HTL Perg für Informatik
Marchlandstraße 48
4320 Perg
Österreich

Diplomarbeit

DomainCopy

Integrity – DomainCopy

Integrity Applikation zum Kopieren von Items zwischen Domänen

Eingereicht von: Hannes Windischhofer, Christoph Seiler – 5AHIF 2015/2016

Betreuungslehrer: Dipl.-Ing. Christian Aberger

Auftraggeber: Magna Powertrain

Betreuer: Dipl.-Ing. Peter Zsifkovits, Dipl.-Ing., Dr. Daniel Schleicher

Eidesstattliche Erklärung

Perg, am 08.04.2016

Hiermit erklären wir eidesstattlich, die hier vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst und ausschließlich die angegebenen Quellen beziehungsweise Hilfsmittel benutzt zu haben und wörtliche oder inhaltliche Stellen aus jenen Quellen auch als solche erkenntlich gemacht haben.

Windischhofer Hannes

Hannes Windischhofer

Seiler Christoph

Christoph Seiler

Danksagung

An dieser Stelle möchten wir uns bei all jenen Personen bedanken, die uns bei der Er- und Fertigstellung dieser Diplomarbeit mit Rat und Tat zu Seite standen und uns unterstützten.

Primär gilt unser Dank unseren beiden Betreuern Herrn Dipl.-Ing. Dr. Daniel Schleicher und Herrn Dipl.-Ing. Peter Zsifkovits, der Administrator des Tools *PTC Integrity*, sowie unserem Freund Berislav Klepic, der uns die Diplomarbeit überhaupt erst ermöglicht und Hilfestellungen im Bereich der *Integrity-API* geleistet hat.

Alle drei haben uns während der gesamten Dauer der Diplomarbeit unterstützt und viele Ideen und Lösungsvorschläge für unsere Arbeit beigesteuert.

Weiters möchten wir uns auch bei Herrn Professor Dipl.-Ing. Christian Aberger für seine nützlichen Ratschläge sowie seine tatkräftige Unterstützung im Zuge der Diplomarbeit bedanken.

Impressum

Schule

- HTBLA Perg für Informatik
- Marchlandstraße 48
- 4320 Perg

Schuljahr

- 2015/2016

Klasse

- 5AHIF

Projektname

- Integrity-DomainCopy (Integrity Applikation zum Kopieren von Items zwischen Domänen)

Projektteam

- Hannes Windischhofer
- Christoph Seiler

Betreuungslehrer

- Dipl.-Ing. Christian Aberger

Inhaltsverzeichnis

1. Kurzbeschreibung

Um Ihnen einen Einblick in die vorliegende Diplomarbeit zu verschaffen, enthalten die nächsten Zeilen eine kurze Zusammenfassung derselben.

Aufgabenstellung

Das Kopieren von Items (Textabschnitt, Bild) ist in dem Programm *Integrity* (ein Application Lifecycle Management System) nur beschränkt möglich, da Items nur innerhalb einer Domäne kopiert und später wieder eingefügt werden können. Mithilfe des Programmes *Integrity-DomainCopy* soll es möglich sein, Items einer Domäne in eine andere zu kopieren.

Realisierung

Damit der Kopiervorgang erfolgreich durchgeführt werden kann, müssen die einzelnen Felder der Items der verschiedenen Domänen einander zugeordnet werden, da sich diese von Domäne zu Domäne unterscheiden. Sichergestellt wird dies dadurch, dass für den Kopiervorgang verschiedene Konfigurationen zu Verfügung gestellt werden, die auch selbst erstellt werden können. Eine Konfiguration besteht aus Name, einer kurzen Beschreibung, einer Information, ob zwischen dem alten, kopierten Item und dem neuen, eingefügten Item ein Trace (eine Verbindung) erstellt werden soll, sowie der Quell- und Zieldomäne. Darüber hinaus enthalten diese Konfigurationen des Kopiervorganges auch das sogenannte Field-Mapping. Das Mapping wiederum stellt sicher, dass der Inhalt der kopierten Items dort eingefügt wird, wo es der Benutzer will.

Das Programm, geschrieben in Java, inklusive der GUI ist eine Integration in *Integrity*.

Ergebnis

Nach Abschluss des Projektes wurde das System und entsprechende Testcases zur Wartung an die Firma Magna Powertrain geliefert. Aufgrund der engen Zusammenarbeit mit dem Auftraggeber während der Entwicklungszeit war keine besondere Einschulung von Seiten des Entwicklerteams notwendig.

2. Abstract

In order to provide a brief insight into our diploma thesis, the following lines contain a short summary of our project, including the functionality and goals of our work.

Scope

The copying of items (e.g. text, pictures) in “Integrity” (an Application Lifecycle Management System) is limited due to the restriction of items only being able to be copied within one domain. With the aid of *Integrity-DomainCopy*, items of one domain can be copied to another.

Implementation

To make sure the copy-process finishes successfully, the item fields of the affected domains have to be allocated to one another because of the fact that they don’t have the same properties. To ensure this allocation, the user can create multiple configurations to change the way the copy-process works. A configuration consists of a name, a short description, an information if a trace (a connection) shall be drawn between the afflicted items plus the source and target domain. Moreover, these configurations contains a so-called field mapping. This mapping ensures that the content of one item is pasted there, where the user wants it to be.

This program, written in Java, including the GUI is an integration in *Integrity*.

Result

After the completion of the project, the system and the respective test-cases for maintenance were given to Magna Powertrain. Due to the close co-operation during the implementation with the client, a special introduction to the system from the developers was not necessary.

3. Stakeholder im Projekt

3.1. Das Team

3.1.1. Hannes Windischhofer

Persönliche Daten	
Geburtsdaten:	Linz, 04.Mai 1997
Staatsbürgerschaft:	Österreich
Religionsbekenntnis:	röm.-kath.
Familienstand:	ledig
Eltern:	Manfred Windischhofer, Beamter Erika Windischhofer, Ärztliche Assistentin
Geschwister:	Philipp Windischhofer, Student

Abbildung 1:
Hannes Windischhofer

3.1.2. Christoph Seiler

Persönliche Daten	
Geburtsdaten:	Amstetten, 10. Jänner 1997
Staatsbürgerschaft:	Österreich
Religionsbekenntnis:	röm.-kath.
Familienstand:	ledig
Eltern:	Ronald Marton, Bankangestellter Sabine Seiler-Stubauer, Bankangestellte
Geschwister:	Pia Stubauer, Schülerin Paul Stubauer, Baby

Abbildung 2:
Christoph Seiler

Für eine genauere Beschreibung des Teams siehe Anhang unter Punkt 14.1.

3.2. Betreuungslehrer

Betreut wird die Diplomarbeit von Professor Dipl.-Ing. Christian Aberger, welcher zugleich seit der vierten Klasse unser vielgeschätzter Java-Lehrer ist. Durch seine hilfsbereite und sympathische Art konnte er uns immer wieder dazu antreiben, das Bestmögliche zu geben und uns, darüber hinaus, das eine oder andere Mal unter unsere Arme greifen.

Persönliche Daten	
Name:	Dipl.-Ing. Christian Aberger
Adresse:	Softwarepark 37
Ort:	4232 Hagenberg
E-Mail:	christian@aberger.at

Abbildung 3: Dipl.-Ing. Christian Aberger

Zusätzlich zu seiner Arbeit als Professor an der HTL Perg ist er auch noch Geschäftsführer der Aberger Software GmbH.

3.3. Auftraggeber

MAGNA POWERTRAIN

Abbildung 4: Magna Powertrain Logo

Abbildung 5: ECS St.Valentin Bürogebäude

Wie bereits erwähnt ist der Auftraggeber der Diplomarbeit das Unternehmen *Magna Powertrain* in St. Valentin in Niederösterreich. Die Ansprechpartner sind Herr Dipl.-Ing. Dr. Daniel Schleicher und Dipl.-Ing. Peter Zsifkovits.

Die *Magna Powertrain* ist eine Firmengruppe der *Magna Holding*, einem kanadisch-österreichischem Unternehmen mit Hauptsitz in Aurora, Kanada. Die Verwaltung des europäischen Zweiges liegt in Wien.

Kontakt	
Name:	Manga Powertrain
E-Mail:	info.ecs@magna.com
Adresse:	Steyrer Straße 32
Ort:	St. Valentin
Tel:	+43 7435 501 0
Web:	www.ecs.steyr.com

Die Geschichte des Unternehmens Magna Powertrain begann 1864, als die Firma Steyr-Daimler-Puch von Josef und Franz Werndl gegründet wurde. Damals diente sie hauptsächlich als Waffenfabrik und Sägemühle, bekannt wurde das Unternehmen 1924 unter dem Namen Steyr Werke AG.

Begonnen wurde mit der Produktion von Fahrrädern, Mopeds und Motorräder, sowie von Autos, Traktoren, LKW und Bussen.

Die drei großen Namen Steyr, Daimler und Puch dominierten in den frühen Jahren den Automobilsektor. Hier liegen auch die Wurzeln von MAGNA.

Heute ist die MAGNA als einer der führenden Zulieferer der Automobilindustrie weltweit bekannt.

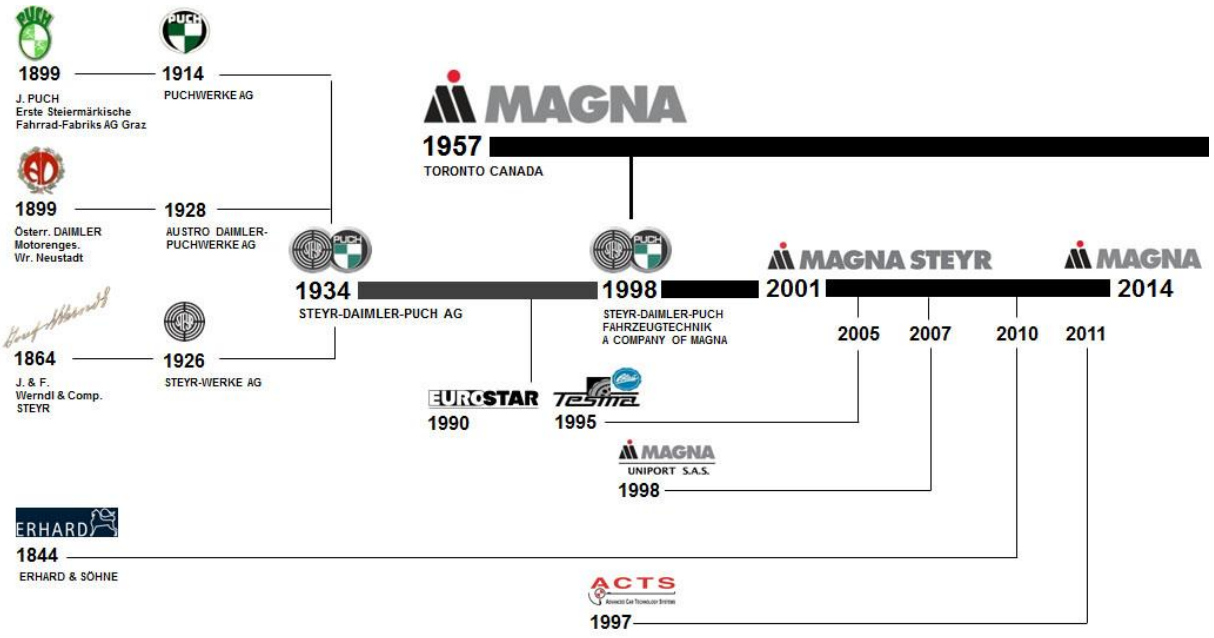


Abbildung 6: Geschichte von Magna

4. Beschreibung des Umfelds

4.1. PTC Integrity

Die folgende Abbildung zeigt die Architektur des Programms *PTC Integrity*.

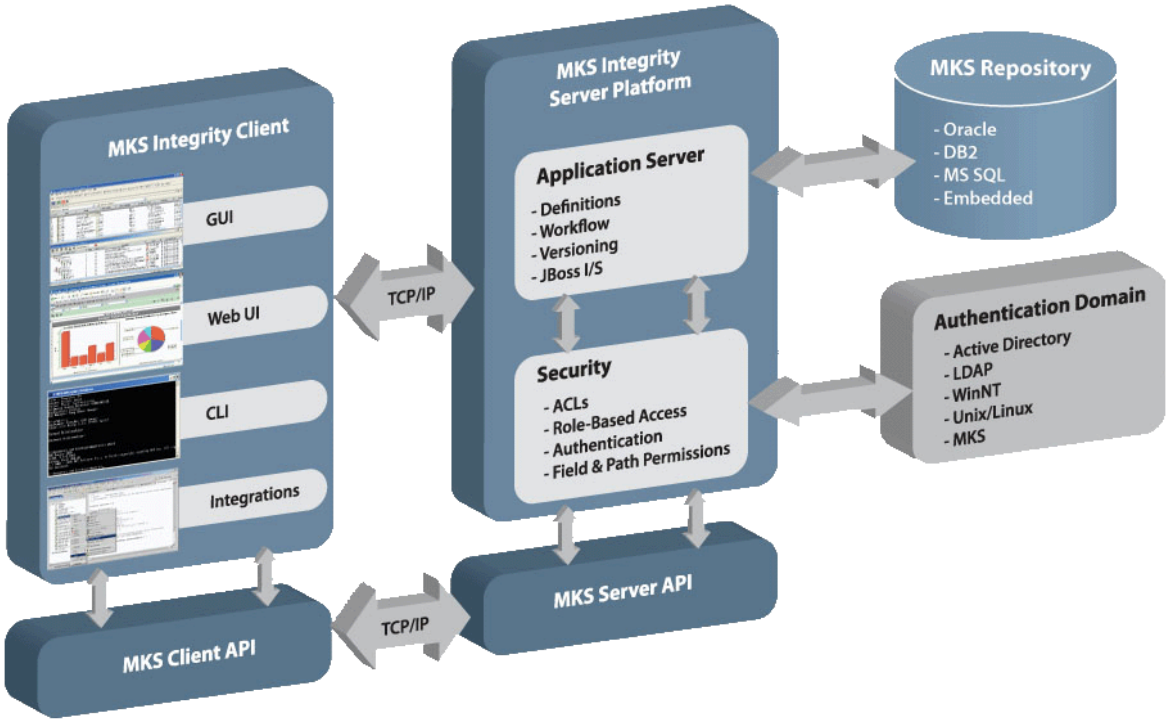


Abbildung 7: Architektur von PTC Integrity [6]

PTC Integrity besteht aus zwei funktionalen Hauptkomponenten:

- *Workflow and Documents*
- *Configuration Management*

Die Diplomarbeit ist eine Erweiterung im Teil *Workflow and Documents*.

