementierungsphase sowie die Testphase. Erst nach erfolgreichem Testdurchlauf wird mit dem nächsten Element der Testumgebung fortgefahren. Tritt bei einem späteren Testumgebungselement ein Problem auf, so kann das Konzept sowie Implementierung des zuvor entwickelten Testumgebungselements nochmals überarbeitet werden, was nicht der Vorgabe des Wasserfallmodells entspricht.

## Hilfsmittel

In diesem Unterkapitel werden die Hilfsmittel der einzelnen Testumgebungselemente näher beschrieben und ihre Auswahl begründet.

### SoftwareVersionsDatabase

Die Firma VAT bietet einerseits das Microsoft Access Programm an, um Daten oder auch den Zugang zu MS-SQL Servern abzuspeichern. Im Folgenden wurde eine Nutzwertanalyse durchgeführt, um die geeignete Datenbanklösung zu bestimmen (siehe Tabelle 2).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Kriterium | Gewichtung | Datenbanklösung | | | | |
| **Microsoft Access** | | **MS-SQL Server** | | |
| Punkte | Gewichtet | Punkte | | Gewichtet |
| Preis | 5 | 10 | 50 | 5 | 25 | |
| Bereits im Einsatz | 15 | 0 | 0 | 10 | 150 | |
| Bedienoberfläche | 10 | 10 | 100 | 5 | 50 | |
| Fehleranfälligkeit | 10 | 5 | 50 | 8 | 80 | |
| Mehrbenutzerverwaltung | 5 | 3 | 15 | 6 | 30 | |
| Mehrbenutzersynchronisation | 5 | 3 | 15 | 5 | 25 | |
| **Total** | 100 |  | **230** |  | **360** | |

Tabelle : Nutzwertanalyse Datenbanklösung

Der Ausschlag ist auf die Lösung MS-SQL Server gefallen, da bereits eine SoftwareVersionsDatabase im Einsatz ist, welche in einer vorherigen Projektarbeit erstellt wurde. Die MS Access Lösung ist nur für wenige Benutzer geeignet, welche hier im Bereich der Benutzerverwaltung und Synchronisation negativ ins Gewicht fällt, welche für die Firmware Database Applikation aber nötig sind.

Für die Modellierung der Datenbanktabellen wird mit MySQL Workbench gearbeitet, da hier bereits Kenntnisse aus dem Studium vorhanden sind. Das Programm erlaubt es grafisch die Struktur der Datenbank wiederzugeben. Ist diese Struktur vollständig erstellt, so werden Skripts geschrieben, welche im SQL Server Management Studio ausgeführt werden um die einzelnen Tabellen zu erstellen. Weiter bietet dieses Programm die Möglichkeit, die einzelnen Tabellen Werte auszulesen und zu manipulieren.

#### Zugriff auf SoftwareVersionsDatabase

Um Anforderungen bezüglich der Testinformationen in der Datenbank ablegen und um sie später im TTIC2 verwenden zu können. Wie auch um die Testresultate abzuspeichern, welche das TTIC2 produziert, um diese später im ETIC2 anzuzeigen, wird der Zugriff auf die SoftwareVersionsDatabase nötig. In den nächsten beiden Unterkapiteln wird beschrieben, dass Test, TestUpdateFirmware sowie TTIC2 mit CVI im Gegensatz zum ETIC2 in C# auf die Datenbank zugreifen. Im nächsten Schritt wird evaluiert welches Hilfsmittel dafür eingesetzt wird.

Einerseits bietet CVI ein SQL Toolkit an, welches einen schnellen und einfachen Zugang zu SQL Servern bietet. Im Gegensatz bietet die .NET Umgebung mit dem Entity Framework die Möglichkeit den SQL Server abzufragen und erstellt ein passendes Modell der Datenbankstruktur.

**Fall 1**: Testinformationen in SoftwareVersionsDatabase hinterlegen und auslesen

* Lösungsvorschlag 1: Programmierumgebung CVI mit SQL Toolkit
* Lösungsvorschlag 2: C Programm in Visual Studio mit Entity Framework
* Lösungsvorschlag 3: Programmierumgebung CVI mit DLL Programm für Zugang zur Datenbank (DLL in Visual Studio mit Entity Framework)

Auch in diesem Fall wird eine Nutzwertanalyse durchgeführt um eine Entscheidung zu treffen:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Kriterium | Gewichtung | Testinformationen in SoftwareVersionsDatabase | | | | | |
| **Lösungsvors 1** | | **Lösungsvors. 2** | | **Lösungsvors 3** | |
| Punkte | Gewichtet | Punkte | Gewichtet | Punkte | Gewichtet |
| Preis | 10 | 3 | 30 | 8 | 80 | 5 | 50 |
| Unterstützung NI Hardware | 40 | 10 | 400 | 0 | 0 | 10 | 400 |
| Direkter Zugriff | 25 | 8 | 200 | 8 | 200 | 0 | 0 |
| Entwicklungszeit | 10 | 3 | 30 | 6 | 60 | 5 | 50 |
| Wartung | 20 | 8 | 160 | 7 | 140 | 2 | 40 |
| **Total** | 100 |  | **820** |  | **480** |  | **540** |

Tabelle : Nutzwertanalyse Zugriff auf SoftwareVersionsDatabase

Nach Auswertung der Nutzwertanalyse (Tabelle 3) zeigt sich ein anderer Lösungsansatz als der ursprüngliche Lösungsvorschlag 3 in der Disposition. Da dieser keinen direkten Zugriff auf die Datenbank ermöglicht, was in einem höheren Wartungsaufwand resultiert. Die Gewichtung kam zustande, dass aus Sicht des Software Tester der Hauptfokus auf der Testabdeckung der Hardware des Ventils liegt und nicht auf den Anschaffungskosten.

**Fall 2**: Testresultate in SoftwareVersionsDatabase hinterlegen

Gleiche Ausgangslage wie in Fall 1, was in der gleichen Lösung resultiert.

**Fall 3**: Testresultate in SoftwareVersionsDatabase auslesen

Die Testresultate werden in der ETIC2 Oberfläche ausgelesen. Im Kapitel 4.2.3 fällt die Entscheidung für die Erstellung des ETIC2 die C# Programmiersprache zu verwenden. Daraus resultieren zwei Alternativen um die Testresultate auszulesen, einerseits mit Hilfe des Entity Framework, anderseits mit Hilfe einer DLL welche in CVI mit Hilfe des SQL Toolkits erstellt wird.

* Lösungsvorschlag 1: C# mit Entity Framework
* Lösungsvorschlag 2: C# mit DLL Programm für Zugang zur Datenbank (DLL in CVI mit SQL Toolkit)

Nimmt man auch hier die gleichen Beurteilungskriterien wie in der Nutzeranalyse (Tabelle 2) aus Fall 1. Das Kriterium Unterstützung NI Hardware fällt dabei weg, da diese nicht angesprochen werden muss. Der Lösungsvorschlag 1 schliesst in allen Kriterien (Preis, Direkter Zugriff, Entwicklungszeit, Wartung) besser ab und wird somit für die Implementation verwendet.

### Test, TestUpdateFirmware, TTIC2

Wie schon im Kapitel 2.2.2.1 erwähnt, kann die NI Hardware mit LabVIEW oder CVI als Programmierumgebung angesprochen werden. Wie schon in 2.2.3.1 erwähnt werden die Tests in der CVI Programmierumgebung erstellt. Die Gründe liegen einerseits in der persönlichen Programmiererfahrungen, im textbasierten Umfeld sowie der besseren Übersicht in einem grösseren Projekt.

Aus denselben Gründen wurde für das TTIC2 auch CVI als Programmierumgebung ausgewählt. Da das TestUpdateFirmware Programm auch mit den einzelnen Tests zu tun hat, fiel auch hier der Entscheid mit CVI zu entwickeln.

### ETIC2

Bei der Wahl der Programmiersprache für die ETIC2 Oberfläche, welche eine klassische Desktopanwendung darstellt, weil die angezeigten Informationen firmenintern bleiben und somit keinen Webzugang benötigt, fiel die Auswahl auf Java oder C#. Dies nicht nur aus persönlichen Interesse an den Sprachen, sondern auch da diese im Studium näher vorgestellt wurden.

Weitere Gründe für die beiden Programmiersprachen sind (Gebhart, 2013):

* Grosse Auswahl an Hilfsfunktionen (Anteil bei 70% einer Applikation)
  + Reduziert den notwendigen Programmieraufwand auf weniger als 40%
  + Sinkt das Risiko für fehlerhafte Implementierung
  + Robuster, da nicht direkt auf Speicherzellen zugegriffen werden kann
  + Komplette Aufrufhierarchie bei Absturz

Die Entscheidung fiel letztendlich auf C#, was mit folgenden Argumenten begründet wird:

* Firmware Database bereits mit C# implementiert worden
* DevExpress schon erworben (grosse Anzahl an Bedienelementen und Bibliotheken)
* Einfacher zu handhabende Programmierumgebung (Visual Studio)

# Auswertung der Testresultate

Das Hauptkapitel, welches die Umsetzung der Arbeit dokumentiert, teilt sich in Unterkapitel auf, welche die einzelnen betroffenen Testumgebungselemente chronologisch beschreibt. Diese teilt sich weiter in den einzelnen grösseren Arbeitsschritte auf, welche noch in Entwurf und Implementierung unterteilt werden.

## SoftwareVersionsDatabase

### Modellierung Firmware Informationen

In diesem Unterkapitel wird die Modellierung der Firmware Informationen beschrieben, welche in einem früheren Projekt erstellt wurde um Informationen bezüglich einzelner Firmwaren abzuspeichern.

#### Entwurf

Es werden verschiedene Typen von Firmwaren in der gleichen Tabelle abgelegt. In einer weiteren Tabelle wird definiert, welche Firmwaren miteinander kompatibel sind.

Wichtigste Informationen bezüglich einer Firmware, die hinterlegt werden, sind:

* Name
* Basis
* System
* Customer
* Autor
* Erstellungsdatum
* PSS Nummer (interne Produkt Nummer)
* Beschreibung
* Kompatible Firmwaren

In der Datenbank werden die Ventilfirmwaren sowie die Motion Controller Firmwaren wie auch Feldbus Firmwaren hinterlegt. Diese Unterscheidung wird im Feld System erkennbar.

Die PSS Nummer ermöglicht den Zugang zum internen ERP System.

Unter kompatible Firmwaren werden die Motion Controller Firmwaren und Feldbus Firmwaren notiert, welche mit der Ventilsoftware lauffähig sind. D.h. es gibt nur Einträge, wenn es sich beim aktuellen Firmware Eintrag um eine Ventilfirmware handelt.

#### Implementierung

**Aktuelle Version**

In der Abbildung 9 ist das aktuelle SoftwareVersionsDatabase Modell ersichtlich.



Abbildung Aktuelle SoftwareVersionsDatabase Struktur

Im Modell wurde noch die DriveParameterFiles und ConfigurationFiles eingezeichnet, da diese näher an den Informationen der Firmware liegen als bei den Testresultaten. Diese beiden Tabellen werden für den Grundzustand des Ventils vor der Ausführung eines Tests gebraucht.

Was bei genaueren Betrachten der Attribute der einzelnen Tabellen auffällt ist, dass diese mehrfach vorkommen. Wie z.B. beim Eintrag des Customer zu sehen ist. Es gibt einen Eintrag Customer in Customers, welche die einzelnen Customers beinhaltet wie auch in der Haupttabelle SoftwareVersions. Dies wiederspricht der Normalisierungsregel. Die Idee der Customers Tabelle liegt darin, dass nur Einträge dieser Tabelle in der Haupttabelle eingetragen werden können. Um z.B. Schreibfehler, unterschiedliche Reihenfolge des Namens und Vornamen vorzubeugen. Wird ein Fehler bemerkt, werden alle Einträge bei Korrektur mitgeändert in der Haupttabelle. Das Konzept wurde hier von der Applikation übernommen, da hier nur Einträge für den User angezeigt werden, welche in der Customers Tabelle hinterlegt sind.

**Verbesserte Version**

Daher wurde das SoftwareVersionsDatabase Modell überarbeitet, welche in der Abbildung 10 ersichtlich ist.