Abbildung 8: Erweiterung im Teil *Workflow and Documents*

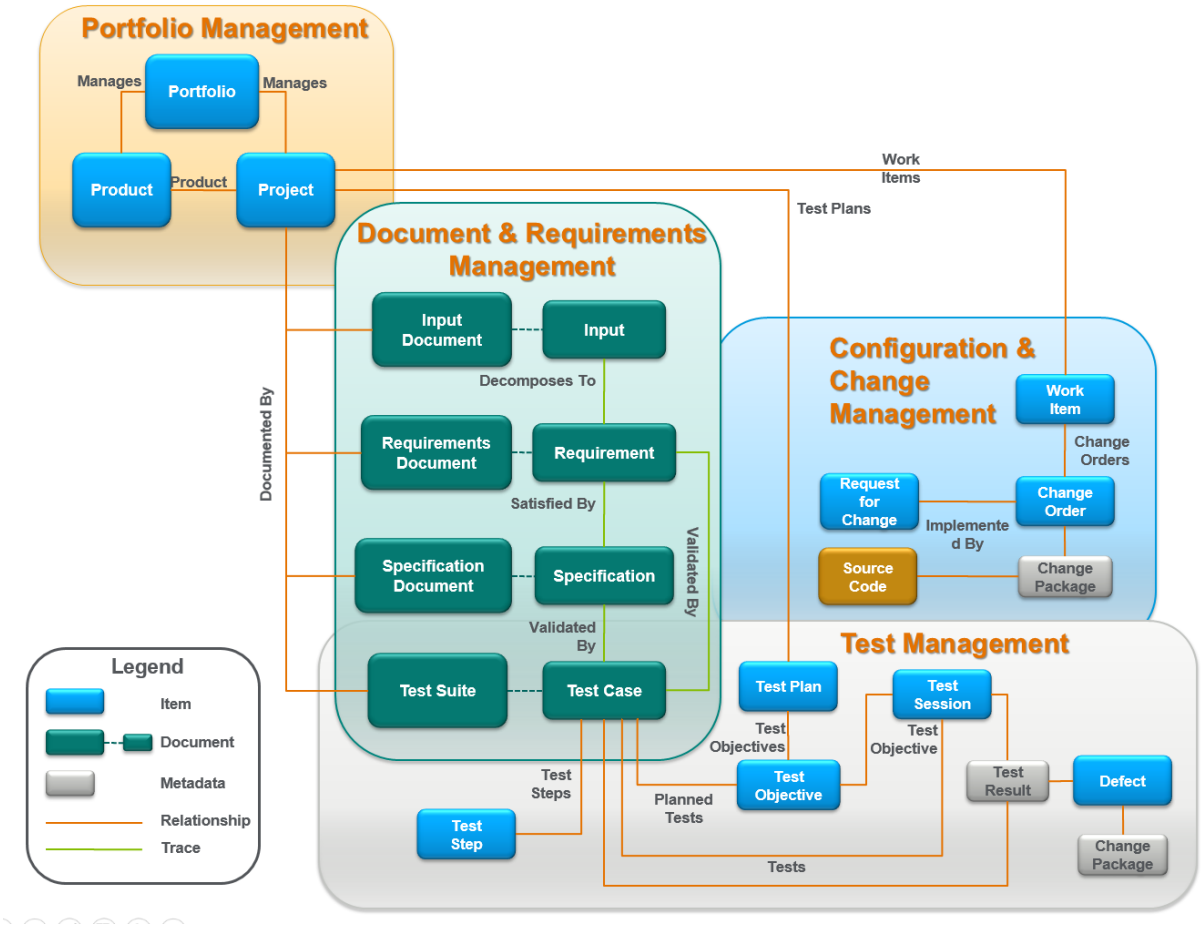


Abbildung 9: Datentyp von PTC Integrity [6]

Dieses schematische Bild zeigt die Datentypen in *Integrity Workflow and Documents. Integrity Workflow and Documents* verwaltet sogenannte Items, jedes Item wird als Datensatz in der zugrundeliegenden SQL-Datenbank angelegt.

Weiters werden in *Integrity* zusammengehörende Items zu Dokumenten zusammengefasst, es gibt vier verschiedene Dokumententypen:

- *Input Document*: Lastenheft
- *Requirement Document*: Pflichtenheft und weitere Anforderungsdokumente
- *Specification Document*: Beschreibung eines Designs von elektromechanischen Systemen oder von Software
- *Test Suite*: Test Spezifikation

4.2. Domänen

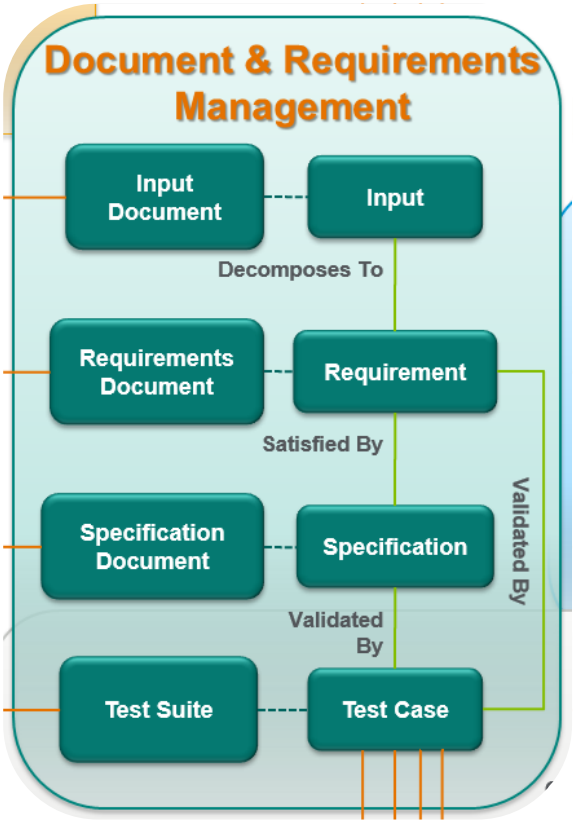


Abbildung 10: Domänen von PTC Integrity [6]

Eine Dokumentenart wie zum Beispiel *Input Document* oder *Requirement Document* wird auch als Dokument Domäne bezeichnet.

Die Graphik *Document & Requirements Management* zeigt in der linken Spalte alle derzeit vorhandenen Arten von Dokumenten und in der rechten Spalte die jeweils dazugehörigen *Items.*

Eine Domäne ist eine Gruppe die mehrere gleichartige Dokumente enthält. Ein Beispiel wäre das Lastenheft im *Input Document* oder das Pflichtenheft im *Requirement Document*.

Ein *Trace* beschreibt die Verbindung zwischen zwei Items. Wenn sich ein Item ändert, wird das zweite automisch mitverändert. Es gibt drei verschiedene Arten von *Traces*:

- Decomposes To
- Satisfied By
- Validated By

Welcher dieser *Traces* im Falle einer Verbindung gewählt wird, hängt von der Kombination der betroffenen Domänen ab.

5. Entstehung und Planung

5.1. Thema und Aufgabenstellung

5.1.1. Beschreibung der Diplomarbeit

Um den Sinn der Diplomarbeit so einfach wie möglich zu erklären, zeigen die folgenden Zeilen ein Beispiel für einen typischen Kopiervorgang:

Ein Lastenheft, also ein *Input Document*, ist in Integrity vorhanden, daraus soll der Mitarbeiter ein Pflichtenheft ableiten. Dazu müssen die Inhalte der *Items*, gespeichert in den Feldern des *Items*, vom Lastenheft in das Pflichtenheft gebracht werden. Vor der Diplomarbeit musste dieses durch erneutes, manuelles Erstellen durchgeführt werden. Mithilfe der Diplomarbeit können die *Items* einfach kopiert werden.

Das Kopieren von Items ist in *Integrity* nur beschränkt möglich, da Items nur innerhalb einer Domäne kopiert und später wieder eingefügt werden können. Der direkte, konventionelle Weg, ein Item zwischen zwei Domänen zu kopieren (also Ctrl+C und Ctrl+V) funktioniert nicht, da sich die einzelnen Dokumente und Felder der Domänen voneinander unterscheiden. Beispielsweise existiert ein Feld in der Quelldomäne, in der Zieldomäne allerdings nicht. Falls man nun einen Kopiervorgang durchführen möchte, würde dieser fehlschlagen. *DomainCopy* kümmert sich hauptsächlich um die Zuordnung dieser Felder, um einen fehlerfreien Kopiervorgang zu ermöglichen. Realisiert wird diese Zuordnung durch Konfigurationen, die der Benutzer nach seinen Wünschen anlegen, ändern oder löschen kann.

Ein Prototyp zum Kopieren zwischen Domänen ist bereits vorhanden.

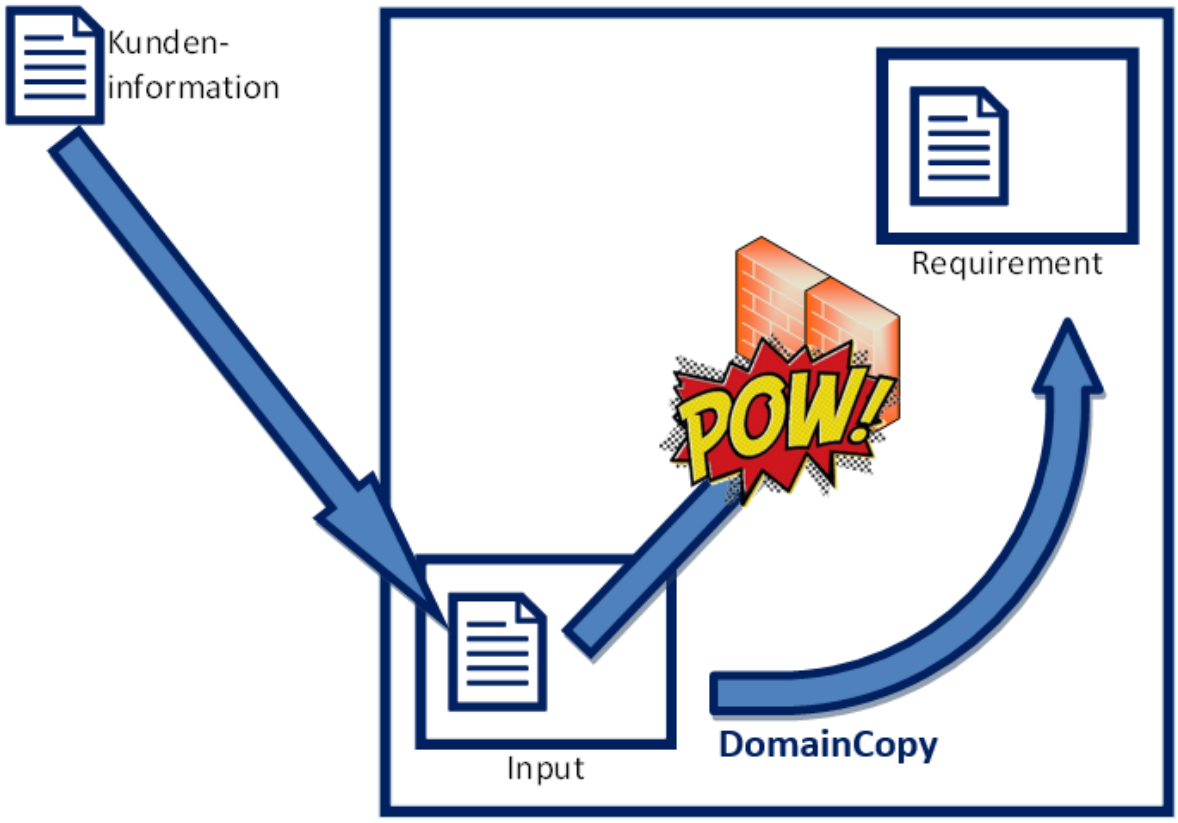


Abbildung 11: Darstellung von Integrity-DomainCopy

Integrity-DomainCopy ist für fünf Anwendungsfälle konzipiert:

- *Input Document → Requirement Document*
- *Requirement Document → Requirement Document*
  Dieser UseCase kann auch ohne *Integrity-DomainCopy* durchgeführt werden, da sich beide *Items* in einer Domäne befinden. Allerdings kann *Integrity-DomainCopy* beim Kopieren sofort einen *Trace* zwischen den Items generieren.
- *Requirement Document → Specification Document*
- *Requirement Document → Test Suite*
- *Specification Document → Test Suite*

Alle anderen Anwendungsfälle werden nicht zugelassen, es sei denn der Administrator fügt diese mithilfe von *Integrity-DomainCopy* hinzu.

5.1.2. Beschreibung des Prototyps

Der Prototyp, auf dem die Diplomarbeit aufbaut, bestand aus einem funktionierenden Kopiervorgang und der dazugehörenden Konfiguration.

Dieser hatte allerdings erhebliche Schwachstellen, die durch die Diplomarbeit ausgebessert wurden:

- keine Möglichkeit, die Konfiguration, und damit den Kopiervorgang, an die sich ständig verändernden Umstände anzupassen, außer diese manuell zu editieren
- kein Kopieren von Bildern oder Formatierungen (fett, kursiv, unterstrichen, …)
- Items wurden über eine temporäre Datei kopiert. Das heißt, die IDs der Items wurden in die Datei geschrieben und beim Einfügen wieder ausgelesen. Danach wurde die Datei wieder gelöscht.
- Eine GUI, die das Arbeiten in einer benutzerfreundlichen Umgebung ermöglicht, war nicht vorhanden.
- Keine Fehlerbehandlung → funktionierte etwas nichts, wurde eine Exception geworfen und das Programm beendet

5.1.3. Problembereich

Das Hauptproblem war, dass der Prototyp nicht für den täglichen Gebrauch tauglich war.

Es wurde nur ein bestimmter Kopiervorgang erlaubt, dieser war innerhalb des Programms nicht konfigurierbar. Folglich konnte der Prototyp nur für einen Kopiervorgang verwendet werden. Wollte man nun ein Item einer anderen Art kopieren, musste die Config-Datei des Prototyps manuell editiert werden.

All das war natürlich nicht benutzerfreundlich und konnte auch nicht ohne entsprechendes Fachwissen durchgeführt werden.

5.2. Zielsetzung

5.2.1. Generelle Ziele des Projekts

- Die Abläufe des Anlegens, Ändern und Löschen von Konfigurationen sowie der eigentliche Kopiervorgang sollen so einfach wie möglich gehalten werden.
    - Realisiert durch benutzerfreundliche Oberfläche und einfache Vorgangsweise.
- Dem User soll so viel Arbeit wie nur möglich abgenommen werden.
- Der User soll Konfigurationen erstellen/löschen/bearbeiten können.
- Der User kann die ausgewählten Konfigurationen während dem Kopiervorgang in einer Vorschau kontrollieren.

5.2.2. Geschäftsziel

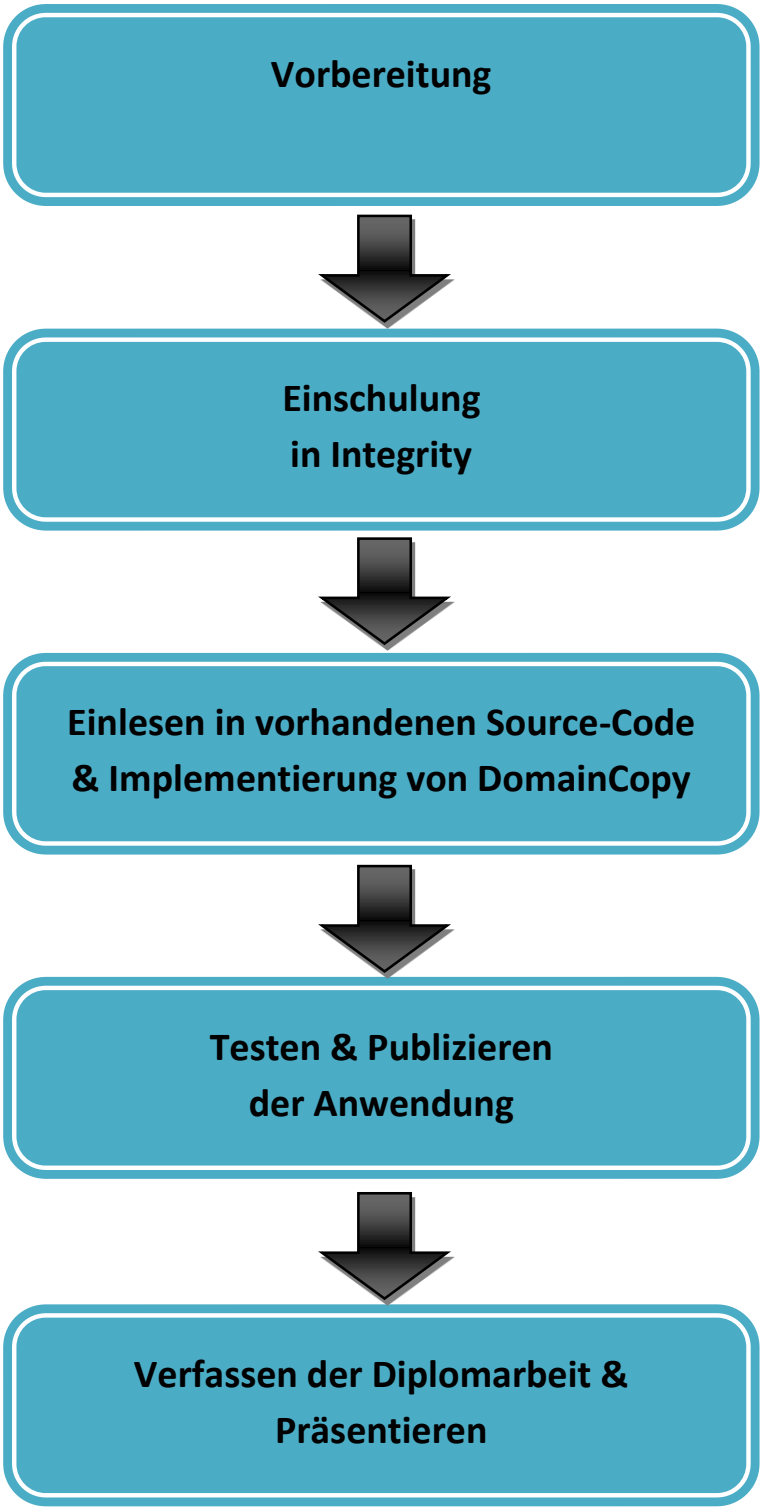
Mithilfe der Diplomarbeit wird der Arbeitsablauf der Mitarbeiter signifikant effizienter gestaltet. Zusätzlich wird das Arbeiten mit *PTC Integrity* erleichtert. So kann in der gleichen Zeit mehr erreicht werden.

5.2.3. Projektziel

Das Projektziel ist die Ablieferung eines funktionsfähigen, benutzerfreundlichen Programms. Alle Funktionalitäten sind zur Zufriedenheit des Auftraggebers implementiert und können einfach und ohne besondere Kenntnisse oder Schulungen verwendet werden.

5.3. Vorgehensweise

Der folgende Plan gibt eine grobe Übersicht, wie bei der Erstellung der Diplomarbeit vorgegangen worden ist:



5.3.1. Vorbereitung

Die Idee für die Diplomarbeit wurde von Berislav Klepic bereits im April 2015 vorgeschlagen, während der letzten Monate in der 4. Klasse an der HTL Perg.

Aufgrund des ersten Projektmeetings war von Anfang an klar, dass das Projektteam in der Firma Magna Powertrain als Werkstudenten angestellt wird und es somit keine Kommunikationsschwierigkeiten mit dem Auftraggeber geben wird. Der 13. Juli 2015 markierte den Beginn der Diplomarbeit Integrity – DomainCopy.

5.3.1.1 Erster Kontakt

Der erste Kontakt mit dem Auftraggeber bestand aus einer Mail, in der die Diplomarbeit mit kurzen Worten zusammengefasst wurde. Da bis dato allerdings noch nie mit *Integrity* gearbeitet, geschweige denn ein Programm darauf aufgebaut wurde, war der Inhalt dieser Mail für das Projektteam größtenteils kryptisch und unverständlich. Diese Wissenslücke wurde bei einem ersten Meeting, noch vor dem ersten Arbeitstag, gefüllt.

Bei diesem Meeting wurde außerdem der Startpunkt der Diplomarbeit gesetzt. In dem Treffen wurde grob geklärt, was zu tun ist und wie das Projekt realisiert werden könnte. Dem Projektteam wurden außerdem mehrere Unterlagen zum Thema *Integrity* und dessen Funktionsweise zur Verfügung gestellt, die immens halfen, die anfängliche Verwirrung zu lüften. Darüber hinaus wurde klar, dass die programmiertechnischen Arbeiten an der Diplomarbeit nur in der Firma erledigt werden können, aufgrund der Tatsache, dass das Programm *Integrity* nur in der Firma verwendet werden konnte und die Diplomarbeit auf dieses aufbaute. Das erforderte natürlich eine gewisse Vorausplanung, da nicht jederzeit an dem Projekt gearbeitet werden konnte.

5.3.2. Einschulung in Integrity

Der erste Arbeitstag, der 13. Juli 2015, startete sofort mit einer Einschulung in *Integrity*. Mithilfe von genaueren Präsentationen und Erklärungen wurde das Programm immer verständlicher. Kurz darauf konnte mit dem Programm bereits gearbeitet werden. So wurden die nächsten zwei Tage großteils damit verbracht, die Funktionalitäten von *Integrity* kennenzulernen und auszutesten, um eine Vorstellung zu bekommen, was mit dem Programm realisierbar ist und um später ohne Probleme damit arbeiten zu können.

5.3.3. Einlesen & Implementieren

Nach der Einschulung in die neue Technologie und dem Testen von *Integrity* wurde mit Phase 3 fortgefahren, dem Einlesen in den bereits vorhandenen Source-Code und das Implementieren von der Diplomarbeit.

Die Phase 3 lässt sich grob in folgende Punkte unterteilen:

- **Einlesen in Source-Code**
  - Da bereits Source-Code in Form eines Prototyps vorhanden war, musste dieser zuerst verstanden werden, um darauf aufbauen zu können. Das erwies sich wegen unterschiedlichen Programmierstilen bzw. einer etwas zweifelhaften Wahl der Variablenbezeichungen als relativ schwierig.

- **Design der Benutzeroberfläche**
  - Beim Entwerfen des Designs wurde ein besonderes Augenmerk auf die Benutzerfreundlichkeit und auf die intuitive Bedienung des Programms gesetzt. Weiters wurden etliche Fehlermeldungen entworfen, die dem Benutzer eine gezielte Fehlerbehandung und –behebung ermöglichen. Durch diese Fehlermeldungen sollte jeder Benutzer in der Lage sein zu erkennen, was eingegeben werden darf und was nicht.

- **Implementierung von *Integrity-DomainCopy***
  - Nachdem der Code des Prototyps einigermaßen verstanden war, wurde mit der Erweiterung desselben begonnen.
  - Der erste Punkt der Implementierung war eine Änderung des Speichervorgangs. Vor der Änderung wurde die ID der Items in einem File gespeichert, nun wird der Kopiervorgang über die OS-Zwischenablage geregelt. Unmittelbar nach der Änderung des Speichervorgangs wurden die ersten Funktionalitäten wie zum Beispiel *CopyConfig* oder *CopyConfigSetup* implementiert.

5.3.4. Testen & Präsentieren

5.3.4.1 Testen

Die Erweiterung wird später von diversen Mitarbeitern der Firma genutzt, dementsprechend sollte die Arbeit fehlerfrei und ohne Probleme ausführbar sein.

Realisiert wird diese Fehlerfreiheit durch entsprechende, vom Auftraggeber gewünschte Test-Cases, also Tests, deren Durchführung genau protokolliert wurde, sowie mithilfe von Unit-Tests.

Für weitere Informationen zum Thema *TestCases,* siehe Punkt 6.3.1

5.3.4.2 Präsentieren

Jedes Jahr wird an unserer Schule der sogenannte *P@bs-Award* abgehalten, an welchem die Projekte sowie die Diplomarbeiten der Maturanten zuerst einem Komitee und danach der Öffentlichkeit präsentiert werden.

5.3.5. Verfassen der Diplomarbeit

Da alle Punkte, welche die Arbeit umfasste, detailliert erklärt und schriftlich festgehalten werden mussten, nahm das Verfassen der Diplomarbeit relativ viel Zeit in Anspruch.

5.4. Funktionsumfang

5.4.1. Überblick Gesamtsystem

In dem folgenden Use-Case Diagramm wird das Grundsystem grafisch dargestellt.

Zu beachten ist, dass der Schritt *PTC Integrity* ein bereits existierendes Programm darstellt und nicht von uns realisiert wurde. Unser Programm dient lediglich als Erweiterung der Funktionalität von *PTC Integrity* und benötigt deswegen den vorderen Schritt.

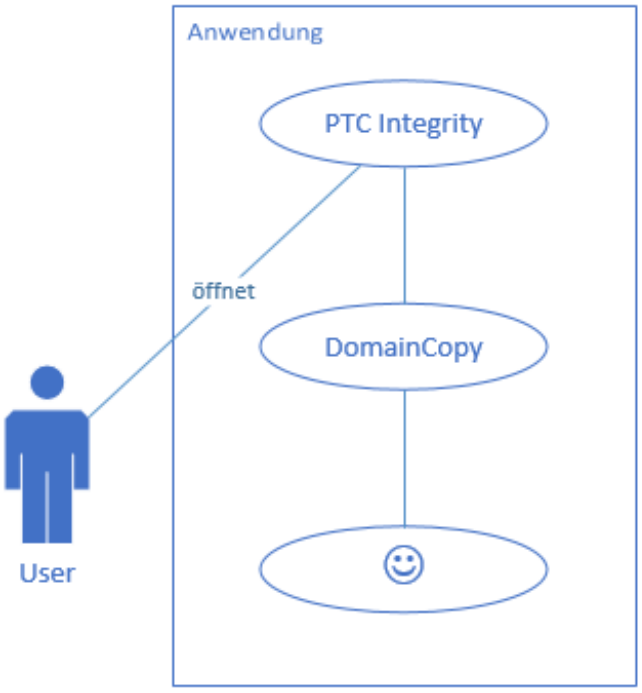


Abbildung 12: Use-Case Diagramm Gesamtsystem

5.4.2. Funktionalität Integrity-DomainCopy

Das folgende Use-Case Diagramm stellt die Diplomarbeit in einfachen Schritten dar:

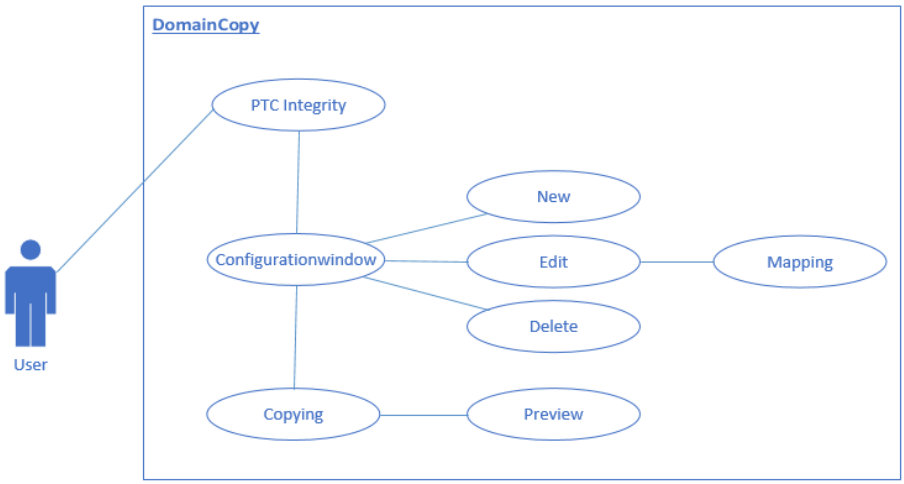


Abbildung 13: Use-Case Diagramm Integrity-DomainCopy

Der User öffnet das Programm *Integrity*. Falls etwas kopiert oder eine Konfiguration bearbeitet werden soll, wird über die Menüleiste unter dem Punkt *Custom* auf den Menüpunkt *ECSDomainCopy Config GUI* geklickt. So wird die eigentliche Diplomarbeit aufgerufen.

Daraufhin öffnet sich das Konfigurationsfenster. In diesem Fenster kann die gewünschte Konfiguration ausgewählt, eine neue erstellt, eine bereits vorhandene bearbeitet oder eine nicht mehr benötigte gelöscht werden.

Wenn eine neue Konfiguration erstellt werden soll, wird auf den Button *New* geklickt. Daraufhin öffnet sich ein neues Fenster, in dem man den Namen, eine kurze Beschreibung, die Quell- sowie die Zieldomäne und ob ein Trace (eine Verknüpfung) erstellt werden soll, eingeben kann.

Wird der Button *Edit* gedrückt, öffnet sich wiederum ein neues Fenster. Hier können die Eigenschaften der ausgewählten Konfiguration geändert, sowie ein neues Mapping angelegt werden.

Natürlich besteht auch die Möglichkeit eine Konfiguration mithilfe des Buttons *Delete* zu löschen.

Soll nun ein Item kopiert werden, wird einfach das Kontextmenü durch einen Rechtsklick auf das entsprechende Item aufgerufen und auf *DomainCopy Copy* geklickt.

Kurz vor dem Einfügen des Items, das auf dieselbe Weise wie das Kopieren funktioniert, wird eine Vorschau angezeigt. Mithilfe dieser kann kontrolliert werden, ob der Einfügevorgang so wie gewünscht durchgeführt werden wird. Sollte die Vorschau nicht den Wünschen des Users entsprechen, kann der Einfügevorgang abgebrochen werden.

Für eine genauere Beschreibung dieser Vorgänge liegt im Anhang unter Punkt 14.2 ein Ablaufdiagramm bei.

5.4.3. Funktionalität CopyConfig

Sobald der User in *Integrity* unter dem Menüpunkt *Custom* auf *ECSDomainCopy Config GUI* klickt, öffnet sich das sogenannte Konfigurationsfenster. In diesem Fenster können Konfigurationen für die jeweiligen Kopiervorgänge ausgewählt werden. Ist keine Konfiguration definiert, muss der User eine erstellen, ansonsten ist das Kopieren und Wiedereinfügen von Items nicht möglich. Hat der Benutzer eine Konfiguration erstellt und als aktiv markiert, wird diese für den nächsten Kopiervorgang verwendet.

Darüber hinaus wird in diesem Fenster mit dem Button *New* auf *CopyConfigSetup* (Punkt 5.4.4) und mit *Edit* auf *CopyConfigEdit* (Punkt 5.4.5) verwiesen. Mit *Delete* kann die ausgewählte Konfiguration gelöscht werden.