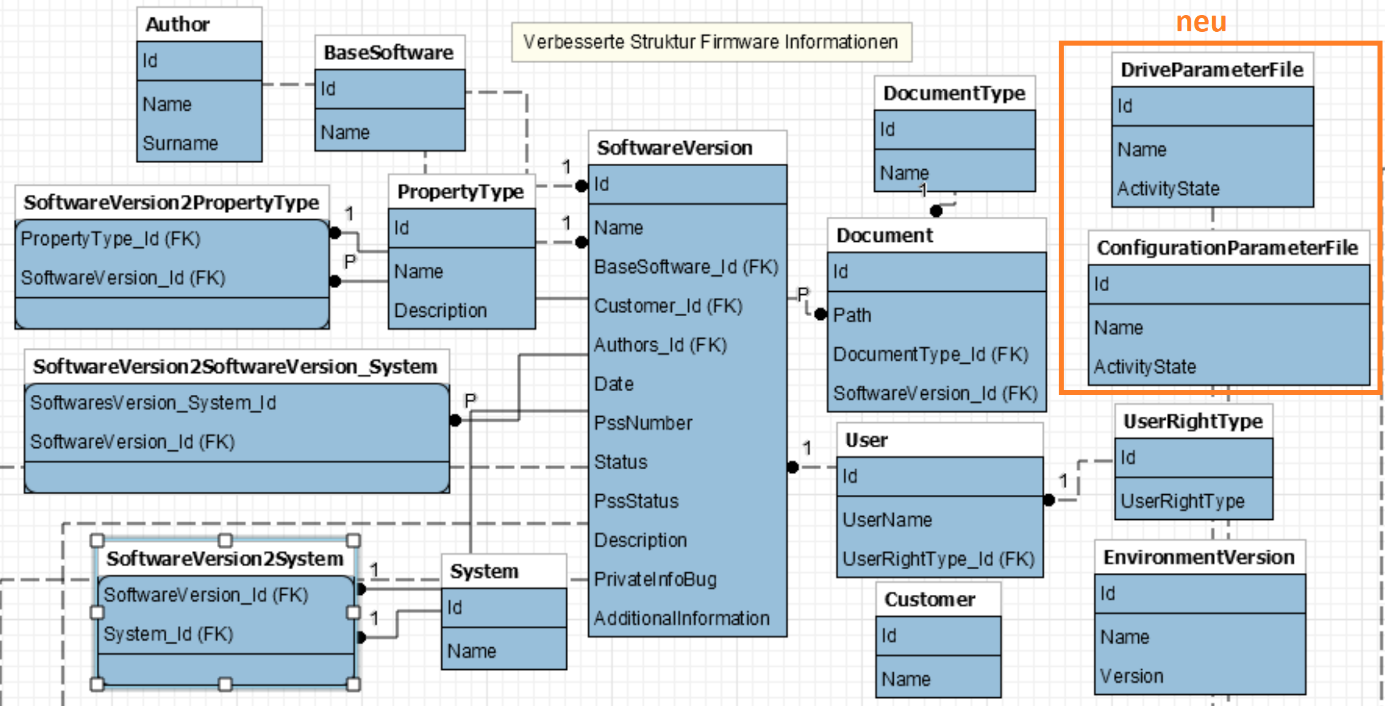


Abbildung : Überarbeitete SoftwareVersionsDatabase Struktur

In diesem Modell stellt die Datenbank sicher, dass der Customer zuerst in der Customers Tabelle eingetragen werden muss und nur seine Referenz in der Haupttabelle eingetragen wird.

Das Modell unterscheidet sich markant in Anbetracht auf die Namensgebung der einzelnen Attribute. Der Tabellennamen kommt jetzt nicht mehr in den einzelnen Attributen zum Tragen. Zudem ist die Authors Tabelle in Bezug auf den Namen aufgesplittet worden was bei einer späteren Auswertung nützlich sein kann.

Um dieses Modell auszuführen, muss die Oberfläche des Firmware Verwaltungstool angepasst werden. Dies kann aus Zeitgründen nicht weiterverfolgt werden. Somit wird in dieser Arbeit noch mit dem ursprünglichen Modell weitergearbeitet. Unter Kapitel 8.1 ist beschrieben, wie später der Umbau auf das neue Modell realisiert werden soll.

### Modellierung Testinformationen

Nachfolgend wird die Erweiterung der SoftwareVersionsDatabase beschrieben um die Testinformationen hinterlegen zu können.

#### Entwurf

Dabei handelt es sich einerseits um Informationen, die den Test näher definieren. Diese sind:

* Name
* Beschreibung
* Erstellungsdatum
* Autor

Anderseits werden Anforderungen des Tests an die Ventil Hardware sowie an die Test Hardware definiert. Die wichtigsten Einträge sind dabei:

* Modul
* Ventilreihe
* Controller
* Interface
* Option
* Test Hardware

#### Implementierung

Die Umsetzung der Modellierung bezüglich Testinformationen ist in Abbildung 11 ersichtlich.

Abbildung : Erweiterung SoftwareVersionsDatabase zur Speicherung der Testinformationen

Die Haupttabelle ist die TestInformation, in welcher es genau einen Eintrag für jeden Test gibt. Hier sind zudem die generellen Daten des Tests hinterlegt. Erweitert wird das Modell mit den einzelnen Versionen eines Tests. So kann später nur durch Angabe der Test Version alle Anforderungen bezüglich der Ventil Hardware aus der Datenbank herausgelesen werden. Denn alle Ventil Anforderungen werden mit der Test Version verheiratet.

Zudem gibt es zu jeder Ventil Hardware Anforderung eine Hilfstabelle, welche die einzelnen Werte definiert. Existiert ein Parameter in der Software mit dieser Information, so wird ein EnumValue Eintrag in der Tabelle eingefügt. Zu diesem Wert wird immer eine Beschreibung dieses Wertes in Textform hinterlegt, dies ist im Feld Enum hinterlegt. Ausser der Modul Information kann es sein, dass mehrere Einträge einer Hardware Anforderung definiert werden sollen. Wie z.B. kann ein Test für mehrere Controller Typen ausführbar sein aber nicht für alle. Das soll mit einem Beispiel verdeutlicht werden:

Der Controller Typ 1 mit Index 1 und Enum ‘IC2H1’ ist sehr ähnlich aufgebaut wie der Controller Typ 3 mit Index 3 und Enum ‘IC2H3’. Mit dem Bitwert 2^1 + 2^3 = 10 kann mit Hilfe eines Wertes mehrere Einträge definiert werden.

Im ersten Version der Modellierung wurde ein Wert für die Definition für den OptionType verwendet. Der Nachteil ist aber das später in der TTIC2 Oberfläche nicht nach einer unterstützten Option gefiltert werden kann, dies wird mit dem SingleOptionType Modelierung gelöst.

### Modellierung Testresultate

Der nächste Abschnitt beschreibt die Erweiterung der SoftwareVersionsDatabase mit den Testresultaten, welche vom ETIC2 gebraucht werden um die Resultate zu den einzelnen Ventilfirmwaren aufzulisten.

#### Entwurf

Der Grund der Zusammenlegung von Firmware Informationen und Testresultate liegt darin, dass nur eingetragene Firmwaren zum Testfall zugelassen sind. Unabhängig ob es sich hierbei um eine Ventil-, Motion Controller- oder Feldbus Firmware handelt.

Um die Spezifikation des ETIC2 zu erfüllen sind folgende zusätzliche Informationen nötig:

* Antriebsfile
* Konfigurationsfile
* Test Kollektion
* Anzahl fehlerhafte Tests
* Ventil Informationen
* Testversion
* Testresultat
* Fehlermeldungen eines Tests

Wichtig ist zudem die Information, welche Tests eine Kollektion beinhaltet. Die Information der Test Version spielt hierbei keine Rolle. Eine Rolle spielt aber die verwendete Ventil Hardware, um die Test Kollektion auszuführen.

Die Test Version ist wichtig für die Information bezüglich Ventilhardware, welche benötigt wird, um den Test ausführen zu können. Der Test wird versioniert um Änderungen bezüglich Hardware berücksichtigen zu können, da das TTIC2 jederzeit auch die älteren Test Collection noch ausführen können muss. Da die meisten Ventilfirmwaren kundenspezifische Entwicklungen sind, wird meistens nicht von der aktuellsten Version weitergearbeitet, sondern eine ältere Version erweitert. Diese Anforderung muss auch für die Tests gelten.

Um eine Auswertung bezüglich ähnlichen Testfehlern machen zu können, sind die ersten Fehlermeldungen in der Datenbank hinterlegt.

#### Implementierung

In der Abbildung 12 ist die Erweiterung der SoftwareVersionsDatabase zu sehen, um die Testresultate abzuspeichern.



Abbildung : Erweiterung SoftwareVersionsDatabase zur Speicherung der Testresulate

Dabei übernimmt die InitialStateFirmware Tabelle die Funktion einen Grundzustand vor dem Start der Test Kollektion herzustellen. Die Modellierung erzwingt, dass die Firmwaren wie auch die DriveParameter- und ConfigurationFile schon in der Datenbank hinterlegt sein müssen, bevor der Grundzustand definiert werden kann.

Aus Software entwicklungstechnischen Gründen wird vor der Freigabe einer Firmware mit einer sogenannten trunk Version gearbeitet. Diese bildet immer zum entsprechenden Zeitpunkt die aktuellste Firmware ab. Um diesem Umstand Rechnung zu tragen und die einzelnen trunk Versionen unterscheiden zu können, wurde ein zusätzliches Attribut ValveFirmwareReleaseTime eingefügt.

In der nächst tieferen Ebene wird die Test Kollektion, die ausgeführt wurde, sowie die Ventil Hardware, auf welcher die Test Kollektion ausgeführt wurde, hinterlegt. Was der Anwender hier interessiert ist die Anzahl fehlerhaften Tests. Weiter ist auch ersichtlich, ob der User den Testablauf abgebrochen hat. Die Test Kollektion besitzt zusätzlich noch einen ActivityState. Dieser wird eingefügt, um zu verhindern das Test Kollektionen, welche schon einen Eintrag im TestCollectionResult haben, gelöscht werden können. So wird zwar die Test Kollektion nicht mehr angezeigt aber ihr ActivityState wird auf False gesetzt.

Die Modellierung lässt bewusst die Möglichkeit offen, ob die erste Ebene den Grundzustand der Firmware abbildet oder die benutzte Ventil Hardware. Wird in der ersten Ebene die Ventil Hardware beschrieben, so kann in der zweiten Ebene die Firmware Informationen aufgelistet werden.

In der untersten Ebene sind alle Tests der Test Kollektion aufgelistet. Hier findet man bei einem Fehler auch die dazugehörigen Fehlermeldungen. Diese ist sehr wichtig, um einen Bug in der Firmware erkennen zu können.

Speziell am Modell ist die CurrentTestsVersion Tabelle. Diese wird bei jedem Start des TTIC2 Programms neu gefüllt. Die Anforderung, dass mehrere Applikationen auf unterschiedlichen Computern gleichzeitig gestartet werden können, führt zum Problem, dass gleichzeitig in die gleiche Tabelle geschrieben werden kann. Um diesen Umstand zu vermeiden wurde für den ersten Ansatz ein zusätzliches Feld mit dem Computernamen eingeführt, um so die einzelnen Applikationen unterscheiden zu können. Doch dieses Konzept scheiterte in der Umsetzung, welches im Kapitel 5.3.1 näher beschrieben wird. Das TTIC2 löst diesen Umstand neu, dass jeder Test mit Hilfe einer Umgebungsvariable seine aktuelle Version liefert. Diese wird beim aufstarten der Applikation in einer Liste hinterlegt. Durch diesen Lösungsweg wird die CuttentTestsVersions und ComputerNames Tabellen nicht mehr gebraucht.

Die aktuelle Testversion wird benötigt, um die entsprechenden Hardware Anforderungen der Tests auslesen zu können und zu entscheiden, ob der Test mit der angeschlossenen Hardware ausführbar ist. Über den Zeitlauf des Tests können sich die Anforderungen verändern, da neue Testfunktionen hinzukommen können. Weiter ist sie auch ein wichtiger Bestandteil für die Auswertung.