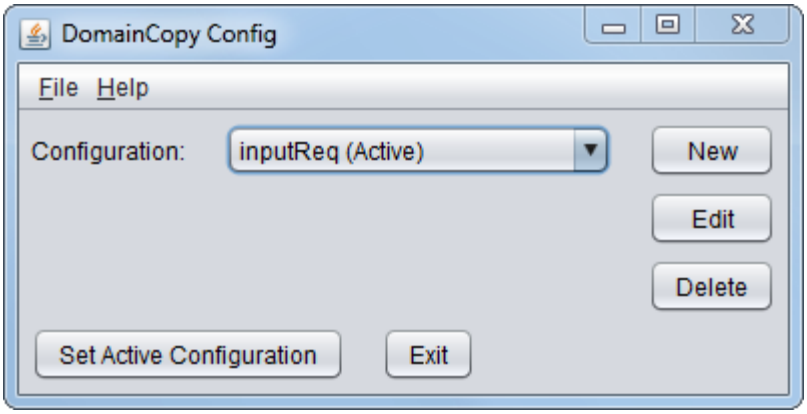


Abbildung 14: Konfigurationsfenster

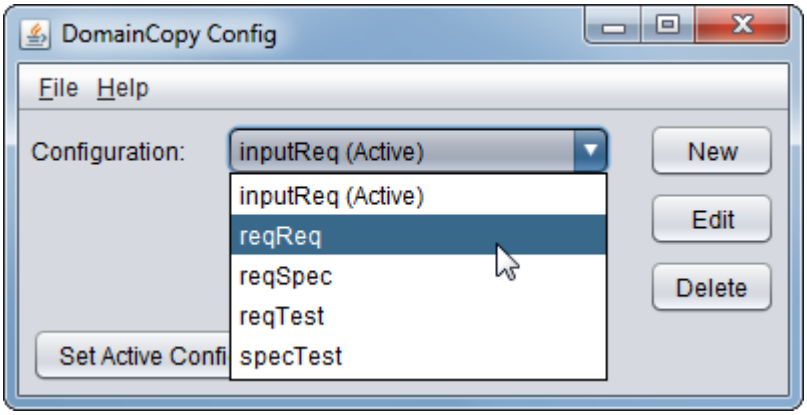


Abbildung 15: Auswahl der gewünschten Konfiguration

5.4.4. Funktionalität CopyConfigSetup

Will der Benutzer keine der bereits vorhandenen Konfigurationen benutzen bzw. bearbeiten, kann er natürlich eine neue erstellen. Dazu betätigt er den Button „New“. Nun kann der User eine neue Konfiguration erstellen, er muss bloß die gewünschten Eigenschaften eingeben. Diese sind:

- der Name - *Name*,
- eine kurze Beschreibung - *Description*,
- Quell- und Zieldomäne – *Source/Target*,
- ob ein Trace gezogen werden soll - *Trace* und
- ob die Vorschau vor jedem Einfügevorgang angezeigt werden soll – S*how Preview*.

Bei Druck auf *OK* wird die Konfiguration gespeichert und kann, nachdem die Field-Mappings richtig hinzugefügt wurden, beim nächsten Kopiervorgang verwendet werden.

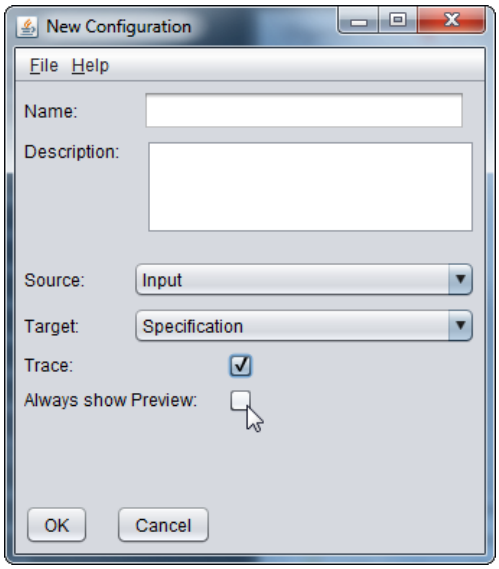


Abbildung 16: neue Konfiguration anlegen

5.4.5. Funktionalität CopyConfigEdit

Mit dieser Funktionalität können bereits existierende Konfigurationen bearbeitet werden. Grundsätzlich können alle Eigenschaften einer Konfiguration geändert werden, also *Name, Description, Show Preview, Trace* und *Mappings.* Die Quell- und Zieldomäne sind allerdings unveränderbar. Falls der User diese ändern will, muss er eine neue Konfiguration erstellen.

Funktionalität des Edit-Fensters (für genauere Informationen siehe Punkt 5.4.9 und 5.4.10):

- neues *Field Mapping* erstellen → mit dem Button *New FieldMapping* oder durch einen Doppelklick in eine leere Zeile der *FieldMapping* - Tabelle
- Löschen des selektierten *FieldMappings* → mit dem Button *Delete* oder durch manuelles Löschen in der *FieldMapping* - Tabelle
- Bearbeiten der *FieldMappings*

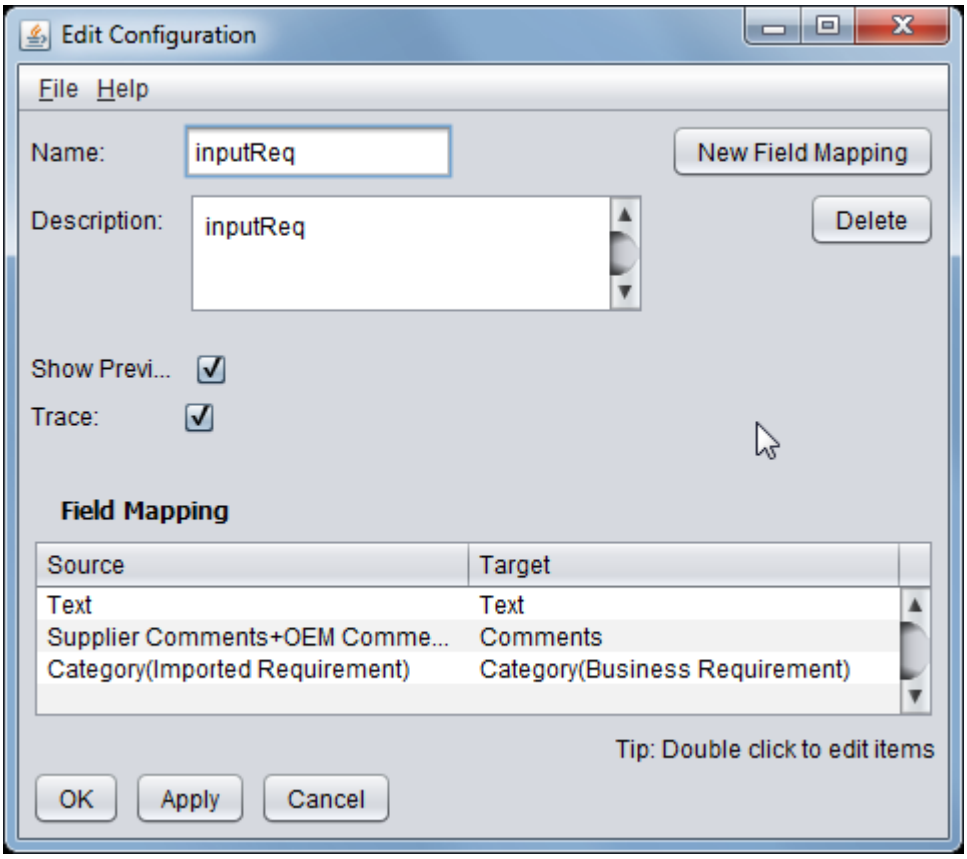


Abbildung 17: Details einer Konfiguration

5.4.6. Funktionalität DomainCopy

DomainCopy kopiert ein oder mehrere Content Items. Diese Funktionalität hat es bereits vor der Diplomarbeit gegeben, jedoch wurden die kopierten Items in einer File zwischengespeichert. Das bedeutet, dass bei jedem Kopiervorgang eine temporäre Datei entstand, die die einzelnen Items enthielt. Verständlicherweise könnte das bei vielen, parallelen Kopiervorgängen ein Problem darstellen. Diese Funktionalität wurde dahingehend verbessert, dass nun die Items bei jedem Kopiervorgang in der OS-Zwischenablage zwischengespeichert werden, anstatt temporäre Dateien zu erstellen.

Weiters konnten weder Bilder noch Formatierungen (Fett, Kursiv, Unterstrichen, etc.) kopiert werden. Die Inexistenz dieser Features wurde im Zuge der Diplomarbeit ausgemerzt.

5.4.7. Funktionalität DomainPaste

DomainPaste fügt die kopierten Content – Items wieder ein. Auch diese Funktionalität gab es bereits vor der Arbeit. Doch eine wesentliche Funktion fehlte noch: Falls der User schon länger keine Änderungen an den Konfigurationen getätigt hat und nicht mehr wusste, welche Konfiguration nun ausgewählt ist, hatte er keine komfortable Möglichkeit um das Ergebnis des Kopiervorgangs zu kontrollieren. Deswegen wurde *DomainPaste* um eine Vorschau-Funktionalität erweitert (Siehe 5.4.14).

Natürlich wurde auch die Möglichkeit, Bilder und formatierten Text wieder einzufügen, implementiert.

5.4.8. Funktionalität MaxCopyItems

MaxCopyItems dient lediglich als Schutzfunktion, um den erheblichen Netzwerk-Traffic zu vermeiden, der bei dem gleichzeitigen Kopieren tausender Items unweigerlich entsteht. Diese Funktionalität besteht nur aus einer Eigenschaft: „wie viele Items dürfen in einem Kopiervorgang kopiert werden?“. Vorgegeben wird dies durch den Integrity-Administrator, der die erlaubte Anzahl zu kopierenden Items in einer Datei festhält. Diese Datei ist nur vom Administrator bearbeitbar, der User, der das Programm verwendet, hat darauf keinen Zugriff.

5.4.9. Funktionalität FieldMappingOne2One

Diese Funktionalität bestimmt das einfache Mapping zwischen zwei Feldern zweier Domänen. Das bedeutet, der Inhalt des Feldes, das in der Spalte *Source* eingetragen ist, wird in das Feld in der Spalte *Target* kopiert.

Der Benutzer kann natürlich Mappings hinzufügen, diese bearbeiten oder bei Bedarf wieder löschen. Es muss allerdings mindestens ein Mapping vorhanden sein, um einen Kopiervorgang erfolgreich durchzuführen können: Das sogenannte *Category-Mapping* (die letzte Zeile in Abb. 18). Falls diese Mapping nicht definiert wurde, wird eine entsprechende Fehlermeldung ausgegeben.

Um festzustellen, ob das Mapping beim Kopiervorgang überhaupt erfolgreich durchgeführt werden kann, werden die einzelnen Felder und deren Zuordnungen überprüft. Falls ein Feld nicht existiert oder nicht auf ein bestimmtes anderes Feld gemappt werden kann, wird eine Fehlermeldung ausgegeben.

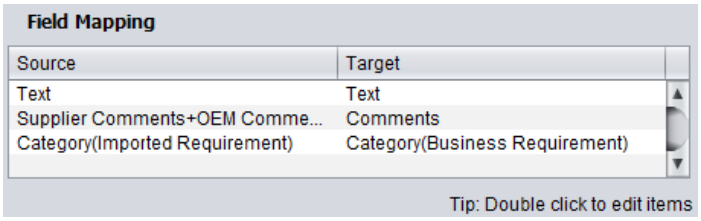


Abbildung 18: FieldMappings

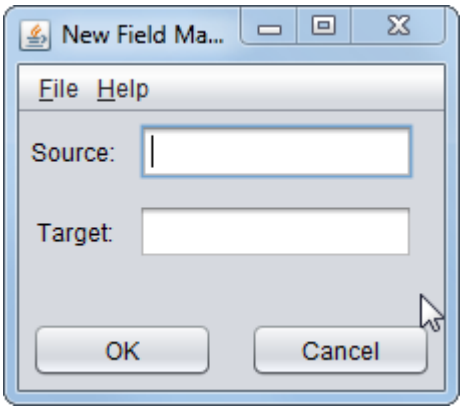


Abbildung 19: neues FieldMapping anlegen

5.4.10. Funktionalität MergeMapping

MergeMapping ist ähnlich wie *FieldMappingOne2One*, bis auf den Unterschied das hier mehrere Felder auf ein gemeinsames Feld durch den „+“ - Operator gemapped werden können.

Bsp.: zweite Zeile in Abb. 20

Hier werden die Felder *Supplier Comments* und *OEM Comments* auf *Comments* gemappt. Das bedeutet, dass der Inhalt dieser beiden Felder beim Kopiervorgang in das Feld *Comments* kopiert wird.

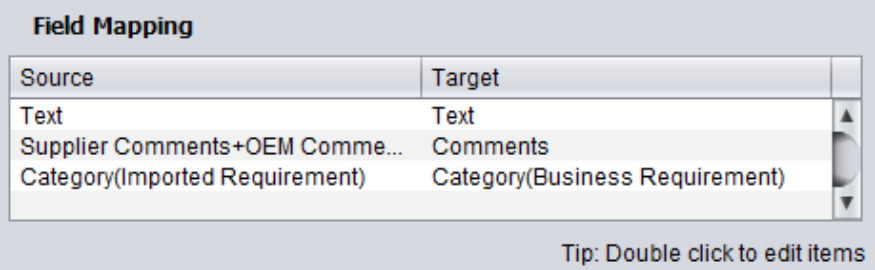


Abbildung 20: MergeMapping

5.4.11. Funktionalität CopyWithTrace

Bevor ein Item kopiert wird kann der User wählen, ob zwischen *Source* und *Target* ein Trace (eine Verknüpfung) gezogen werden soll oder nicht. Diese Verknüpfung dient dazu, Änderungen, die an einem Item durchgeführt werden, auch an dem zweiten Item auszuführen. Ob ein Trace besteht zeigt die Eigenschaft *Trace Status*.

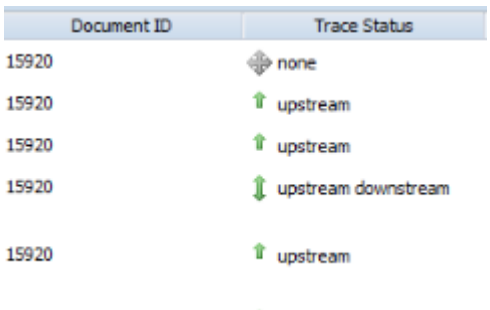


Abbildung 21: Darstellung eines Traces in PTC Integrity

5.4.12. Funktionalität Logging

Logging ist eine Funktion zur Aufzeichnung der einzelnen Kopier- und Einfüge-Vorgänge. Jedes Mal, wenn etwas kopiert oder eingefügt wird, wird dieser Vorgang in einem Log-Verzeichnis mit Datum und Uhrzeit mitprotokolliert. So kann der Administrator eventuelle fehlgeschlagene Kopiervorgänge genauer unter die Lupe nehmen und das Problem beheben.

5.4.13. Funktionalität Timeout

Diese Funktionalität dient wie *MaxCopyItems* lediglich als Schutzfunktion. Falls zwischen dem Kopier- und dem Einfügevorgang zu viel Zeit vergangen ist, wird dem Benutzer, der eventuell nicht mehr weiß, welche Konfiguration er ausgewählt hat, vor dem Einfügen eine Vorschau angezeigt, um sicherzugehen, dass genau das passiert, was der Benutzer wünscht. (mehr dazu im Punkt 5.4.14)

Weiters hat der Benutzer die Möglichkeit, den Kopiervorgang an dieser Stelle abzubrechen, sollte das notwendig sein.

5.4.14. Funktionalität Preview

Falls der Benutzer zu lange inaktiv war bzw. die Möglichkeit, immer Vorschauen anzuzeigen aktiviert hat, wird vor dem Einfügen der Items eine Vorschau angezeigt. Hier sieht der Benutzer was geschehen wird, sollte er sich entscheiden, den Kopiervorgang fortzusetzen. Andernfalls kann der Vorgang jederzeit abgebrochen werden.

Die folgende Darstellung zeigt, wie die Vorschau aussieht.

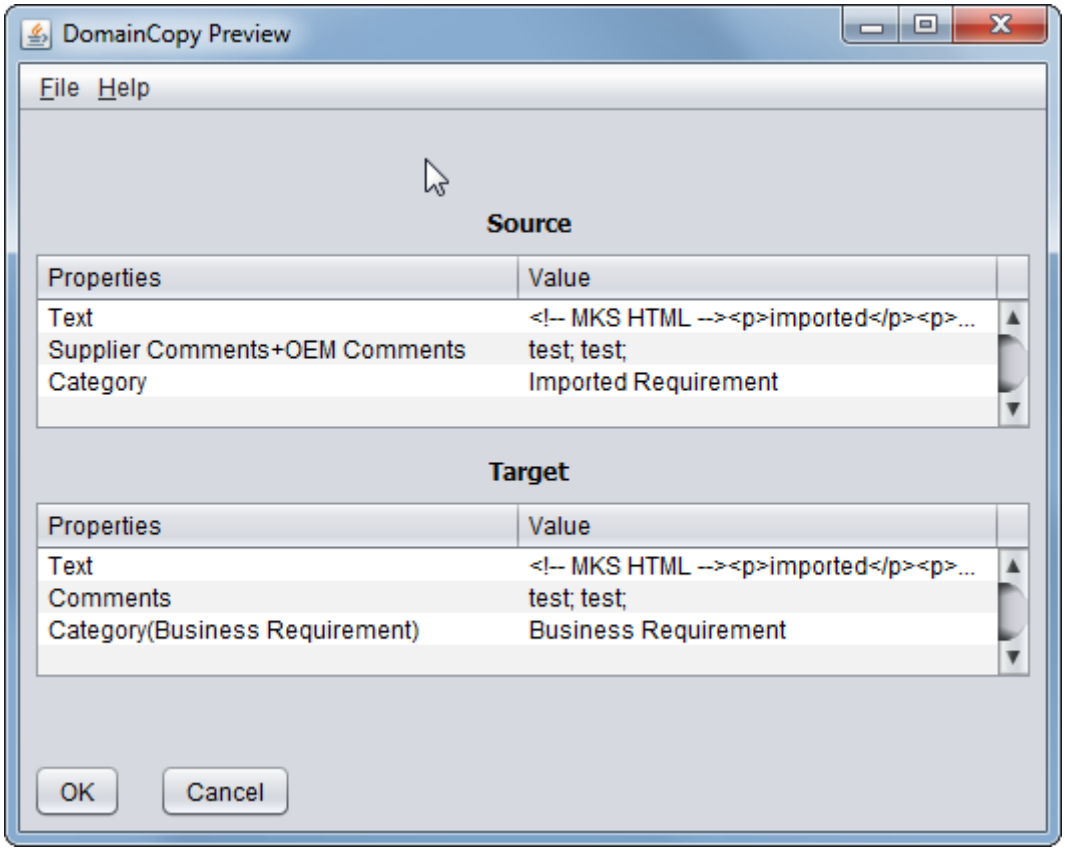


Abbildung 22: DomainCopy Preview (Vorschau)

5.4.15. Error-Handling

Da bei der Bedienung des Programms trotz aller Maßnahmen einiges schiefgehen kann, wurde eine extensive Fehlerbehandlung implementiert. Diese besteht hauptsächlich Fehlermeldungen, die aus dem User-Input generiert werden.

5.4.15.1 Fehler: Category-Mapping does not exist

Dieser Fehler tritt auf, wenn bei der Erstellung der Mappings einer Konfiguration kein *Category-Mapping* erstellt wurde und versucht wurde, die Konfiguration zu speichern.

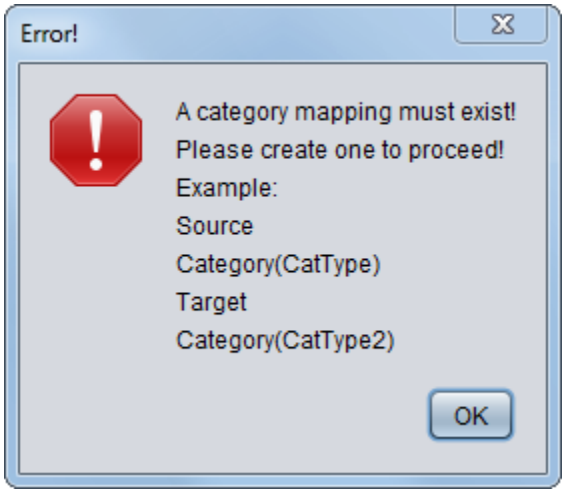


Abbildung 23: Category-Mapping does not exist

5.4.15.2 Fehler: Category does not exist

Dieser Fehler tritt auf, wenn ein *Category-Mapping* erstellt wurde, aber die angegebene Kategorie nicht in der jeweiligen Domäne existiert. In diesem Fall wird in der Fehlermeldung der Name der nicht existenten Kategorie angezeigt.

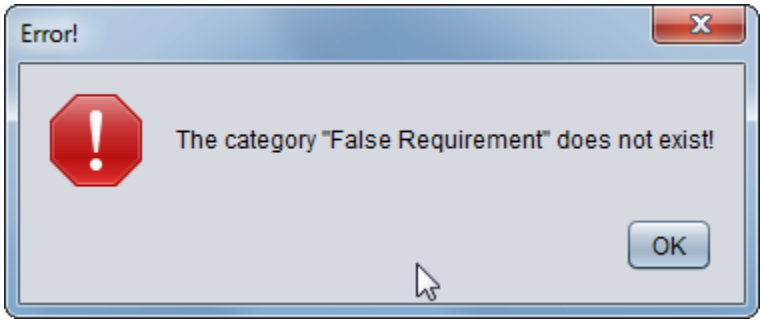


Abbildung 24: Category does not exist

5.4.15.3 Fehler: Item does not exist

Dieser Fehler tritt auf, wenn der Benutzer einen nicht existenten Itemnamen in die Mapping-Tabelle eingetragen hat. In diesem Fall werden in der Fehlermeldung der Name der nicht existierenden Items, sowie die entsprechende Domäne angezeigt.

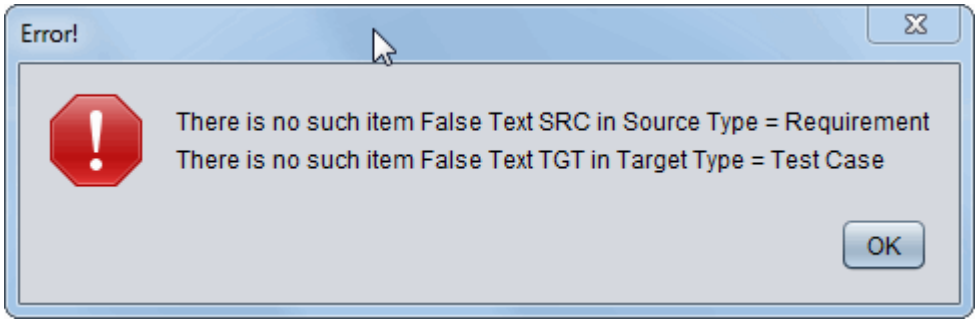


Abbildung 25: Item does not exist

5.4.15.4 Fehler: Amount of categories is not the same

Die entsprechende Fehlermeldung erscheint, wenn die Anzahl der Kategorien in *Source* und *Target* beim *Category-Mapping* unterschiedlich ist. Tritt dieser Fall auf, wird er abgefangen, da bei einer unterschiedlichen Anzahl von Kategorien Probleme wie beispielsweise eine Zuweisung einer Kategorie auf „nichts“ (*NullPointerException*) entstehen.

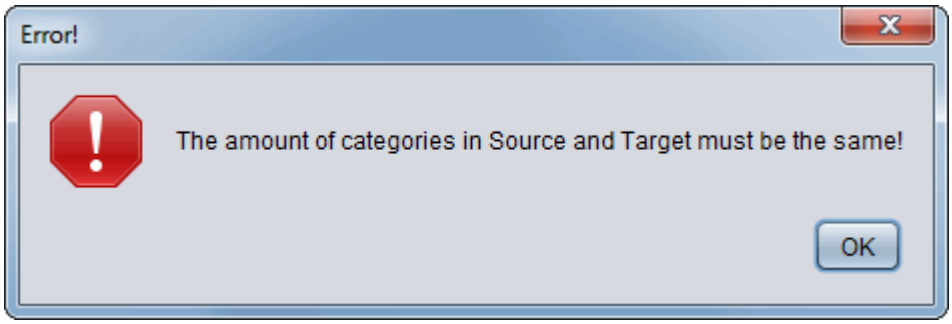


Abbildung 26: Amount of categories is not the same

5.4.15.5 *Fehler: ConfigName already exists*

Der Fehler tritt auf, wenn der Benutzer des Programms versucht, eine Konfiguration zu erstellen, deren Name bereits in der Liste der existierenden Konfigurationen enthalten ist. Der Grund, warum dieser Fehlerfall abgefangen wird, ist der Fakt, dass der Name einer Konfiguration einzigartig sein muss, um diese eindeutig identifizieren zu können.

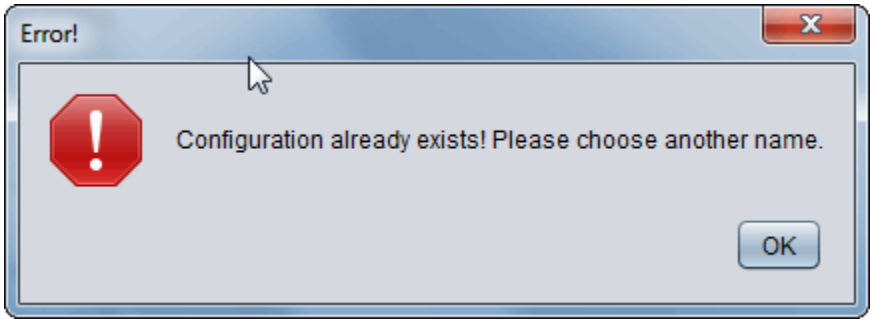


Abbildung 27: ConfigName already exists

5.4.15.6 *Fehler: Combination of Source and Target not allowed*

Gibt der Benutzer beim Erstellen der Konfiguration eine ungültige *Source-Target* Kombination ein, wie beispielsweise *Input --> Test*, erscheint diese Fehlermeldung. Grund für die Implementierung ist die Limitierung der Funktionalität von Seiten des Auftraggebers.

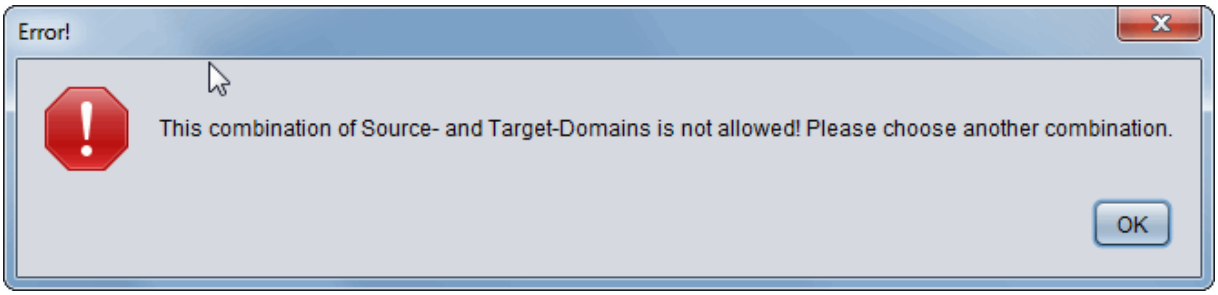


Abbildung 28: Combination of Source and Target not allowed

5.4.15.7 Fehler: File does not exist

Dieser Fehler tritt auf, wenn eine Datei, die für den fehlerfreien Ablauf des Programms notwendig ist, nicht vorhanden ist. In der Fehlermeldung wird der Name der fehlenden Datei angezeigt.

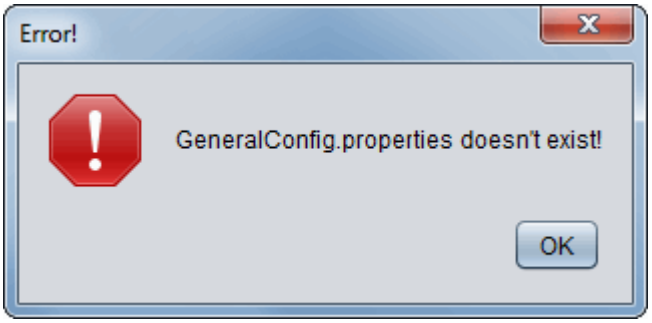


Abbildung 29: File does not exist

5.4.15.8 Fehler: Category is not mapped

Die entsprechende Fehlermeldung erscheint, wenn die Kategorie des Items, das kopiert werden möchte, nicht in der Konfiguration gemappt ist. Abgefangen wird dieser Fall, da wiederum Probleme beim Kopiervorgang auftreten werden, wenn ein Item Teil einer Kategorie ist, die nicht gemappt wurde.

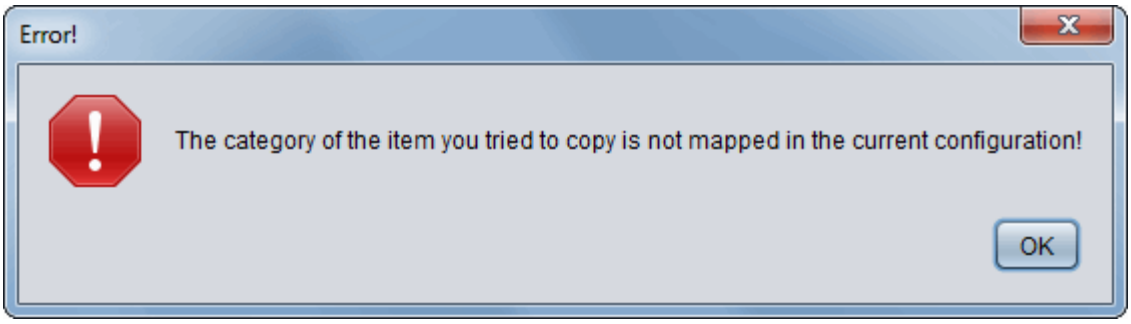


Abbildung 30: Category is not mapped

5.4.15.9 Fehler: Source Domain is not specified

Dieser Fehler tritt auf, wenn der User ein Item aus einer Domäne kopieren möchte, die nicht als Quelldomäne in der derzeit ausgewählten Konfiguration angegeben ist.