In der Testphase wurde bewusst, dass der erste Ansatz mit dem ExecutionDate in der TestCollectionResult zum Poblem führen kann, wenn die gleiche Test Kollektion am gleichen Tag ausgeführt wird. Aus diesem Grund wird jetzt die CreationDateTime verwendet, um auch in diesem Fall das Resultat der Test Kollektion auseinander halten zu können.

### Modellierung Firmware Bugs

Der nächste Abschnitt zeigt die Modellierung der Firmware Bugs auf. Diese ist auch in der SoftwareVersionsDatabase zu finden, da diese aus den Testresultaten resultieren, welche auch im ETIC2 verwaltet werden.

#### Entwurf

Umsetzung wird aus Zeitgründen nicht als realistisch für die Arbeit betrachtet. Was aber umgesetzt wird, sind die nötigen Datenbankfelder. Darunter fallen folgende Kriterien:

* Fehlerart
* Status des Fehlers
* Ventil Hardware
* Fehlerbeschreibung
* Wichtigkeit der Fehlerbehebung
* Datum des Fehlerfundes
* Datum der Fehlerbereinigung

#### Implementierung

Die Umsetzung der Firmware Bugs Modellierung ist aus der Abbildung 13 zu entnehmen.



Abbildung : Erweiterung SoftwareVersionsDatabase zur Speicherung der Firmware Bugs

Über den FailureType Eintrag kann herausgelesen werden, ob der Fehler reproduzierbar ist oder nur sporadisch auftritt. Der StatusType zeigt an, ob der Fehler schon behoben ist, noch offen oder das Fehlverhalten nicht behoben werden kann oder soll. Mit den ControllerType und HardwareIdentificationLevel1 und 2 Felder werden die verwendete Ventil Hardware gekennzeichnet bei welchem der Fehler aufgetreten ist. Neben dem Controller wird in der Hardware Identifikationslevel 1 zwischen dem verwendeten Interface, Option oder der Ventilreihe unterschieden. In der zweiten Ebene wird diese Eigenschaft konkretisiert. Im Falle der Ventilreihe, wird in der zweiten Ebene Einträge wie FaceSeal, SFV oder R655 eingetragen. Dabei wird dies nur eingetragen, wenn der Fehler nur in der definierten Ventilreihe auftritt. Weiter wird das Fehlverhalten genauer beschrieben, wie auch unter Bemerkungen ergänzende Informationen unter welchen Umständen der Fehler ausgelöst wird. Zudem wird kategorisiert, wie schlimm die Folgen des Fehlers in der Praxis sind. Somit kann anschliessend bestimmt werden, welche Fehler zuerst in der Software behoben werden sollen. Dazu wird das Datum des ersten Auftretens des Fehlers eingetragen sowie, wenn der Fehler behoben werden konnte.

## Test

### Abspeicherung Testinformationen

Nach Komplettierung der SoftwareVersionsDatabase folgt als nächster Schritt die Abspeicherung der Testinformationen in der erstellten Datenbank. Diese erfolgt wie schon im Kapitel 4.2.1.1 erwähnt, mit Hilfe des SQL Toolkits, Details zum Funktionsumfang sowie einem Codebeispiel sind im Anhang unter C.2. SQL Toolkitzu entnehmen.

#### Entwurf

In den Testinformationen sind einerseits die spezifischen Informationen zum jeweiligen Test hinterlegt. Diese werden im TTIC2 gebraucht, um dem Anwender den Zweck des Tests zu erklären. Anderseits sind dort die Anforderung an die Ventil Hardware hinterlegt. Die detaillierten Elemente der Testinformationen sind im Kapitel 5.1.1 ersichtlich.

Die Abbildung 14 zeigt unteranderem die Ausgangslage, d.h. wie die Testinformationen früher in einem Textfile abgespeichert, später im TTIC2 wieder ausgelesen und in der Oberfläche angezeigt wurde.

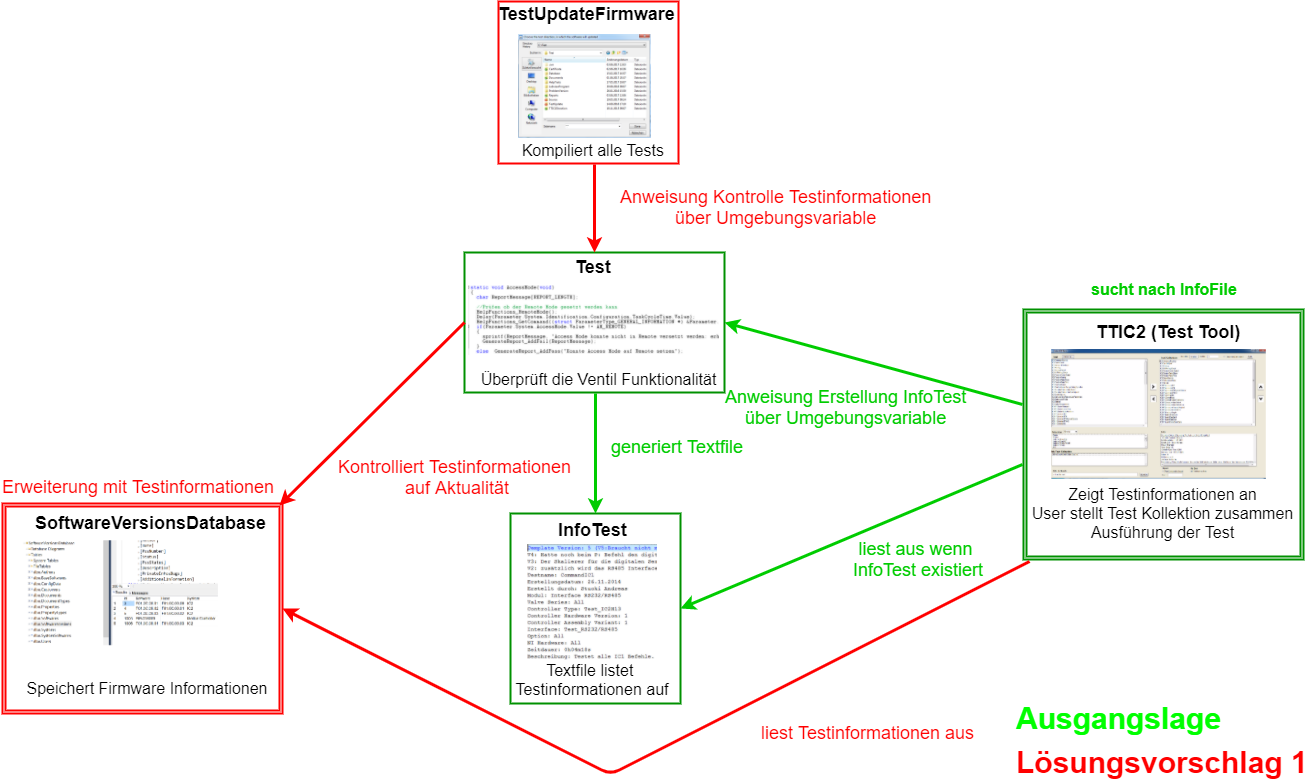


Abbildung : Erster Ansatz Abspeicherung Testinformationen

Der Benutzer wählt beim Aufstarten des TTIC2 den Pfad aus, aus welchem die einzelnen Tests aufgelistet werden. Der Grund der Auswahl liegt darin, dass nicht immer die aktuellste Ventil Firmware getestet werden soll, sondern verschiedene Firmware Entwicklungen angepasst an den Kunden.

Um die Testinformationen auszulesen, sucht das TTIC2 im Testordner nach dem InfoTest Textfile. Existiert dieses nicht, wird der Test aufgerufen und lässt ihn dieses Textfile erzeugen. Diese Funktion wird über eine Umgebungsvariable initiiert, welche vor Ausführung der Testapplikation gesetzt wird.

Der naheliegendste Ansatz für die Abspeicherung der Testinformationen mit Integration der SoftwareVersionsDatabase ist, dass nicht mehr die InfoTest Textfile erzeugt wird, sondern das TTIC2 die Testinformationen aus der Datenbank ausliest. Um die aktuellen Testinformationen in der Datenbank zu haben, ist es unumgänglich, dass diese auf Aktualität überprüft werden. Jetzt stellt sich die Frage, wann ist dies notwendig? Die Antwort fällt nicht schwer, nämlich bei jedem neuen Firmware Release. Das TestUpdateFirmware Programm wird schon verwendet um bei einem neuen Firmware Release die Firmware Struktur upzudaten. Jetzt wird diese zusätzlich noch damit beauftragt die Testinformationen in allen Tests auf Aktualität zu prüfen (siehe Lösungsvorschlag 1 Abbildung 14).

#### Implementierung

Die erste Aufgabe, die Verbindung zur SoftwareVersionsDatabase erwies sich schwieriger als angenommen. Das Beispiel von Seite des SQL Toolkits dies mit Hilfe eines DSN Eintrages, erwies sich nicht als praktikabel. Da dafür Admin Rechte nötig sind, um den Eintrag neu auf einem Rechner erstellen zu können. Die Verbindung mit dem SQL Server kann auch unter Angabe eines Connection Strings aufgebaut werden (Community, 2016).

Im nächsten Schritt werden Funktionen erstellt, um die Testinformationen in die Datenbank zu schreiben, welche im Kapitel 5.1.2.2 in der Modellierung beschrieben sind. Um dem Leser ein besseres Verständnis der Implementierung zu geben, findet sich im Anhang (C.2.2 Code Ausschnitt um einen neuen Eintrag zu erstellen) ein Beispiel um einen Eintrag in der Datenbank mithilfe des SQL Toolkits zu erstellen.

Im nächsten Abschnitt wird das Prinzip erklärt, wie die Testinformationen und die Hardware Anforderungen des Tests in der Datenbank abgespeichert werden. Zuerst werden die Hardware Anforderungstypen in der Datenbank hinterlegt, welche in einem Strukturfile definiert werden.

Für jeden einzelnen Test wird ein Eintrag erstellt, wenn dieser noch nicht eingetragen ist. Diese besitzen unterschiedliche Versionen, welche in der Versionstabelle eine Referenz auf den jeweiligen Test bekommen. Dabei wird die Hardware Anforderungen an den Test mit einer Verbindungstabelle zwischen der Referenz zur Test Version und der Referenz zum Hardware Anforderungstyp erreicht.

Die Grundidee dieser Umsetzung liegt darin die Testinformationen von den einzelnen Firmwaren zu trennen. Durch die Versionierung des Tests können Rückschlüsse auf dem zu benötigten Ventil Hardware bei jedem einzelnen Firmware Release gezogen werden. Dabei ist es nicht zwingend, dass sich die Testinformationen zwischen den Firmwaren verändern.

Als nächstes wurde die Anweisung zur Testinformationsüberprüfung über das TestUpdateFirmware Programm implementiert (siehe Kapitel 5.2.2).

### Testinformationen an TestUpdateFirmware über Umgebungsvariablen

#### Entwurf

Wie im Unterkapitel 5.3.1 erwähnt, wird vermutet, dass das ständige Schliessen und Öffnen der Datenbankverbindung zu unerwarteten Abstürzen führt. Also muss der Lösungsansatz vorsehen, dass die Verbindung nur einmal auf und später abgebaut wird. Um dies zu erreichen, kann der Test selber nicht mehr die Funktion übernehmen, die Daten in der SoftwareVersionsDatabase zu hinterlegen.

Daraus folgt der Lösungsansatz, dass die TestUpdateFirmware die Verbindung zur SoftwareVersionsDatabase herstellt. Die einzelnen Tests liefern über Umgebungsvariablen die Testinformationen. So kann das TestUpdateFirmware Programm die Aktualität der Testinformationen garantieren, siehe Abbildung 15 Lösungsvorschlag 2.