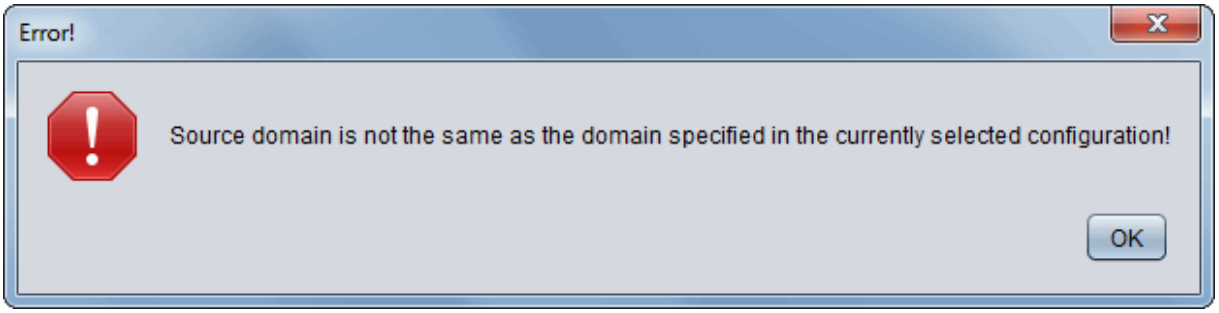


Abbildung 31: Source Domain is not specified

5.4.15.10 Fehler: Target Domain is not specified

Versucht der Benutzer ein kopiertes Item in einer nicht als Zieldomäne in der derzeit ausgewählten Konfiguration angegebenen Domäne einzufügen, tritt dieser Fehler auf.

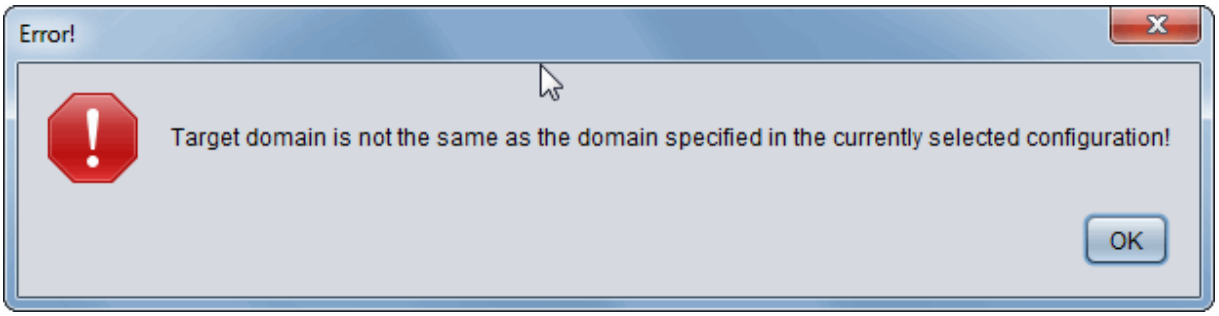


Abbildung 32: Target Domain is not specified

5.4.15.11 Fehler: MaxCopyItems exceeded

Diese Fehlermeldung erscheint, wenn der Benutzer versucht, mehr Items als erlaubt in einem Kopiervorgang zu kopieren. Wie viele Items gleichzeitig kopiert werden dürfen, wird durch die Funktionalität *MaxCopyItems* (Punkt 5.4.8) vorgegeben.

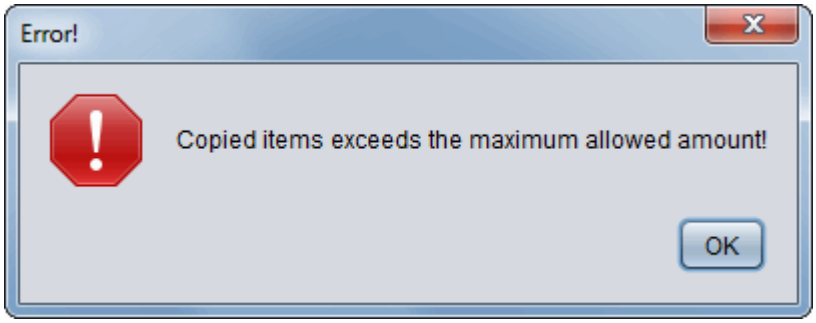


Abbildung 33: MaxCopyItems exceeded

5.5. Release einer Integrity ECS Integration

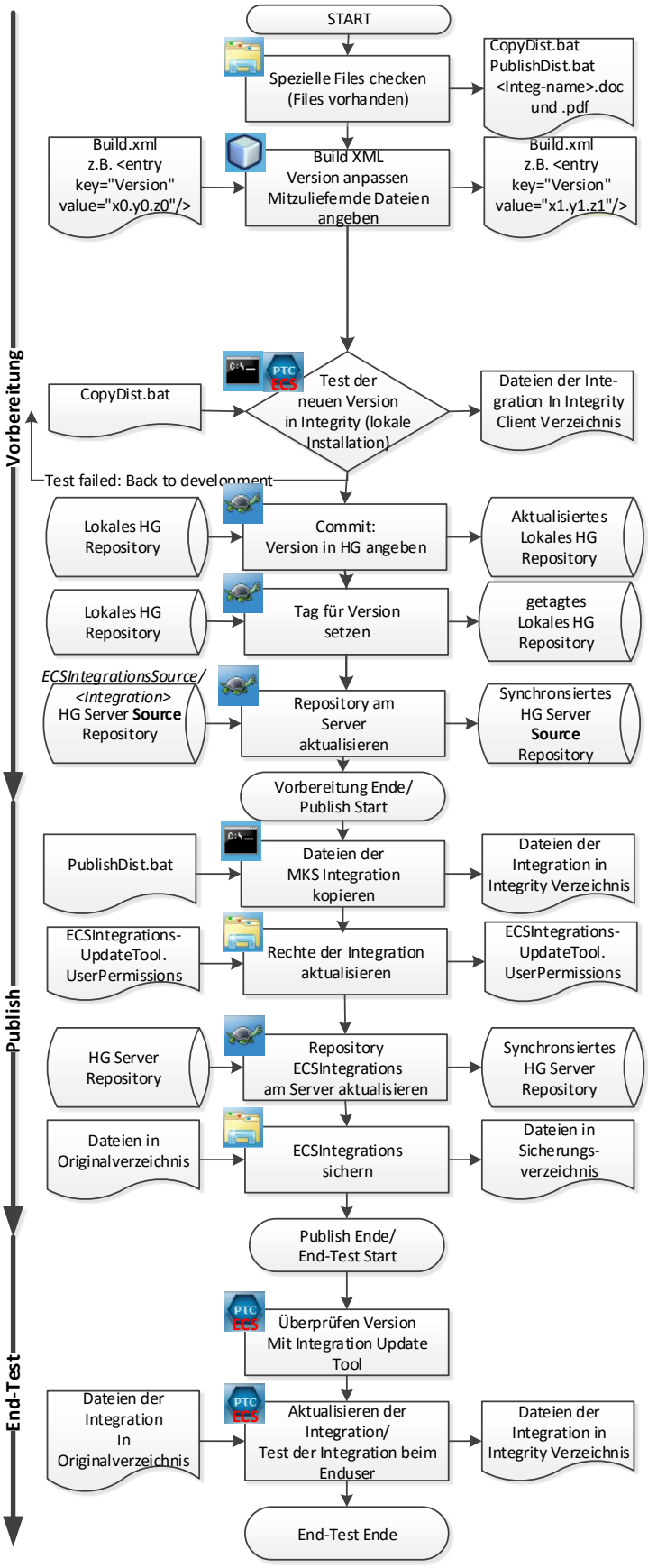


Abbildung 34: Release einer Integrity ECS Integration [9]

Der Release-Prozess einer *Integrity* Integration ist in der MAGNA genau geregelt. Dieser Prozess musste auch bei der Diplomarbeit durchlaufen werden, damit die Integration später verwendet werden kann.

5.5.1. Vorbereitung

In dieser Phase geht es hauptsächlich um den finalen Test der Integration sowie um das Pushen der benötigten Daten auf den firmeninternen Server via *Tortoise HG.* Zuallererst wird sichergestellt, dass spezielle Dateien, die für die Integration benötigt werden (wie beispielsweise die Konfigurations-Dateien oder die Dokumentation der Anwendung), vorhanden sind. Weiters werden die aktuelle Versionsnummer und mitzuliefernde Daten (z.B.: *GeneralConfig.properties, DomainConfig.prop*) in die *Build.xml* geschrieben.

Danach wird der finale Test durchgeführt. Falls hierbei Bugs oder Fehler auftreten, wird wieder an dem Projekt gearbeitet.

Ist beim Test alles in Ordnung, wird nur noch die Versionsnummer in *Tortoise HG* angegeben, ein passender Tag (z.B.: „V1_2_3“) gesetzt und die Daten auf den Server gepusht.

5.5.2. Publish

Mithilfe der *PublishDist.bat* werden nun die Daten der Integration in das Integrity Verzeichnis kopiert. Anschließend werden die Rechte der Integration im *UpdateTool* (eine von MAGNA programmierte Anwendung, die die einzelnen Integrationen am aktuellsten Stand hält) aktualisiert.

Danach wird das Repository in *Tortoise HG* nochmals aktualisiert und die Integration gesichert.

5.5.3. End-Test

In dieser Phase wird nun versucht, die Integration beim Enduser mithilfe des *UpdateTools* in *Integrity* zu laden. Abschließend wird die Integration beim User getestet.

5.6. Hinzufügen einer Integration in Integrity

Die Programmdateien von *Integrity-DomainCopy* liegen am firmeninternen Server.

- Der Ordner *ECSDomainCopy* muss vom Server in das *Integrity*-Installationsverzeichnis kopiert werden.
- Danach wird *Integrity* gestartet und unter dem Menüpunkt *Viewset* → *Customize…* → *Custom* → Bei neuem Eintrag *Edit…* → Hier werden alle benötigten Daten eingegeben

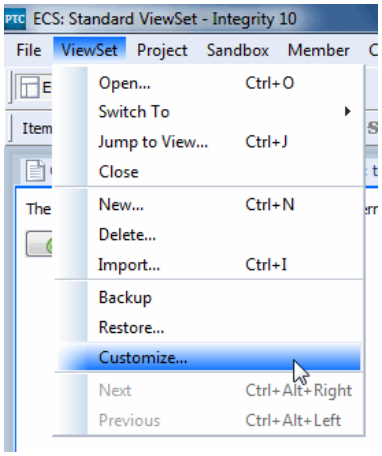


Abbildung 35: Viewset --> Customize

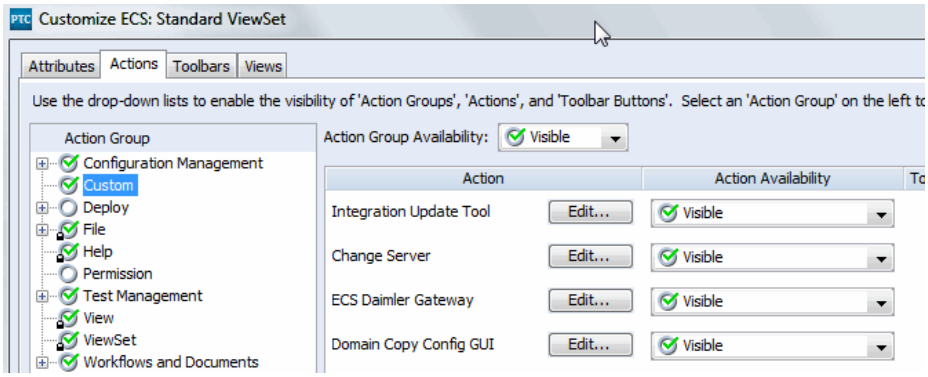


Abbildung 36: Customize Viewset

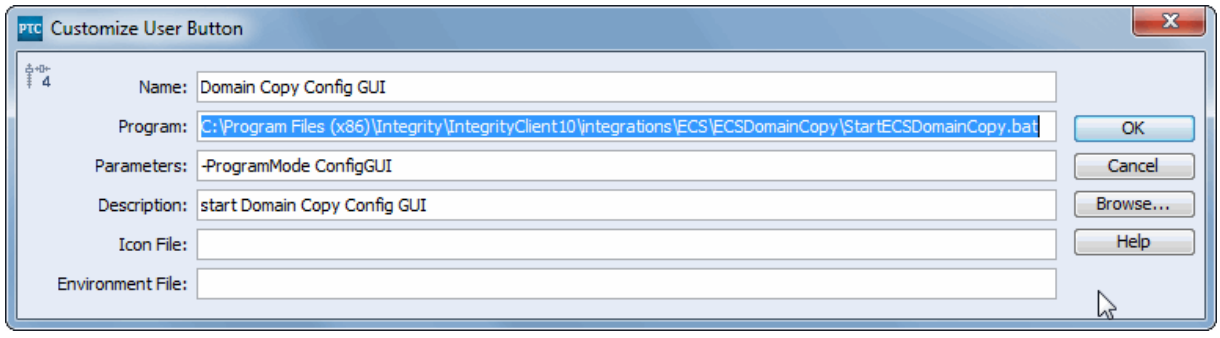


Abbildung 37: Daten für das Hinzufügen der Integration

- Bei *Action Availability* muss Visible ausgewählt werden

Sind diese Schritte erledigt, kann die Integration mithilfe eines Menüpunktes unter *Custom* aufgerufen werden.

5.7. Verwendete Ressourcen

5.7.1. Software

5.7.1.1 NetBeans IDE

- NetBeans ist eine kostenfreie Entwicklungsumgebung, die es ermöglicht, Java-Anwendungen zu programmieren. Es ist die offizielle IDE für Java 8 und ist komplett in Java implementiert.
- NetBeans läuft auf mehreren OS-Systemen, wie zum Beispiel Solaris, Mac OS und Linux, die eine Java Virtual Machine (JVM) installiert haben.
- NetBeans besitzt Benutzerschnittstellfunktionen, Quellcode-Editor, GUI-Editor (Graphical User Interface), Versionskontrolle außerdem unterstützt es verteilte Anwendungen (wie zum Beispiel CORBA, RMI, etc.) und Web-Anwendungen (JSPs, Servlets, etc.). [10]
- Da alle Entwickler der Firma MAGNA NetBeans verwenden, wurde es auch im Rahmen der Diplomarbeit verwendet.

Abbildung 38: NetBeans IDE

5.7.1.2 PTC Integrity

- PTC Integrity ist ein Software-Lifecycle-Management-System und eine Application-Lifecycle-Managment-Plattform (ALM), welche von der Firma MKS entwickelt und erstmals 2001 veröffentlicht wurde.
- Integrity unterstützt ganzheitliche Systeme und Software-Design um zu definieren, wie Produkt, Anwender und die Umwelt wirklich miteinander interagieren.
- 2010 wurde die erste stabile Version publiziert, wobei die Anwendung immer noch in aktiver Entwicklung steht.
- Die Software nutzt die Client/Server – Architektur und bietet sowohl eine Desktop- als auch eine Webanwendung für das Arbeiten mit Integrity. [9]
- In der Firma MAGNA wird Integrity zur besseren Verwaltung mehrerer Projekte, Prozesse und Vorgänge verwendet.

Abbildung 39: PTC Integrity

5.7.1.3 TortoiseHg

- TortoiseHg ist ein freier Client für den Versionsverwaltungs-Dienst Mercurial.
- Es steht unter der GNU General Public License (GPL).
- TortoiseHg ist als Shell-Erweiterung implementiert. Das bedeutet, es integriert sich in den Windows-Explorer (bzw. Nautilus oder Gnome) und ist daher außerhalb einer IDE (wie beispielsweise NetBeans) verwendbar.
- Die Hauptaufgabe ist eine Versions-, Revisions- und Sourcekontrolle. [11]
- Dieses Programm wird in der Firma MAGNA hauptsächlich für Publishing-Zwecke verwendet. Da die Diplomarbeit keine Ausnahme in der Regel des Publishing-Prozesses darstellt, wurde dieses Programm benutzt.