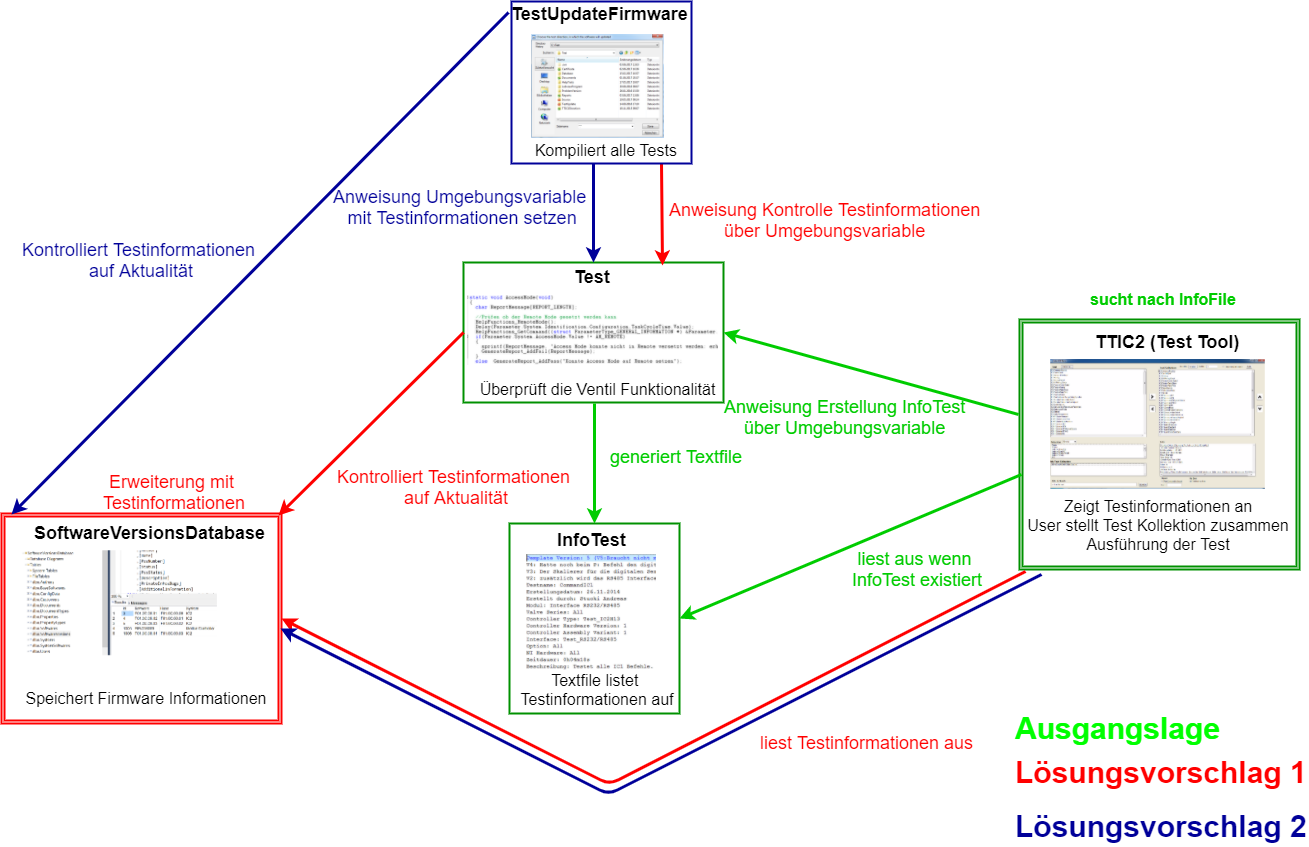


Abbildung : Verbessertes Konzept zur Abspeicherung der Testinformationen

#### Implementierung

Der Test fragt in der main Funktion die TestOptions Variable ab, welche vom TestUpdateFirmware Programm gesetzt wird (siehe Kapitel 5.3.1), um nur die Testinformationen in die Umgebungsvariablen zu schreiben. Es werden mehrere einzelne Umgebungsvariablen angelegt, um einerseits die Testbeschreibung zu hinterlegen, andererseits werden alle unterstützten Hardware Typen des Tests definiert. Dabei wird auf Umgebungsvariablen von CVI zurückgegriffen, da diese ständig überschrieben werden. Das TestUpdateFirmware aktualisiert anschliessend die Testinformationen in der SoftwareVersionsDatabase (siehe Kapitel 5.3.3), welche vom TTIC2 zur Anzeige in der Oberfläche verwendet wird.

## TestUpdateFirmware

### Anweisung Update Testinformationen

#### Entwurf

Wie im vorherigen Kapitel erwähnt und in Abbildung 14 Lösungsvorschlag 1 ersichtlich, startet das TestUpdateFirmware Programm die einzelnen Tests, sodass die Testinformationen in der Datenbank aktualisiert werden können. Wichtig dabei ist es, dass der Test selbst nicht ausgeführt wird, dies wird mit Hilfe einer Umgebungsvariable erreicht.

#### Implementierung

Das TestUpdateFirmware setzt eine Umgebungsvariable, welche später im Test dazu genützt wird um nur die Testinformationen auf Gültigkeit zu überprüfen. Bei der Umgebungsvariable handelt es sich um eine von der C Libary, da sie nicht initialisiert werden muss. Genauere Details sind im Anhang (C.1.1.1. Gegenüberstellung zu C Libary Umgebungsvariable) zu finden.

In der Testphase dieses Ansatzes tritt nach einer zufälligen Anzahl von erfolgreichen ausgeführten Tests ein unerwarteter Absturz auf. Aus dem Absturzprogramm des CVI gehen keine genaueren Informationen hervor. Dies kann auch im Debug Modus nicht reproduziert werden. Meine Vermutung ist, dass beim ständigen Auf- und Abbau der Datenbankverbindung eine Ressource nicht ordnungsgemäss geschlossen werden kann.

Aus diesem Grund wird ein neuer Ansatz gewählt, welche im Kapitel 5.2.2 beschrieben wird, da dieser auch Einfluss auf die Testentwicklung hat. Dadurch wird zuerst wieder die Tests angepasst und anschliessend das TestUpdateFirmware Programm dementsprechend angepasst.

### Abfrage Testinformationen

#### Entwurf

Nach dem gescheiterten Lösungsversuch 1 (siehe Abbildung 15), wird nun nach dem Ausführung des Tests die Umgebungsvariablen für die Testinformationen ausgelesen und zwischen gespeichert. Diese wird für den nächsten Arbeitsschritt benötigt um die Testinformationen auf Aktualität zu überprüfen.

#### Implementierung

Bei den Umgebungsvariablen, welche die Testinformationen des Tests beinhaltet, handelt es sich um die von CVI (siehe Kapitel 5.2.2.2). CVI bietet Callback Funktionen an, welche bei Veränderung der Umgebungsvariable ausgelöst werden. So wird nach Ausführung des Tests, welche die Umgebungsvariable mit den Testinformationen beschreibt, der Wert der Umgebungsvariable ausgelesen und zwischengespeichert. Dieser wird im nächsten Arbeitsschritt mit dem Wert in der SoftwareVersionsDatabase auf Aktualität überprüft.

### Testinformationen in SoftwareVersionsDatabase schreiben

#### Entwurf

Zuerst werden die Hardware Anforderungstypen auf Aktualität geprüft. Anschliessend werden die ausgelesenen Testinformationen Umgebungsvariable Werte auf Aktualität in der SoftwareVersionsDatabase geprüft. Diese Funktionalität wurde bereits im Kapitel 5.2.1.2 beschrieben. Nun wurde sie von den einzelnen Tests in das TestUpdateFirmware Programm übernommen. Deswegen wird auf das Implementationskapitel verzichtet.

## TTIC2

**In** Abbildung 16 **sind die beteiligten Testumgebungselemente, welche für die Erweiterung des TTIC2 gebraucht werden ersichtlich.**

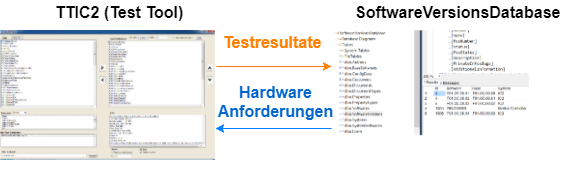


Abbildung : TTIC2 Zugriff auf SoftwareVersionsDatabase

Die SoftwareVersionsDatabase liefert auf der einen Seite die Hardware Anforderungen. Andererseits werden die Testresultate abgespeichert. In den nächsten Unterkapiteln werden die einzelnen Arbeitsschritte genauer erläutert.

### Auslesung der Testinformationen

#### Entwurf

Die wichtigste Frage im Entwurf zur Auslesung der Testinformationen befasst sich damit, wenn die Daten ausgelesen werden sollen. Dabei gibt es mehrere Ansätze. Einerseits können die Daten beim Aufstart des TTIC2 ausgelesen und anschliessend zwischengespeichert werden. Dies hat zum Vorteil, dass anschliessend die Daten immer schnell zur Verfügung sind. Der zweite Ansatz ist, dass die Daten nur bei Anforderung ausgelesen werden. Hier liegt der Vorteil darin, dass die Daten nur ausgelesen werden , wenn sie auch wirklich benötigt werden und somit braucht das Programm weniger lang beim Aufstarten. Da die Applikation nicht zeitkritisch ist werden die Daten nur bei Anforderung ausgelesen.

#### Implementierung

Die erste Frage stellt sich welche Informationen werden minimal beim Aufstarten des TTIC2 benötigt um anschliessend die Testinformationen auf Aufforderung zu bearbeiten. Unabhängig von den Testinformationen muss zuerst eine Liste mit allen verfügbaren Tests erstellt werden. Dies wird erreicht mit dem das im Test Pfad nach exe File durchsucht wird. Der Pfad wird durch den Benutzer beim Auftart des TTIC2 ausgewählt.

Wie schon im Kapitel 5.2.1.2 erwähnt, werden die Hardware Anforderungen mit der Testversion verlinkt. Hat man eine Referenz zu der Testversion kann man nicht nur die verlinkten Hardware Anforderungen auslesen sondern auch die generellen Testversions Informationen. Die generellen Testinformationen können mit Wissen des Testnamen ausgelesen werden.

Fazit: Zusätzlich wird die Referenz zur Testversion gebraucht. Die Testversion ist nur Inhalt des einzelnen Tests. Da die vorherigen Testversionen auch noch unterstützt werden müssen kann nicht die aktuellste Version in der Datenbank zum entsprechenden Test hinterlegt werden. Somit muss unweigerlich der einzelne Test aufgerufen werden und seine Testversion abgefragt werden. Dies wird nachdem der Benutzer den Pfad angegeben hat durchgeführt und jeder Test wird einzeln abgefragt. In einem Ladebalken wird der Fortschritt grafisch dargestellt, siehe Abbildung 17.

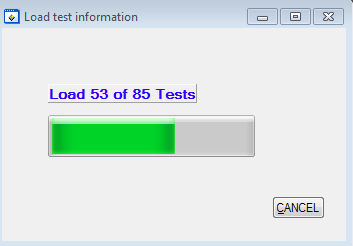


Abbildung : Fortschrittbalken Auslesung Testversionen

### Hinterlegung des Grundzustandes

#### Entwurf

Um zu einem späteren Zeitpunkt ein Fehlerfall wieder reproduzieren zu können ist es unumgänglich die gleiche Ausgangslage vor der Ausführung des Tests herzustellen. Dazu zählen die einzelnen Firmwaren, welche das Ventil benötigt. Zusätzlich können Kundenspezifische Ventileinstellungen in einem File definiert werden. Diese können sich über die Laufzeit verändern und haben zusätzlich einen Index, welcher hochgezählt wird. Zusätzlich kann auch die Reihenfolge der Tests einen Einfluss auf das Testergebnis haben, welche in der Testkollektion definiert wird.

#### Implementierung

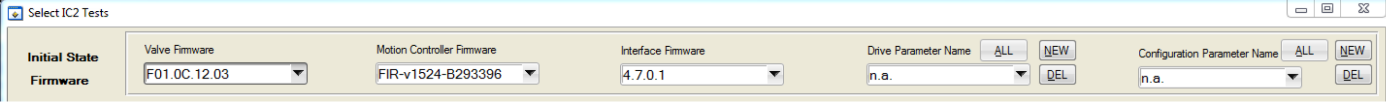
Nachdem der Benutzer den Pfad der Tests ausgewählt hat, kommt die Standardansicht des TTIC2, in welchem die Testkollektion zusammen gestellt wird. Neu in dieser Masterarbeit realisiert, kann im oberen Bereich der Ansicht der Grundzustand vom Benutzer definiert werden, siehe Abbildung 18.