Abbildung 40: TortoiseHg

6. Umsetzung

6.1. Allgemeines

Am 13. Juli 2015 wurde eine Einschulung in das Programm *Integrity* durch den entsprechenden Zuständigen abgehalten, um zu verstehen, wie dieses aufgebaut ist, wie es funktioniert und wie es verwendet wird.

In den folgenden Tagen wurde der bereits vorhandene Source-Code durchleuchtet, um eine Idee zu bekommen, wie die Diplomarbeit zu realisieren ist. Sobald ein gewisser Durchblick gewonnen wurde, wurde mit der Programmierung der Funktionalitäten begonnen.

6.2. Funktionalitäten

6.2.1. Funktionalität CopyConfig

Als Kernteil des Projekts wurde diese Funktionalität so bald als möglich erledigt. Anfangs enthielt die Konfiguration des Kopiervorgangs nur einen Namen, eine Beschreibung und eine *Source*- und *Targetdomain*. Später wurden die Eigenschaften einer Konfiguration um die Möglichkeit einen Trace (eine Verbindung) zwischen den kopierten Items zu ziehen, erweitert. Weiters wurde dem Benutzer bereits bei der Erstellung der Konfiguration die Wahl gestellt, ob die Vorschau eines Kopiervorgangs immer, oder erst ab einer gewissen Leerzeit angezeigt werden soll.

Die einzelnen Konfigurationen werden in *.prop*-Dateien abgespeichert, um sie im Programm leicht auslesen zu können. In den folgenden Kapiteln wird nun der Inhalt dieser *.prop*-Dateien, also der Konfigurationen, genauer erklärt.

6.2.1.1 Vom Benutzer erstellte Konfigurationen

```
Name=inputReq
Description=asdfe
Source=Input
Target=Requirement
Trace=False

inputType=Input
reqType=Requirement
specType=Specification
testType=Test

relationshipFieldBetweenSameTypes=Is Related To'
relationshipFieldBetweenInputAndReqTypes=Decomposed From
relationshipFieldBetweenReqAndSpecTypes=Satisfies
relationshipFieldBetweenReqAndTestTypes=Validates
relationshipFieldBetweenSpecAndTestTypes=Validates

srcFields=Text,Supplier Comments+OEM Comments,Category(Imported Requirement),
targetFields=Text,Comments,Category(Business Requirement),
```

Abbildung 41: Benutzerkonfigurationen

Die Felder *Name, Description, Source, Target* und *Trace* werden vom Benutzer bei der Erstellung der Konfiguration angegeben.

Der nächste Block an Information beschreibt, wie die einzelnen Domänen vom *Integrity* im Programm verwendet und beschrieben werden. Mehr dazu in Punkt 6.2.1.2.

Anschließend werden die Arten von Traces beschrieben, die für eine bestimmte Kombination von *Source* und *Target* verwendet werden dürfen. Beispielsweise darf für eine Konfiguration zwischen zwei identen Domänen nur die Trace-Art *Is Related To* verwendet werden, während für Konfigurationen zwischen *Input* und *Requirement* nur die Art *Decomposed From* in Frage kommt.

Die letzten beiden Zeilen beschreiben das Mapping der Konfiguration. Dem Beispiel der obigen Konfiguration folgend, wird hier der Inhalt des Feldes *Text* auf das Feld *Text* der Zieldomäne gemappt. Weiters werden die Inhalte von *Supplier Comments* und *OEM Comments* gesammelt in das Feld *Comments* der Zieldomäne geschrieben. Weiters existiert noch ein sogenanntes Kategorie-Mapping.

Genaueres zum Mapping unter Punkt 6.2.7 und 6.2.8.

6.2.1.2 Vom Programm erstellte Konfigurationen (GeneralConfig.prop, SelectedConfig.prop, DomainConfig.conf)

Neben den Konfigurationen, die vom User generiert werden, existieren auch noch solche, die vom Programm erstellt wurden, um einige Konstanten vorzugeben. Diese Dateien können vom Benutzer des Programms nicht verändert werden, sehr wohl aber vom Administrator, der den Inhalt dieser Dateien manuell ändert.

```
1  <Domains
2  Input,Input Document,Input
3  Requirement,Requirement Document,Requirement
4  Specification,Specification Document,Specification
5  Test,Test Suite,Test Case
6  Domains/>
7
8  <Relationships
9  Input,Requirement,Decomposed From
10 Requirement,Requirement,Decomposed From
11 Requirement,Specification,Satisfies
12 Requirement,Test,Validates
13 Specification,Test,Validates
14 Relationships/>
15
16 <MaxCopyItems
17 5
18 MaxCopyItems/>
```

Abbildung 42: vom Programm erstellte Konfiguration

Der erste Informationsblock beschreibt die einzelnen Domänen, die derzeit in *Integrity* verfügbar sind. Der Administrator kann hier jederzeit seine eigenen Domänen eintragen, falls der Bedarf besteht.
Der Inhalt des *Domain*-Blocks ist folgendermaßen aufgebaut:

Input, Input Document, Input
Domänenname, Dokumentenname, Itemname

Der erste Eintrag einer Zeile beschreibt den Namen der Domäne (z.B.: *Input, Text, …*), der darauffolgende den Namen eines Dokuments der Domäne (z.B.: *Input Document, Test Suite, Requirement Document, …*). Beim letzten Eintrag handelt es sich um die interne Bezeichnung des Namens eines Dokumentes einer Domäne in *Integrity* (z.B.: *Input, Test Case, …*).

Der *Relationships*-Block beschreibt, zwischen welchen Domänen ein *Trace* gezogen werden darf. Falls der Wunsch besteht, einen *Trace* auch zwischen anderen Domänen zu ziehen bzw. die vorhandenen Beziehungen zu bearbeiten oder löschen, kann der Administrator diese Änderungen jederzeit vornehmen.
Der Inhalt dieses Blocks ist folgendermaßen aufgebaut:

Input, Requirement, Decomposed From
Quelldomänenname, Zieldomänenname, Trace-Name

Der erste Eintrag beschreibt den internen *Integrity*-Namen der Quelldomäne, also die Domäne, von der kopiert werden soll. Der zweite Eintrag beschreibt wiederum den internen Namen der Zieldomäne, also wohin kopiert werden soll. Schließlich handelt es sich beim letzten Eintrag um den Namen des zu verwendenden *Trace* in *Integrity*.

Alle Informationen, die in den Konfigurationen zu finden sind, werden vom Programm ausgelesen und dort an der benötigten Stelle verwendet bzw. angezeigt.

6.2.2. Funktionalität CopyConfigSetup

CopyConfigSetup beschreibt das eigentliche Erstellen einer Konfiguration.

6.2.2.1 Ablauf

Nachdem der Benutzer die gewünschten Daten eingegeben hat (Name, Beschreibung, *Source-/Target-Domain*, Trace, Vorschau), wird eine Datei im *Properties*-Ordner erstellt. Diese Datei stellt dabei nur das Grundgerüst einer Konfiguration dar (Mehr Informationen zum Aufbau unter Punkt 6.2.1). Der wichtigste Teil fehlt noch; nämlich das *Field-Mapping*. Dieses wird erst beim ersten Bearbeiten der Konfiguration hinzugefügt.

6.2.3. Funktionalität CopyConfigEdit

CopyConfigEdit ist das einfache Bearbeiten der Konfiguration.

6.2.3.1 Ablauf

Die Änderungen, die der User vornimmt, werden wieder in die .prop-Datei der Konfiguration zurückgeschrieben, sodass die Änderungen permanent gespeichert sind.

6.2.4. Funktionalität DomainCopy

Da diese Funktionalität einen weiteren Kernteil der Diplomarbeit darstellt, wurde ihr eine erhöhte Priorität eingeräumt. Sie ermöglicht das Vormerken von den zu kopierenden Item-IDs. Diese werden im Zuge des Einfügevorgangs verwendet, um die Daten der Items, die durch die ID eindeutig identifiziert werden, an anderer Stelle wieder einzufügen.

Da *DomainCopy* schon als Prototyp verfügbar war, wurde nicht viel an der Grundfunktionalität geändert, abgesehen von der Änderung, dass die IDs der zu kopierenden Items nicht mehr in einer temporären Datei zwischengespeichert werden, sondern innerhalb der OS-Zwischenablage.

6.2.4.1 Ablauf

Sobald der Benutzer den Kopiervorgang einleitet, wird überprüft, ob sich der Benutzer in der von der derzeit ausgewählten Konfiguration angegebenen Quelldomäne befindet, sowie ob nicht mehr Items als erlaubt markiert wurden. Ist das in Ordnung, werden die IDs der markierten Items in die OS-Zwischenablage geschrieben. Von dort werden die IDs beim Einfügevorgang wieder ausgelesen und mit deren Verwendung weitergearbeitet (weitere Informationen unter Punkt 6.2.5).

6.2.5. Funktionalität DomainPaste

DomainPaste stellt das Gegenstück von *DomainCopy* dar. Die Items, die *DomainCopy* kopiert, fügt *DomainPaste* wieder ein. Auch diese Funktionalität war bereits im Prototypen vorhanden, sie wurde jedoch soweit verändert, dass die Items aus der OS-Zwischenablage ausgelesen werden, Bilder wieder eingefügt werden können sowie Formatierungen (Fett, Kursiv, Unterstrichen, etc.) kopiert werden können.

6.2.5.1 Ablauf

Leitet der Benutzer einen Einfügevorgang ein, wird zuallererst überprüft, ob der Benutzer sich in einer validen Zieldomäne befindet. Vorgegeben wird diese Eigenschaft durch die derzeit ausgewählte Konfiguration. Wird hier kein Fehler festgestellt, werden die IDs der einzufügenden Items aus der OS Zwischenablage ausgelesen.

Anschließend wird unter bestimmten Voraussetzungen eine Vorschau des Einfügevorgangs, also wie das „Endprodukt“ aussehen wird, angezeigt. Für eine genauere Beschreibung siehe Punkt 6.2.11.

Danach erst wird das Item am Zielort erstellt. Verwendet wird dazu das *createcontent*-Kommando, das das neue Item mit den Daten des Zielitems unter Berücksichtigung des Mappings generiert. Weiters wird vor dem Einfügen des Inhaltes eines Feldes überprüft, ob dieses ein *RichTextField* ist. Ist das der Fall, werden alle Formatierungen (Kursiv, Fett, Unterstrichen, …) übernommen. Andernfalls wird nur *Plaintext* in das Feld geschrieben.

Ist das Item fertig generiert, d.h. es sind alle Inhalte in der gewünschten Form an der gewünschten Position, werden etwaige Bilder an das Item angefügt. Dazu wird der Name des Bildes mithilfe des *issues*-Kommandos aus dem Quellitem ausgelesen. Dieser wird wiederum dazu verwendet, um das eigentliche Bild mithilfe des *addAttachment*-Kommandos an das fertig eingefügte Item anzuhängen.

Ist dieser Vorgang abgeschlossen, wird durch die Funktionalität *CopyWithTrace* (Punkt 6.2.9) überprüft, ob ein Trace gezogen werden soll.

Sind alle diese Prozesse erfolgreich abgeschlossen, wurde das Item von Dokument A in Domäne B nach Dokument C in Domäne D kopiert.

6.2.6. Funktionalität MaxCopyItems

Diese Funktionalität gibt die Anzahl der maximal erlaubten Items an, die auf einmal kopiert werden dürfen und findet vor dem eigentlichen Kopiervorgang Anwendung.

Sie wurde implementiert, indem in der Datei *DomainConfig.conf* eine entsprechende Eigenschaft hinzugefügt wurde. Diese kann der Integrity-Administrator nach Belieben verändern. Der User, welcher *IntegrityDomainCopy* verwendet, hat darauf keinen Zugriff.

```
16  <MaxCopyItems
17  5
18  MaxCopyItems/>
```

Abbildung 43: Funktionalität MaxCopyItems

6.2.7. Funktionalität FieldMappingOne2One

FieldMappingOne2One, und darauf aufbauend *MergeMapping* stellen einen signifikanten Teil der Diplomarbeit dar, da Kopiervorgänge ohne das Mapping nicht durchgeführt werden können.

6.2.7.1 Ablauf

Das Mapping, das der User beim Bearbeiten der Konfiguration eingibt, wird beim Betätigen des *OK-Buttons* auf Unstimmigkeiten überprüft. Falls beispielsweise ein Feld nicht existiert, wird das dem User durch eine Fehlermeldung mitgeteilt. Für genauere Informationen zur Fehlerbehandlung siehe Punkt 5.4.15.

Das Mapping wird dann beim nächsten Kopiervorgang benutzt.

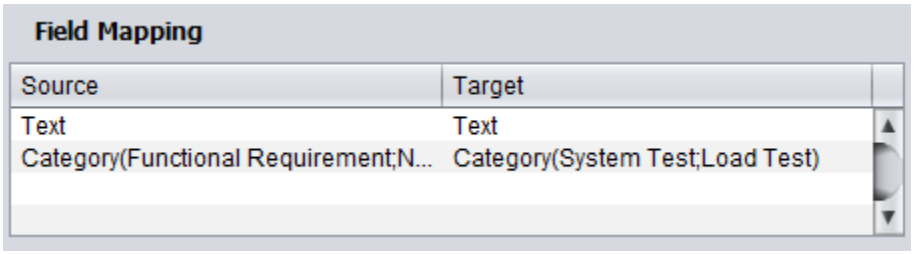


Abbildung 44: Funktionalität FieldMappingOne2One

6.2.8. Funktionalität MergeMapping

MergeMapping ist, wie zuvor bereits beschrieben, eine Art des Mappings. Der einzige Unterschied ist, dass hier mehrere Felder auf eines gemappt werden, beispielsweise *OEM Comments* und *Customer Comments* werden auf das Feld *Comments* gemappt.

6.2.8.1 Ablauf

Realisiert wurde das durch den „+“-Operator, der ein *MergeMapping* signalisiert, wie im darauffolgenden Screenshot verdeutlicht wird. Ist ein Plus im Source-Mapping vorhanden, wird der Inhalt der angegebenen Felder zusammengefasst und in das Zielfeld geschrieben.

Es ist allerdings, aufgrund von Wünschen des Auftraggebers, nicht möglich, den Inhalt eines Feldes auf mehrere Felder aufzuteilen.

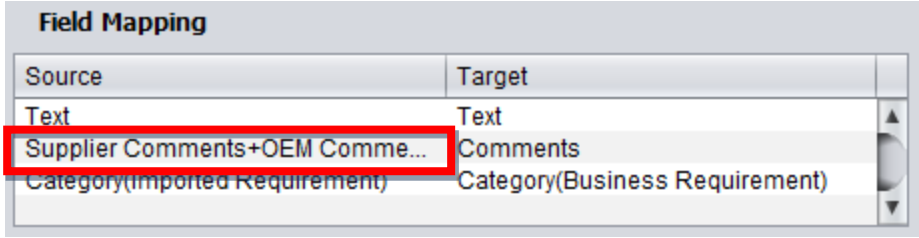


Abbildung 45: Funktionalität MergeMapping

6.2.9. Funktionalität CopyWithTrace

CopyWithTrace wird im Programm durch einen Parameter in der verwendeten Konfiguration realisiert. Dieser Parameter, je nachdem wie der User entscheidet, wird als *true* oder *false* gesetzt.