Abbildung : Definition Grundzustand TTIC2

Beim Aufstarten des TTIC2 wird aktuell angeschlossene Ventil Hardware ausgelesen und ihre Werte werden in den Auswahlfeldern gesetzt. Die Auswahlfelder werden mit den Werten, welche sich in der SoftwareVersionsDatabase hinterlegt sind, gefüllt. Ist ein Eintrag nicht in der Datenbank zu finden, so wird der undef Eintrag gesetzt. Bei der Valve Firmware kann eine trunk Version ausgewählt, welche dann anschliessend Manuel ausgewählt werden kann um sie zu laden. Ist kein Feldbus Interface angeschlossen (EtherCAT, DeviceNet) so wird auch hier der undef Eintrag gesetzt. Werden keine kundenspezifische Ventileinstellungen ausgewählt (Drive- und Configuration File) so werden alle Ventileinstellungen auf Standard zurück gesetzt.

Da die Drive- und Configuration Files nicht in der Firmware Database bearbeitet werden können, liefert die TTIC2 Oberfläche die Möglichkeit den Filenamen zu erstellen oder wieder zu entfernen. Ist das Files, welche gelöscht werden möchte schon in einem älteren Testresultat verwendet worden. SO kann dieser Eintrag nicht gelöscht werden, sondern der Activity State wird auf False gesetzt. Diese Files führen einen Index, wird dieser erhöht so wird der Activity State des letzten aktiven Files auf False gesetzt. Mit dem All Button werden alle inaktiven Files aufgelistet.

Ist der Grundzustand und die Testzusammenstellung vom Benutzer ausgewählt worden, so startet der Ablauf der Testkollektion mit dem Drücken des Run Buttons. Jetzt wird geprüft, ob der definierte Grundzustand bereits sich auf der Ventil Hardware sich befindet. Wenn nicht, wird dieser automatisch hergestellt. Ausnahmen sind, wenn bei der Valve Firmware eine trunk Version ausgewählt ist oder bei den anderen Firmwaren der Eintrag undef ausgewählt ist, ausser wenn es sich nicht um ein Feldbus Interface handelt. In diesem Ausnahmen wird der Benutzer aufgefordert die Firmware Manuel auszuwählen.

### Abspeicherung der Testresultate

#### Entwurf

Zur Auswertung ist einerseits die Ausgangslage, der sogenannte Grundzustand, von besonderer Bedeutung um später ein Fehlerfall nachbilden zu können. Weiter interessiert welche Tests das sich in der Zusammenstellung, der sogenannten Test Kollektion sich befinden. Die einzelne Auswertung der Tests darf dabei nicht fehlen mit der dazugehörigen Fehlermeldungen in einem negativen Testfall.

#### Implementierung

Nachdem die Überprüfung des Grundzustand im TTIC2 erfolgreich durchlaufen wurde wird die Hardware geprüft, ob diese den Anforderungen aller Tests entspricht. Wenn dies nicht der Fall ist wird dies dem Benutzer mitgeteilt und er kann sich Entscheiden ob er trotzdem die möglichen Tests ausführen möchte oder den Testablauf abbrechen. Startet der Ablauf der Testkollektion so wird zuerst alle Parameter zurück gesetzt. Sind im Grundzustand Drive- oder Configuration Files definiert, so werden diese Parameter jetzt gesetzt. Anschliessend wird dieser Zustand abgespeichert um diesen nach jedem ausgeführten Test wieder neu zu setzen. Im nächsten Schritt wird der Grundzustand und die Test Kollektion in der SoftwareVersionsDatabase hinterlegt. Weiter wird auch die verwendete Ventil Hardware abgespeichert. Startet ein einzelner Test so wird dieser im ResultTestCollection Tabelle eingetragen. Nach Ablauf des Tests wird sein Resultat und die Endzeit neu gesetzt. Dabei ist aufgefallen, dass die von CVI zur Verfügung gestellte Funktion GetSystemTime für die Stunde nach Mitternacht den Wert 24 liefert. Hingegen wird dieser Wert mit 0 in der Datenbank hinterlegt und auch wieder ausgelesen. Dies musste in der Implementation berücksichtigt werden. Je nach Einstellung der «soll der Test bei einem Fehler abgebrochen werden» wird der Testablauf bei einem Fehler abgebrochen oder fortgeführt. Nach Ablauf der Testkollektion wird der Eintrag mit den Anzahl fehlerhaften Tests gesetzt sowie die Endzeit der Testkollektion neu gesetzt. Weiter wird bei einem Abbruch deren Grund hinterlegt.

## ETIC2

In diesem Kapitel wird erklärt, wie die Testresultate im ETIC2 angezeigt werden, um eine schnelle und einfache Auswertung vornehmen zu können. Weiter soll eine einfache Suche nach Fehlermeldungen inkludiert sein, um eine Aussage zu treffen, ob dieses Fehlerbild schon in der Vergangenheit aufgetreten ist. In der Disposition wurde zudem die Aufgabe die Applikation mit Unit Tests zu verifizieren. Da das ETIC2 nur Daten von der Datenbank über das Entity Framework wiedergibt wird bewusst darauf verzichtet.

### Anbindung SoftwareVersionsDatabase

#### Entwurf

Das ETIC2 braucht Zugriff auf die Testresultate der SoftwareVersionsDatabase um diese später anzeigen zu können. Dabei handelt es sich um eine bereits existierende Datenbank.

#### Implementierung

Das .NET Framework, in welchem auch das Visual Studio dazugehört, bietet gratis das Entity Framework für relationale Datenbanken an. Dieses bietet verschiedene Modellierungsansätze an je nachdem ob die Datenbank schon besteht oder ob der Code oder das Model zuerst kommt. Da die Datenbank schon besteht und die Entity-Klassen manuell zu erstellen sehr arbeitsintensiv ist, wird der Ansatz Model First mit bestehender Datenbank gewählt. Mit Hilfe des Assistenten vom Entity Framework wird nun aus der bestehenden Datenbankstruktur das passende Modell erstellt, diese werden in entsprechende Klassen umgewandelt und im Projekt inkludiert. Im erzeugten Code wird einzig der Datenbankkontext angepasst, so dass ein Verbindungsstring übergeben werden kann.

### Codierung nach MVVM

#### Entwurf

Durch Auswahl der Programmiersprache C# für das Auswertungstool ETIC2 (siehe Kapitel 4.2.3) und den bereits erworbenen Erfahrungen aus der Vergangenheit mit der Firmware Database sowie dem erworbenem DevExpress Bibliothek für WPF Elemente wird das ETIC2 mit dem MVVM Entwurfsmuster gelöst.

In der Entwurfsphase lag der Fokus in der Gestaltung der Ansicht. Das wurde auch schon beim Start der gesamten Arbeit überlegt, da die Modellierung der Datenbank sehr entscheidend ist und auf die Applikation abgestimmt worden ist. Das Ziel lag im einfachen Aufbau und der schnellen Auswertung der Testresultate. Um dies zu erreichen war schnell klar die Daten in unterschiedliche Ebenen einzuteilen, siehe Abbildung 19. Dabei stellte sich die Frage, welche Daten stehen im Vordergrund. Dies konnte einerseits mit der Firmware beantwortet werden, da für einen Testreport einer Firmware die Auflistung aller ausgeführten Tests zu dieser Firmware interessiert. Wie bekanntlich führt das TTIC2 nicht einzelne Tests aus sondern werden zu einer Test Kollektion zusammen gefasst. Da nicht nur die Ventil Firmware entscheidend ist für das Testergebnis sondern auch die Motion Controller sowie die Interface Firmwaren einen Einfluss haben, werden diese zusammen in der gleichen Ebene aufgelistet. Zum Ausgangszustand sind neben den Firmwaren auch die kundenspezifischen Einstellungen sowie die ausgewählte Test Kollektion entscheidend.

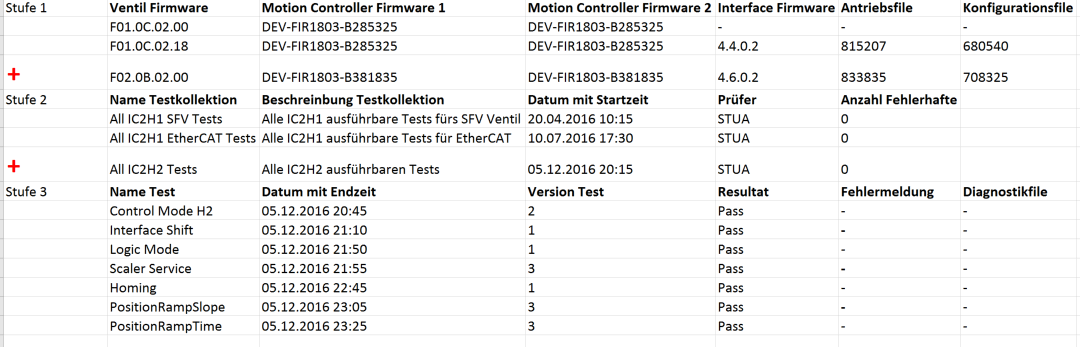


Abbildung : ETIC2 Konzept Ansicht Testresultate

Für die Einführung und Garantie über Einhaltung der Anforderungen an die Ventilmechanik ist es wichtig die einzelnen Tests auf den verschiedenen Ventilreihen auszuführen. Um dies einfach und schnell zu kontrollieren sowie einen Testreport erstellen zu können, soll es auch eine weitere Ansicht geben, in welchem diese Informationen in der obersten Ebene angezeigt wird.

In den unteren Ebenen sollen Informationen zur Anzahl fehlerhaften Tests folgen sowie deren Auflistung der einzelnen Tests. Wird in einem Test ein Fehlerfall festgestellt, so sollen die letzten 3 Fehlermeldungen aufgelistet werden. Diese sollen nach ähnlichen in der Vergangenheit aufgetretenen Fehler durchsucht werden können.

#### Implementierung

In der Umsetzungsphase ist als erstes die Umsetzung der gewünschten Ansicht geprüft worden. Dabei wurde als erstes die DevExpress Bibliothek nach nützlichen Datenansichten durchsuch. Dabei stellte sich schnell heraus, dass das Master-Detail via Entity Framework Element sehr nahe dem Konzept kommt, siehe Abbildung 20.

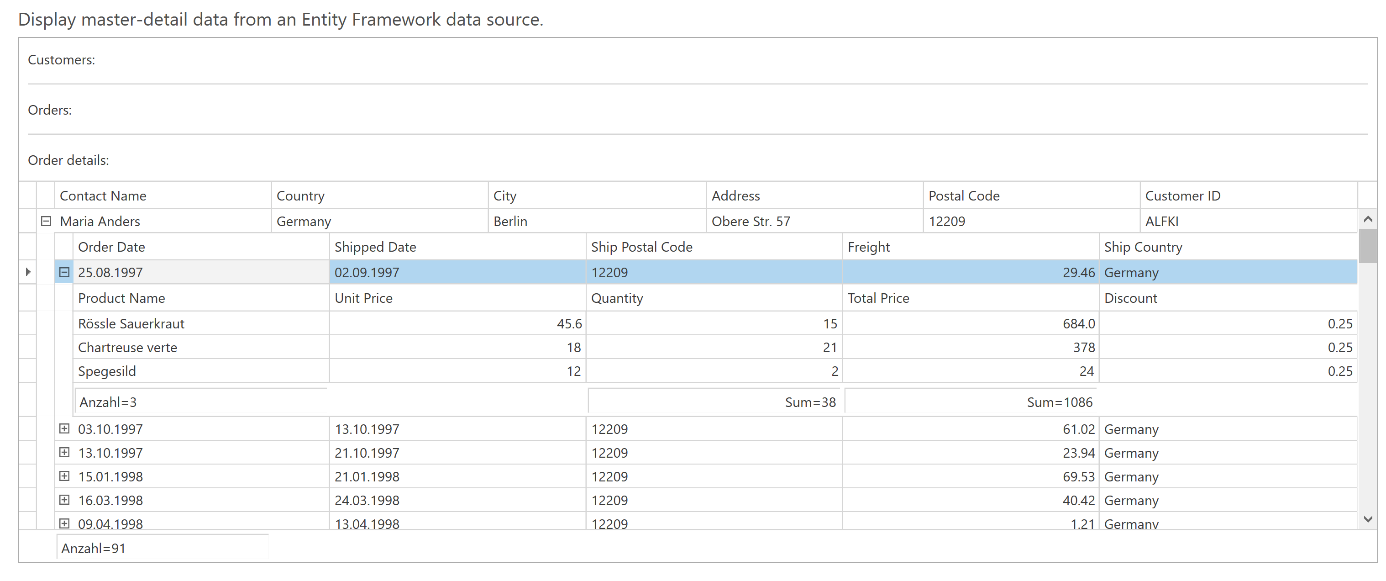


Abbildung : DevExpress Master-Detail via EntityFramework Element

**Model**

Im Model werden die Testresultate von der SoftwareVersionsDatabase ausgelesen. Andererseits soll eine Trennung der Datenbankklassen mit denen von der Applikation erreicht werden.

Der DatabaseAccessManager stellt eine Verbindung durch Übergabe des Verbindungsnamen mit dem vorher erstellten Datenbankkontext vom Entity Framework her, siehe Abbildung 21.

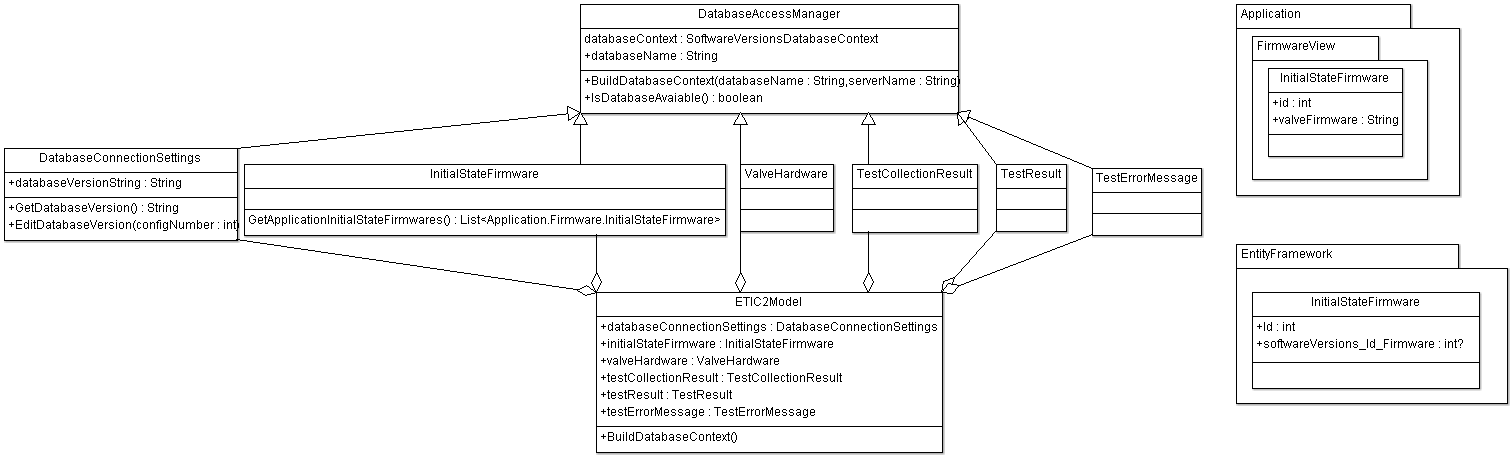


Abbildung : Model ETIC2 in UML

Die DatabaseConnectionSettings Klasse wird erstellt um der Datenbankversion eine Nummer zu geben, welche später für die Überprüfung dient ob die Applikationsversion mit der Datenbankversion überein stimmt. Weiter sind Klassen (InitialStateFirmware, ValveHardware, TestCollectionResult, TestResult, TestErrorMessage) erstellt worden um die Daten aus der Datenbank abzuholen und diese in verwendbare Daten für die Applikation umzuwandeln. Das heisst, dass die Referenzen ersetzt werden mit den realen Datenwerten. Dazu wurden die Klassen für die Applikation in einem eigenen Namespace definiert.

Das ETIC2Model beinhaltet alle Datenbankobjekte, welche vom DatabaseAccessManager ableiten. Diese werden später im ViewModel angeprochen um die Daten auszulesen. Weiter besitzt das ETIC2Model auch eine Funktion um die Datenbankverbindung mit allen Datenbankklassen herzustellen. Da diese in der Applikation geändert werde kann.

**ViewModel**

Um das Master-Detail Element umsetzen zu können müssen die Daten der einzelnen Werte zusammengefasst werden. Weiter müssen die darunter gelegenen Ebene in der oberen in einer Liste eingetragen werden. Das wird im InitialStateFirmwareViewModel Klasse gezeigt, welche die oberste Ebene der Firmware Ansicht darstellt, siehe Abbildung 22.

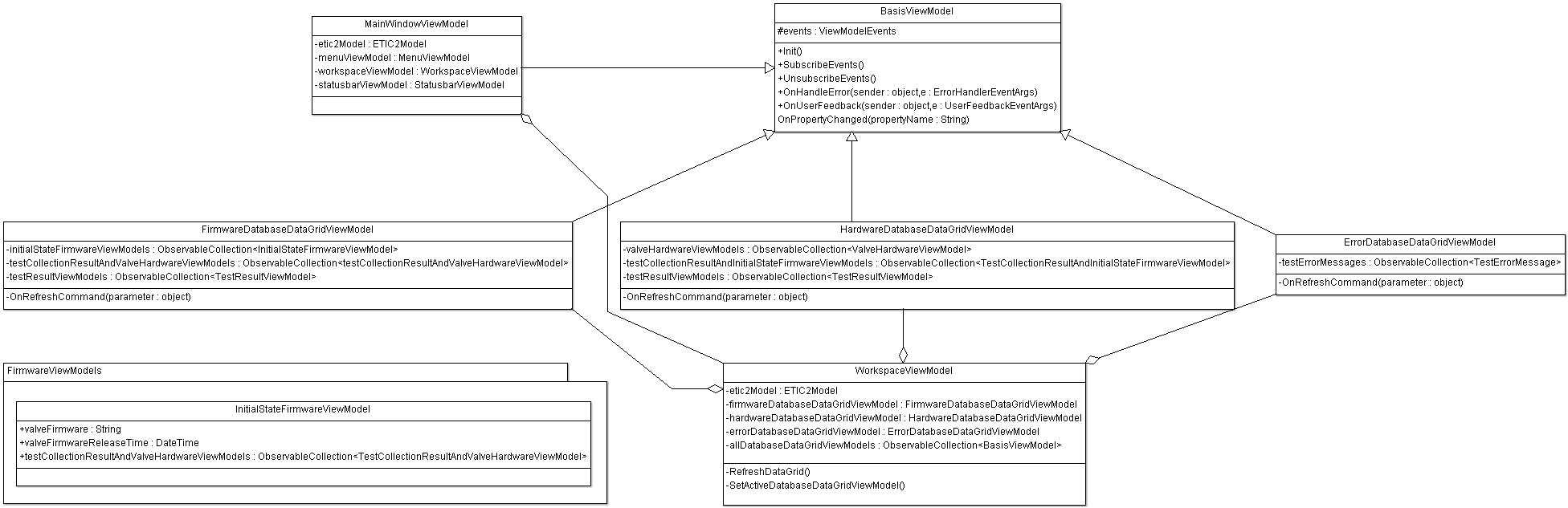


Abbildung ViewModel ETIC2 in UML

In der unteren Ebene werden die Informationen der Test Kollektion sowie der verwendeten Test Hardware aufgelistet. Weiter geht es mit den einzelnen Test Resultaten und den Fehlermeldungen. Die gesamten Daten ist in der FirmwareDatabaseDataGridViewModel hinterlegt. Es gibt noch zwei weitere, einerseits wird die Hardware Auflistung nach dem gleichen System entwickelt. Anderseits gibt es noch eine Ansicht der Fehler, welche nicht in Ebenen aufgeteilt ist. Der Verwendungszweck wird anschliessend im Unterpunkt View erklärt. In Zukunft werden die Ansichten noch erweitert mit derBuglist, um die Fehler im ETIC2 bearbeiten zu können, siehe Kapitel 8.2. Diese einzelnen Ansichten werden im WorkspaceViewModel verwendet, wobei nur eine aktiv sein kann. Das WorkspaceViewModel wird im MainWindowViewModel erzeugt. Welches noch weitere ViewModels für das Menu sowie der StatusBar besitzt.

Alle ViewModel Klassen leiten von der BasisViewModel Klasse ab. Diese Klasse ihrerseits von INotifyPropertyChanged ableitet, welche benötigt wird um Änderungen einer Geschäftsobjekteigenschaft an ein gebundenes Steuerelement mitzuteilen. Weiter besitzt das BasisViewModel virtuelle Methoden (Init, SubscribeEvents, UnsubscribeEvents) um diese in den ViewModels zu überschreiben. Weiter besitzt die Klasse Events um ein Dialog zu öffnen oder den Benutzer über Auffälligkeiten oder Fehler zu informieren.

**View**

Die Ansicht des TTIC2 ist in 4 Module unterteilt, siehe Abbildung 23.

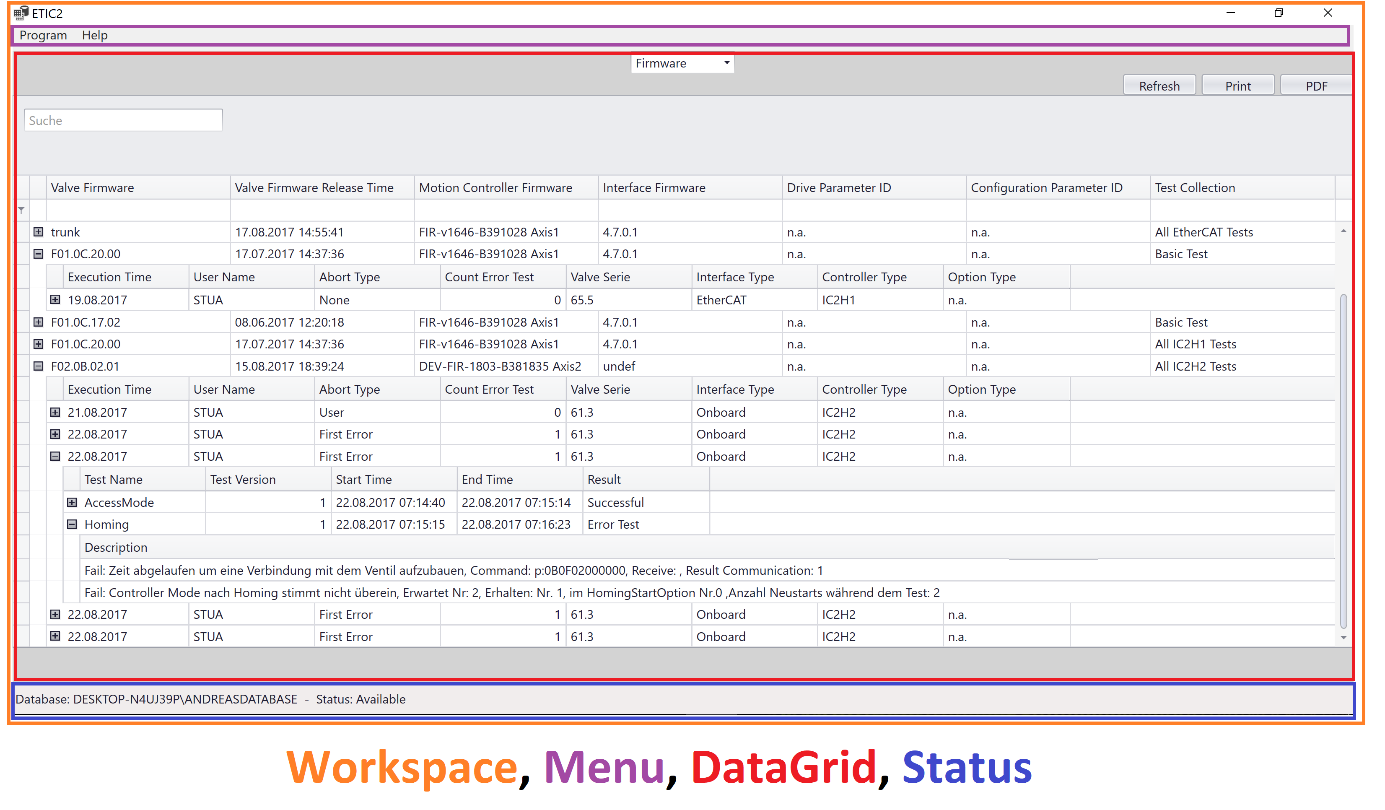


Abbildung : View ETIC2

Das Workspace umfasst die ganze Ansicht. Darin befindet sich im oberen Bereich das Menu. In welchem die Datenbankeinstellungen geändert werden können. Zusätzlich können weitere Informationen über die Applikation abgefragt werden sowie die Applikation beenden.

Unterhalb dem Menu kann der Benutzer mit Hilfe eines Auswahlfeldes bestimmen, welche Ansicht des DataGrids er gerne haben möchte. Dabei kann er zwischen Firmware, Hardware und Error selektionieren. Wählt er z.B. Firmware aus, so wird in der ersten Ebene der Grundzustand der Firmwaren aufgelistet. Die Ansicht umfasst 4 Ebenen bis in der untersten die einzelnen Fehlermeldungen aufgelistet werden. In der Implementierungsphase fiel auf, dass das Master-Detail Element von DevExpress Einschränkungen bezüglich des Suchpanels hat. Nämlich kann nur über die erste Ebene gesucht und gefiltert werden, was für den Anwendungsfall nicht akzeptabel ist. Die Recherche bei DevExpress hat ergeben, dass mit einer kleinen Anpassung auch die zweite Ebene durchsucht werden kann. Was aber bei dieser Alternative als Einschränkung kommt, dass nur noch 2 Ebene mit dem Master-Detail Element möglich ist. Um die Suche auch über 4 Ebenen zu ermöglichen müsste zwangsweise einen eigenen Filter implementiert werden und auch die entsprechenden Ebenen müssten aufgeklappt werden. Dieser Ansatz ist aus Zeitgründen nicht weiter verfolgt worden. Fazit, das Master-Detail bietet nur eine Suche in der ersten Ebene an was schlussendlich zur Lösung führte. So muss zwangsweise in der ersten Ebene die Fehlermeldungen aufgelistet werden. Die Suche nach den Fehlermeldungen wird gebraucht, wenn bei der Auswertung ausfällt das ein Fehler aufgetreten ist. Anschliessend möchte man herausfinden, ob dieser schon einmal aufgetreten ist. Wenn dieser öfters aufgetreten ist, möchte man eine Systematik des Fehlers herausfinden hinsichtlich der verwendeten Hardware oder auch der Software. Dies liefert also neben der Anforderung die Fehlermeldungen in die erste Ebene zu nehmen auch noch, dass zusätzlich Hardware und Firmware Informationen in dieser aufgelistet werden. Dies wurde schlussendlich auch unter dieser Ansicht umgesetzt, in der Tabelle 4 sind nochmals alle Lösungsvorschläge und ihre Einschränkungen kurz aufgelistet.

|  |  |
| --- | --- |
| Lösungsvorschlag | Einschränkung |
| Detail-View über vier Ebenen | Suchfeld nur über erste Ebene |
| Detail-View mit Workaround Ansatz DevExpress (DevExpress, 2016) | Detail-View nur über zwei Ebenen |
| Detail-View über vier Ebenen mit eigenem Filter | Nicht innerhalb geeigneter Zeit implementierbar |
| Detail-View mit einer Ebene | Keine Einteilung der Daten mehr möglich |

Tabelle : Lösungsvorschläge Suche nach Fehlermeldung in ETIC2

Die letzte Ansicht, der Hardware, ist das Konzept mit den 4 Ebenen gleich wie bei der Firmware Ansicht. Der einzige Unterschied liegt darin, dass bei der Hardware Ansicht die erste Ebene von der Hardware eingenommen wird und die Informationen bezüglich der verwendeten Firmwaren wird in der zweite Ebene aufgelistet. Alle drei Ebenen können mit einem Refresh Knopf upgedatet werden. Dabei werden alle Daten neu aus der Datenbank ausgelesen und in die Ansicht eingefügt.

Unterhalb dieser DataGrid Ansicht befindet sich die Datenbank Statuszeile. Diese liefert den aktuellen Datenbanknamen der Verbindung sowie ihren Status.

### Ausgabe Bericht

#### Entwurf

Für die interne Analyse sowie die Auszeichnung gegenüber dem Kunden in Bezug auf die Software Qualität, ist eines der wichtigsten Ziele des ETIC2, auf Knopfdruck einen Report der Testresultate erstellen zu können. Weiter als eine nützliche Funktion wird das Drucken der Testansicht erachtet.

#### Implementierung

Die DevExpress Bibliothek liefert viele Möglichkeiten wie das DataGrid exportiert werden kann. Zu den möglichen Formaten gehören PDF, SLS, XLSX, RTF, CSV. Um die Daten an einem Kunden zu schicken bietet sich das PDF Format an, da dieses Format auf allen Betriebssystemen gleich aussieht. Standardansicht des PDF ist im Hochformat, aber dies ist bei einer breiten Tabelle nicht von Vorteil. Aus diesem Grund werden die Testresultate des ETIC2 im Querformat erstellt. Weiter wurde auch das CSV Format ausprobiert. Da es nur die erste Ebene auflistet, wird es nicht als brauchbar eingestuft.

Auch hinsichtlich der Möglichkeit die Ansicht zu drucken, bietet DevExpress zwei Standardfunktionen an. Einerseits kann eine Druckvorschau erstellt werden oder gleich direkt gedruckt werden. Die Vorschau macht nicht viel Sinn, da auch hier nur Querformat in Frage kommt. Der zu verwendende Drucker kann auch in der direkten Form noch ausgewählt werden, dadurch ist die direkte Druckfunktion implementiert worden. Alle drei unterschiedlichen Ansichten der Testresultate können über die Knöpfe PDF exportiert bzw. über Print gedruckt werden. Diese Funktionen sind im Codebehind programmiert worden, da es ausschliesslich mit der View zu tun hat. Zusätzlich im Codebehind gelöst ist, dass die Einstellungen der Ansichten bei Schliessen der Applikation abgespeichert werden und beim nächsten Aufstarten wieder übernommen werden.

# Ergebnisse

In diesem Kapitel wird beschrieben, ob die unter Kapitel 3.3 und 3.4 definierten Ziele umgesetzt werden konnten.

## Quantitative Ziele

* Die SoftwareVersionDatabase muss gleichzeitig Schreibanfragen von vier Benutzern bearbeiten können.

Dieses Ziel konnte in der Realität noch nicht geprüft werden. Aber es kann davon ausgegangen werden, dass die Implementation dies zulässt den die SoftwareVersionsDatabase läuft auf einem SQL Server 2012. Dieser kann bis zu 32767 Benutzer Verbindungen aufbauen (Jorgensen, 2012). Auf diese Datenbank wird nur aus dem TTIC2 geschrieben oder die Testinformationen werden aus dem TestUpdateFirmware aktualisiert. Das TTIC2 schreibt auf die Testresultate und das TestUpdateFirmware auf die Testinformationen somit kann hier kein Konflikt entstehen. Das TTIC2 erstellt bei jedem neuen Ablauf eine neue TestCollectionResult Eintrag, welcher von den anderen offenen Programmen nicht beschrieben wird. Wann die Daten genau geschrieben werden spielt keine Rolle, da es sich nicht um eine Zeitkritische Anwendung handelt. Das ETIC2 Programm wird nur dazu gebraucht die Testresultate auszulesen, somit kann es nie zur Problematik kommen das veränderte Daten, welche zu einem früheren Zeitpunkt ausgelesen worden sind, in die Datenbank geschrieben werden.

* Jede einzelne Testkollektion muss im ETIC2 zu einem Grundzustand zugeordnet werden.

## Qualitative Ziele

# Diskussion

* Abfrage nach dem Datum hat mit 24h geantwortet, wobei die Datenbank mit 00h Beginnt -> musste angepasst werden

# Ausblick

## Umsetzung Überarbeitung SoftwareVersionsDatabase

Wie unter Kapitel 5.1.1 beschrieben, besitzt das aktuelle SoftwareVersionDatabase Modell noch verbesserungspotential. Die Umsetzung braucht die Anpassung der Software Verwaltungstool Oberfläche. Dies wurde in WPF und nach dem MVVM Pattern Konzept erstellt. Nun die Anpassung darf keinen Einfluss auf das ModelView wie auch der View haben. Daher liegt der Ansatz nahe das Modell auf dem neueren Modell der Datenbank anzupassen. Dazu wird eine Wrapper Klasse erstellt, in welcher die Umwandlung der alten auf die neue Struktur erfolgt.

## Integration Buglist in ETIC2

Die Datenbankfelder für die Verwaltung der Firmware Fehlern wurde bereits in dieser Arbeit erstellt. Weiter soll jetzt im ETIC2 die Option bestehen diese Firmware Fehler anzeigen zu können. Auch in diesem Projekt ist es wichtig eine schnelle Möglichkeit zu haben um nach Fehlern zu suchen. Weiter soll dem User die Möglichkeit gegeben werden, die Firmware Fehler auch bearbeiten zu können.

Die Ansicht besteht aus einem ausgewählten Eintrag, welche alle Informationen im oberen Bereich aufgelistet wird. Der grösste Teil nimmt die Ansicht aller in der Datenbank abgespeicherten Einträge ein.

## Logfile in ETIC2

Aktuell werden alle auftretende Ereignisse dem User über ein Dialog mitgeteilt. Diese ist nur zum aktuellen Zeitpunkt dem Benutzer ersichtlich. Diese sollen zusätzlich in einem Logfile hinterlegt werden um diese später Auswerten zu können und mögliche Bugs in der Implementierung ausmerzen zu können.

# Verzeichnisse

## Literaturverzeichnis

Community. (2016). Abgerufen am 17. 05 2017 von SQL Server 2012 connection strings: https://www.connectionstrings.com/sql-server-2012/

DevExpress. (27. April 2016). *Search panel for detail views work around*. Von https://www.devexpress.com/Support/Center/Question/Details/T372420/search-panel-for-detail-views-work-around abgerufen

Gebhart, G. (2013). *Praktisches & Grundsätzliches zur Informatik*. Von http://greiterweb.de/spw/Wahl-einer-Programmiersprache.htm abgerufen

Marugg, L. (03. 04 2010). PG\_Info\_Hardware. VAT Interne Präsentation. Haag.

National Instruments. (01 2002). Abgerufen am 17. 05 2017 von LabWindows/CVI SQL Toolkit Reference Manual: http://www.ni.com/pdf/manuals/370502a.pdf

National Instruments. (08 2012). Von Network Variables: http://zone.ni.com/reference/en-XX/help/370051V-01/cvi/libref/cvinetworkvariables/ abgerufen

Pavlov, P. (März 2014). *Scrum vs Wasserfall - Ein Vergleich und Erfahrungen aus der Praxis*. Von https://de.slideshare.net/axxessio/scrum-vs-wasserfall abgerufen

QMETHODS. (2017). *Glossar Qualitätssicherung & Testmanagement*. Von https://www.qmethods.de/glossar\_qualitaetssicherung-test.html abgerufen

The Open Group. (2016). Abgerufen am 16. 06 2017 von getenv: http://pubs.opengroup.org/onlinepubs/9699919799/functions/getenv.html

The Open Group. (2016). Abgerufen am 16. 06 2017 von putenv: http://pubs.opengroup.org/onlinepubs/9699919799/functions/putenv.html

VAT Group AG. (2017). Abgerufen am 26. Februar 2017 von http://www.vatvalve.com/de/business/industry

## Abkürzungsverzeichnis

|  |  |
| --- | --- |
| NI | National Instruments |
| WPF | Windows Presentation Foundation |
| DLL | Dynamic Link Libary |

## Abbildungsverzeichnis

[Abbildung 1: Basiskonzept Ventil Controller (Marugg, 2010) 6](#_Toc491588015)

[Abbildung 2: Parameterbaumstruktur der Software 7](file:///C:\Masterarbeit\schriftliche%20Arbeit\Masterarbeit\20170502_ETIC2_Masterarbeit_Stucki_A_v12.docx#_Toc491588016)

[Abbildung 3: Ist-Zustand der Testumgebung 8](#_Toc491588017)

[Abbildung 4: Ansicht der TTIC2 Oberfläche für die Auswahl der Testkollektion 12](#_Toc491588018)

[Abbildung 5: Ursprüngliches Konzept Masterarbeit 15](#_Toc491588019)

[Abbildung 6: Konzept Masterarbeit Testumgebung komplett 16](#_Toc491588020)

[Abbildung 7: Wasserfallmodell (QMETHODS, 2017) 18](#_Toc491588021)

[Abbildung 8: Vorgehensweise Testumgebung 19](#_Toc491588022)

[Abbildung 9 Aktuelle SoftwareVersionsDatabase Struktur 24](#_Toc491588023)

[Abbildung 10: Überarbeitete SoftwareVersionsDatabase Struktur 25](#_Toc491588024)

[Abbildung 11: Erweiterung SoftwareVersionsDatabase zur Speicherung der Testinformationen 26](#_Toc491588025)

[Abbildung 12: Erweiterung SoftwareVersionsDatabase zur Speicherung der Testresulate 27](#_Toc491588026)

[Abbildung 13: Erweiterung SoftwareVersionsDatabase zur Speicherung der Firmware Bugs 29](#_Toc491588027)

[Abbildung 14: Erster Ansatz Abspeicherung Testinformationen 30](#_Toc491588028)

[Abbildung 15: Verbessertes Konzept zur Abspeicherung der Testinformationen 31](#_Toc491588029)

[Abbildung 16: TTIC2 Zugriff auf SoftwareVersionsDatabase 33](#_Toc491588030)

[Abbildung 17: Fortschrittbalken Auslesung Testversionen 34](#_Toc491588031)

[Abbildung 18: Definition Grundzustand TTIC2 34](#_Toc491588032)

[Abbildung 19: Report Ansicht währendem die Tests ausgeführt werden 43](#_Toc491588033)

[Abbildung 20: Funktionsumfang des SQL Toolkits 45](#_Toc491588034)

[Abbildung 21: Code Ausschnitt um einen neuen Eintrag zu erstellen 46](#_Toc491588035)

## Tabellenverzeichnis

[Tabelle 1: Phasen des Wasserfallmodells in Dokumentation 20](#_Toc491509877)

[Tabelle 2: Nutzwertanalyse Datenbanklösung 21](#_Toc491509878)

[Tabelle 3: Nutzwertanalyse Zugriff auf SoftwareVersionsDatabase 22](#_Toc491509879)

[Tabelle 4: Vergleich Umgebungsvariable 45](#_Toc491509880)

## Glossar

|  |  |
| --- | --- |
| Antriebsfile | Enthält alle Ventilhardware spezifischen Abweichungen gegenüber den Standard Einstellungen, welche in der Firmware hinterlegt sind. |
| CVI | Ist eine ereignisorientierte Programmiersprache, welche auf C basiert und von National Instruments entwickelt wurde. . |
| DevExpress | Käuflich erworbene Bibliothek für die Verwendung von WPF Elementen |
| Diagnostik File | Enthält alle Ventilparameter mit ihren aktuellen Werten. Zur genaueren Auswertung eines Fehlers. |
| Enum | Ist ein Aufzählungstyp mit einer endlichen Wertemenge. Die zulässigen Werte werden mit einem eindeutigen Namen definiert. |
| ERP | Enterprise-Resource-Planing: Unternehmerische Software mit deren Hilfe Ressourcen wie Kapital, Personal rechtzeitig und bedarfsgerecht geplant und gesteuert werden kann. |
| ETIC2 | Evaluation Tool Integrierter Controller 2: Auswertungsoberfläche für die Testkollektionen. Integrierter Controller werden Controller genannt, welche direkt mit der Ventil Hardware verbunden sind. |
| Grundzustand | Der Grundzustand setzt sich aus den Angaben der Ventil Firmware, der Motion Controller Firmware sowie optional der Interface Firmware, des Antriebsfiles sowie Konfigurationsfiles zusammen. Jeder Grundzustand erhält einen eindeutigen Namen. |
| IC | Integrierter Controller: Der Controller befindet sich direkt beim Vakuumventil. |
| IC2 | Integrierter Controller der zweiten Generation. Neueste Generation der IC Generation, welche mit den Tests qualifiziert wird. |
| Konfigurations-  file | Enthält alle Abweichungen der Firmware gegenüber den Standard Ventil Firmware Einstellungen, welche in der Firmware hinterlegt sind. |
| MVVM | Mode View ViewModel |
| .NET | Sammelbegriff für mehrere von Microsoft entwickelte Software-Plattformen. |
| SVN | Apache Subversion: Freie Software zur Versionsverwaltung. |
| TTIC2 | Test Tool Integrierter Controller 2: Testoberfläche für alle integrierten Ventilcontroller der 2. Generation |

# A. Projektmanagement

## A.1. Zeitplan

# B. Testumgebung

### B.1. TTIC2

In der unteren Abbildung wird das Reportfenster gezeigt, welche im Kapitel 2.2.6.1 beschrieben wird.



Abbildung : Report Ansicht währendem die Tests ausgeführt werden

# C. Hilfsmittel

## C.1. CVI

### C.1.1. Umgebungsvariable

Im nächsten Abschnitt soll die Umgebungsvariable, welche von NI zur Verfügung gestellt wird, näher vorgestellt werden. Die Umgebungsvariable wird gebraucht um Daten zwischen mehreren Applikationen auf dem gleichen wie auch auf unterschiedlichen Rechnern zu teilen. Wenn sich die Applikationen auf unterschiedlichen Rechnern befindet, werden die Daten über TCP miteinander ausgetauscht. Die Umgebungsvariablen können direkt oder indirekt erzeugt werden. Direkt können die Variablen über das Distributed System Manager Programm erzeugt werden wie auch programmatisch in der CVI Programmierumgebung. Die Variablen können einzeln wie auch mehrere in einem Cluster definiert werden. Es kann definiert werden ob auf die Variable geschrieben oder ob sie nur gelesen werden kann oder beides wie auch deren Datentyp. Es kann auch bestimmt werden, wann die Umgebungsvariable beschrieben werden soll, immer sofort oder erst ab einer bestimmten Datenmenge. Das Distributed System Manager Programm kann bei der indirekten Variante gebraucht werden, um die aktuellen Werte auszulesen oder auch einen Wert zu manipulieren, was beim Debuggen einige Zeit ersparen kann. (Community, 2016)

### C.1.1.1. Gegenüberstellung zu C Libary Umgebungsvariable

Die C Libary bietet die Möglichkeit eine Umgebungsvariable vom Typ String zu benutzen. Unter Angabe des Namens der Umgebungsvariable und dessen Wert wird dieser unter dem System abgelegt (The Open Group, 2016). Dieser kann unter Angabe des Namens der Umgebungsvariable ausgelesen werden. (The Open Group, 2016) Dabei kann die erzeugte Umgebungsvariable nicht mehr überschrieben werden.

|  |  |  |
| --- | --- | --- |
| Umgebungsvariable | | |
| Kriterium | CVI | C Libary |
| Lese Zugriff | immer | immer |
| Schreib Zugriff | überschreibbar | Einmalig im Betrieb |
| Datentyp | gebräuchlichsten | String |
| Cluster zusammenfassen | ja | nein |
| Austausch über unterschiedlichen Rechnern | ja | nein |
| Initialisierung notwendig | ja | Direkt mit Wert versehbar |

Tabelle : Vergleich Umgebungsvariable

## C.2. SQL Toolkit

### C.2.1. Funktionsumfang

Das Ziel des nächsten Abschnittes ist es den Funktionsumfang des SQL Toolkits vorzustellen. Die Abbildung 21 zeigt dabei die einzelnen Funktionen auf.



Abbildung : Funktionsumfang des SQL Toolkits

Als erster Schritt wird über die DBConnect Funktion eine Verbindung zur Datenbank aufgebaut. Diese wird anschliessend gebraucht, um entweder über Automatic SQL, Explicit SQL oder Immediate SQL Anweisungen auszuführen um Daten auszulesen, zu verändern oder zu erstellen, welche in der Datenbank hinterlegt sind.

**Auflistung Vor- und Nachteile:**

Die Automatic SQL Variante ermöglicht einfache Select Abfragen wie auch Create Table Befehle. Dabei wird die Anweisung von der SQL Toolkit übernommen.

Bei der Explicit SQL Variante muss die SQL Anweisung vom Programmierer geschrieben werden. Aus diesem Grund können komplexere Select wie auch andere Arten von Anweisungen ausgeführt werden.,

Mit der Immediate SQL Möglichkeit entfallen die DBActiveSQL und DBDeactiveSQL Funktionsaufrufe. Diese wird eingesetzt, wenn nach der Anweisung, keine Daten mehr vorhanden sein müssen.

Bei den ersten beiden Varianten kann die Tabelleninformation in unterschiedliche Datentypen ausgelesen werden und liegt dem Programm gleich zur Weiterverarbeitung bereit. Es ist nötig vor der Ausführung einer SQL Anweisung zuerst System Ressourcen anzulegen, welche nach der Ausführung wieder freigegeben werden. Hierbei handelt es sich um temporäre Files. (National Instruments, 2002)

### C.2.2 Code Ausschnitt um einen neuen Eintrag zu erstellen



Abbildung : Code Ausschnitt um einen neuen Eintrag zu erstellen

Im Code-Ausschnitt (Abbildung 19) wurde auf die Fehlerüberprüfung verzichtet, um eine bessere Übersicht dem Leser zu geben. Der Verbindungsaufbau sowie Abbau zur Datenbank wird nur als Funktionsaufruf dargestellt. Die Aufgabe der Funktion Database\_WriteControllerTypeTestInDatabase besteht darin zu prüfen, ob der Controller Type schon mit der Test Version verlinkt wurde. Die Verlinkungstabelle ist nötig, um mehrere Controller Typen einer Test Version zuweisen zu können. Die unterschiedlichen Controller Typen sind in einer eigenen Tabelle (ControllerTypes) definiert. Bei der Verlinkung wird dabei eine Referenz des Controller Typs angegeben.

Nach Aufbau der Verbindung wird mit dem Controller Type Eingangsstruktur, der Controller Type definiert, wenn dieser noch nicht existiert. Die Funktion liefert die ID des Controller Types zurück. Nun wird in der Verlinkungstabelle geprüft, ob die Verbindung schon vorhanden ist. Dazu wird die DBActivateSQL Funktion verwendet, um einen Select Befehl auszuführen. Mit den DBBindCol Funktionen können Tabellenwerte im entsprechenden Datenformat direkt ausgelesen werden. Mit der while Befehl DBFetchNext werden die einzelnen Tupel der ControllerType Tabelle ausgelesen. Mit der if Anweisung wird kontrolliert, ob die Verlinkung schon eingetragen ist, wenn ja wird die while Bedingung abgebrochen und die Funktion verlassen. Ist hingegen kein Eintrag gefunden worden, werden die Expliziten Ressourcen wieder geschlossen und ein neues Mapping mit DBBeginMap Funktion erstellt. Analog mit DBMapColumnTo können Attribute der Tabelle in der Automatic Variante direkt gebunden werden. Mit der DBActiveMap Funktion wird die zu editierende Tabelle angegeben. Die DBCreateRecord erzeugt einen neuen Eintrag, in welchem anschliessend die gebundenen Werte gesetzt werden, bevor mit dem DBPutRecord Anweisung das Tupel in der Datenbank abgespeichert wird. Am Ende werden mit dem DBDeactivateMap Anweisung die erzeugten Automatic Ressourcen geschlossen.

# Selbständigkeitserklärung

Mit der Abgabe dieser Abschlussarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat (Bei Teamarbeiten gelten die Leistungen der übrigen Teammitglieder nicht als fremde Hilfe):

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Abschlussarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremdem Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Ort, Datum: ………………… Unterschrift Studierende/r: ………............