6.2.9.1 Ablauf

Steht der Parameter auf *true*, wird der Kopiervorgang mit Trace durchgeführt. Realisiert wird dieser Vorgang durch das *editissue*-Kommando, das dem neuen Item einen Verweis auf das Quellitem hinzufügt. Sobald der Vorgang abgeschlossen ist, zieht sich zwischen dem Original-Item und der Kopie eine Verbindung, die eventuelle Änderungen eines Items auf das andere überträgt.

6.2.10. Funktionalität Logging

Hierbei dient die Java-Klasse *Logger* zur Ausgabe der gewünschten Parameter auf der Konsole, sowie zum Speichern derselben in einer .log-Datei.

6.2.11. Funktionalität Preview

Diese Funktionalität findet beim Einfügevorgang Anwendung.

6.2.11.1 Ablauf

Ist der Parameter *Always Show Preview* in der derzeit verwendeten Konfiguration auf *true* gesetzt, wird die Vorschau vor jedem Einfügen angezeigt. Andernfalls kommt die Funktionalität *Timeout* (Punkt 5.4.13) zum Tragen.

Kommt es nun zum Anzeigen der Vorschau, werden die entsprechenden Daten in die *Source*- bzw. *Target*-Tabelle geschrieben. Klickt der Benutzer nun auf *OK*, werden die Daten auf ihre Richtigkeit überprüft. Beispielsweise wird sichergestellt, dass die Kategorie des einzufügenden Items auch in der aktuell ausgewählten Konfiguration gemappt ist. Ist das nicht der Fall, wird eine Fehlermeldung ausgegeben und der Einfügevorgang abgebrochen. Mehr zur Fehlerbehandlung unter Punkt 5.4.15.

6.3. Testing

6.3.1. Testcases

Ein Testcase ist ein elementarer, funktionaler Softwaretest. Er dient zur Überprüfung einer Eigenschaft eines Testobjektes.

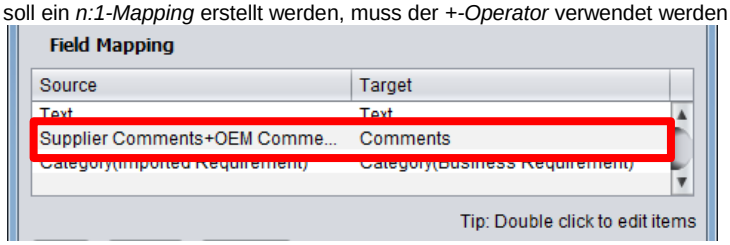
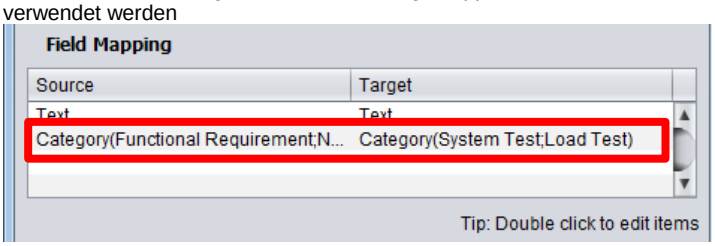
Bestandteile eines Testcases sind:

- die Benennung sowie eine Beschreibung eines Testcases,
- die Vorbedingung, die vor dem Test erfüllt sein muss,
- die Eingabedaten, falls vorhanden, für die Durchführung des Tests,
- die Erwartungswerte des Testcases und
- der Ablauf des Testcases.

Testfälle lassen sich in positive und negative Testfälle unterteilen:

- positive Testfälle:
  - der Testcase wird mit gültigen Vorbedingungen und Eingaben überprüft
    - Create New Config: es wird eine neue Konfiguration angelegt
- negative Testfälle:
  - der Testcase wird mit ungültigen Vorbedingungen und Eingaben überprüft
    - Create New Config – Config already exists: es wird versucht eine neue Konfiguration anzulegen, allerdings ist der Name bereits vergeben [14]

Eigenschaft	Beschreibung/Wert
Testcase / Beschreibung	Gesamter Ablauf von DomainCopy (allgemein)
Vorbedingung	-
Eingangsdaten	-
Erwartungswerte /-funktionen / -daten	• Kopiervorgang wird erfolgreich abgeschlossen • keine Exceptions Ablauf befindet sich auf der nächsten Seite!!

Eigenschaft	Beschreibung/Wert
Ablauf	**Schritt 1:** • bei Custom → Domain Copy Config GUI • das DomainCopy Config-Fenster öffnet sich (Startfenster) • nun kann die gewünschte Konfiguration für den folgenden Kopiervorgang ausgewählt werden • entweder es wird eine bestehende ausgewählt und als aktiv gesetzt mithilfe des Buttons *Set Active Configuration*, eine neue erstellt mittels *New,* eine bestehende bearbeitet mit dem Button *Edit* oder die markierte gelöscht mit *Delete* **Schritt 2 (New):** • mit Klick auf den Button *New* öffnet sich das Fenster *New Configuration* • hier kann der Benutzer die gewünschten Daten (Name, Description, Source, Target, Trace, Always Show Preview) eingeben • bei Klick auf den Button *OK* wird die Konfiguration samt deren Eigenschaften gespeichert und kann nun bearbeitet werden, um das Mapping hinzuzufügen • bei Klick auf den Button *Cancel* wird abgebrochen und die Konfiguration nicht gespeichert **Schritt 2 (Edit):** • bei Klick auf den Button *Edit* öffnet sich das Fenster *Edit Configuration* • hier kann der Benutzer die gewünschten Daten (Name, Description, Always Show Preview, Trace, Mapping) ändern • bei Klick auf den Button *New Field Mapping* öffnet sich das Fenster *New Field Mapping* • bei Klick auf den Button *Delete* wird das ausgewählte *Mapping* gelöscht • bei Klick auf *OK* wird gespeichert und das Fenster geschlossen • bei Klick auf *Apply* wird nur gespeichert, das Fenster bleibt geöffnet • bei *Cancel* wird abgebrochen und das Fenster geschlossen **Schritt 3 (Mapping):** • damit eine Konfiguration funktionsfähig ist, benötigt sie ein *FieldMapping* • dieses wird entweder durch den Button *New Field Mapping* erstellt oder durch einen Doppelklick in eine leere Zeile der *Field Mapping*-Tabelle hinzugefügt • die eingegebenen Daten müssen valide Itemfelder sein, außerdem muss ein richtiges *Category-Mapping* vorhanden sein • soll ein *n:1-Mapping* erstellt werden, muss der *+-Operator* verwendet werden  • sollen mehrere Kategorien auf einmal gemappt werden, muss der *;-Operator* verwendet werden  **Schritt 4 (Kopiervorgang):** • soll ein Item kopiert werden, wird auf dieses rechtsgeklickt → DomainCopy auswählen • sofern das Item in der richtigen Domäne vorhanden ist, wird der Kopiervorgang fehlerfrei durchgeführt, andernfalls wird eine Fehlermeldung ausgegeben und der Kopiervorgang abgebrochen • soll das Item wieder eingefügt werden wird im Target-Dokument rechtsgeklickt → DomainPaste auswählen • sofern das Item in der richtigen Domäne eingefügt wird, wird der Einfügevorgang fehlerfrei durchgeführt, andernfalls wird eine Fehlermeldung ausgegeben und der Einfügevorgang abgebrochen • bei einem fehlerfreien Vorgang wird eine Vorschau angezeigt, die den gesamten Kopiervorgang skizziert, bestätigt der Benutzer mit *OK* wird der Vorgang abgeschlossen, andernfalls wird er abgebrochen
Bewertung	

6.3.2. Mock-Objects

Mock-Objekte kommen in der Softwareentwicklung vor und werden als Platzhalter für echte Objekte verwendet.

6.3.2.1 Einsatz

Mock-Objekte sind sinnvoll, wenn das echte Objekt

- durch Fehler während der Entwicklungsphase nicht beschädigt werden soll,
- ineffizient oder sehr komplex ist,
- nicht existiert oder
- Methoden enthält, die nicht oder schwer rückgängig zu machende Vorgänge durchführen.

Durch Mock-Objekte werden Schnittstellen implementiert, durch welche das testende Objekt, im Falle der Diplomarbeit die Unit-Tests, auf Rückgabewerte der eigentlichen Umgebung zugreifen kann. Diese Objekte liefern keine echten Werte zurück, sondern lediglich welche die für den Testfall passen. Verwendet werden Mock-Objekte, um das Verhalten von voll ausprogrammierten Objekten nachzuahmen. [12]

```
@Override
public String getAssociatedType(String mainType) {

    switch(mainType){
        case "Input": return "Input Document";
        case "Requirement": return "Requirement Document";
        case "Specification": return "Specification Document";
        case "Test": return "Test Case";
    }

    return null;
}
```

Abbildung 46: Mock-Object – getAssociatedType

```
public interface IntegrityActionMock {
    public String createTargetNode(String targetType, String parentID, ...);
    public boolean editField(String targetNodeID, String FieldEdit);
    public boolean addRelationship2Src(String targetNodeID, String srcIssueID, ...);
    public String getIssueType(String ID);
    public String getAssociatedType(String mainType);
    public String getFieldContent(String issueID, String fieldName, ...);
    public Boolean isRichContentField(String fieldname);
    public boolean fieldExists(String fieldname);
    public void releaseSession(String location);
}
```

Abbildung 47: Mock-Interface

6.3.3. Unit-Tests

Ein Unit-Test wird in der Softwareentwicklung angewandt, um funktionale Einzelteile von Programmen zu testen, also auf ihre korrekte Funktionalität prüfen.

6.3.3.1 Testtechniken

Es gibt mehrere Testarten die für Unit-Tests zutreffen, sowie in der Diplomarbeit verwendet wurden:

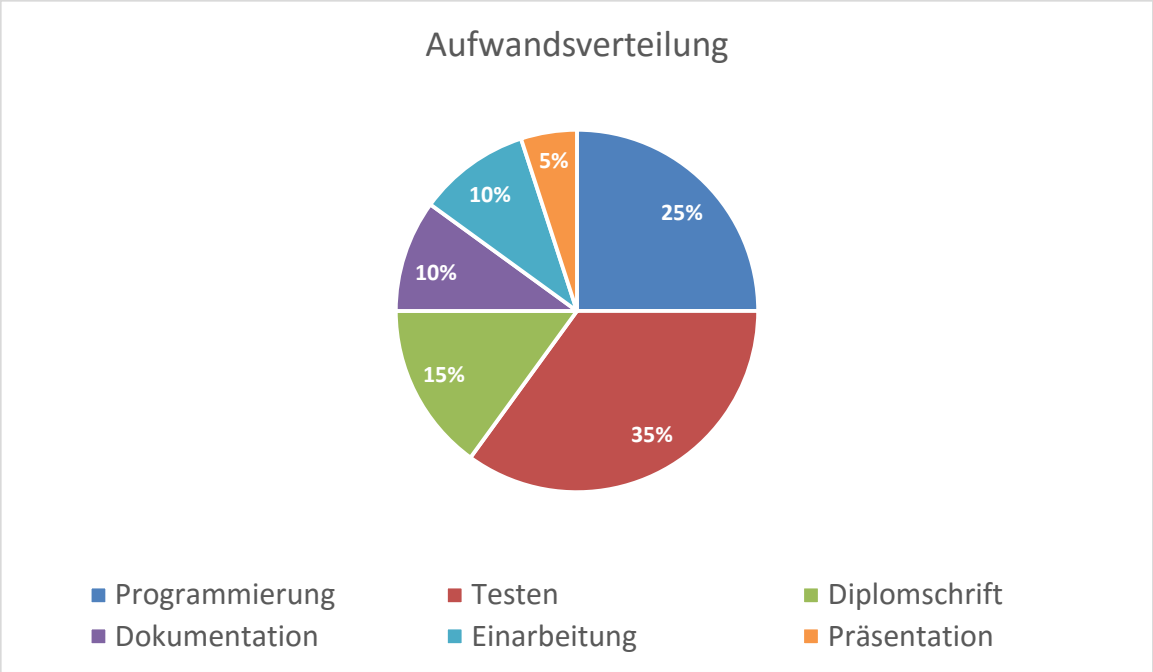
- ein Entwicklertest
  - es werden Testversionen an die Personen, die zukünftig mit dem Programm arbeiten, verteilt, um Ergebnisse zu protokollieren und evtl. Fehler zu identifizieren.
- ein White-Box-Test
  - die zu testenden Methoden oder Programmteile werden aus dem gesamten Quellcode herausgehoben, um effizientere Testvorgänge zu ermöglichen. [13]

```
@Test
public void getAssociatedTypeTest() {
    String test = this.iaml.getAssociatedType("Input");
    assertTrue(test.equals("Input Document"));
}
```

Abbildung 48: Unit-Test - getAssociatedTypeTest

7. Aufwandsverteilung

Die folgende Grafik zeigt die Aufwandsverteilung im Zuge unserer Diplomarbeit. Danach werden die verschiedenen Punkte (Programmierung, Testen, Diplomschrift, Dokumentation, Einarbeitung und Präsentation) genauer beleuchtet.



7.1. Programmierung

Für die Programmierung der eigentlichen Funktionalitäten wurde ein großer Teil der Gesamtarbeitszeit aufgewendet. Hierbei wurde der Workflow immer wieder durch unerwartete Probleme unterbrochen, von denen einige einen Workaround erforderten.

Über die Funktionalitäten der Anwendung hinaus wurde außerdem ein Mock-Object programmiert, das die Anbindung des Programms an die API simulierte. Treu zu seiner Natur als Mock-Object lieferte dieses Objekt keine realen Daten, sondern festgelegte Werte, die von dem jeweiligen Testfall abhängten. Darauf aufbauend wurden einige Unit-Tests erstellt.

7.2. Testen

Das Testen der Anwendung war ein weiterer großer Teil unserer Diplomarbeit. Neben den herkömmlichen Tests und Debugs während des Entwicklungsprozesses, erstellten wir auch einige Unit-Tests. Mit deren Hilfe wurden, im Nachhinein gesehen, einiges an Zeit und Haareraufen eingespart.

7.3. Diplomschrift

Im Vergleich zur Programmierung und dem Testen wurde für das Schreiben der Diplomarbeit nur wenig Zeit benötigt. Da dies aber trotzdem einen essentiellen Teil dieses Projekts darstellte, wurden Sätze immer wieder umgeschrieben und Rechtschreibfehler ausgebessert, bis wir mit dem Endergebnis zufrieden waren.

7.4. Dokumentation

Der Dokumentation unseres Arbeitsfortschritts wurde mehr Zeit gewidmet, als ursprünglich geplant war. Der Hauptgrund für die Wichtigkeit dieser Dokumentationen war die Wartbarkeit unserer Anwendung nach dem Projektabschluss.

7.5. Einarbeitung

Da unsere Arbeit auf das Programm „Integrity“ aufbaut, mussten wir zuerst eine Schulung absolvieren, um uns die Funktionalitäten und den Zweck dieses Programms näherzubringen. Weiters mussten wir uns in den bereits vorhandenen Prototypen von „DomainCopy“ einarbeiten, um diesen sinnvoll um die gewünschten Funktionalitäten erweitern zu können.

7.6. Präsentation

Aufgrund der P@bs-Veranstaltung wurde im Zuge der Diplomarbeit eine Präsentation gestartet, die den einfachen Ablauf sowie den Aufbau von *Integrity-DomainCopy* beschreibt.

8. Begriffe

8.1. PTC Integrity

Integrity ist ein Application Lifecycle Management System, entwickelt von *PTC* und veröffentlicht in 2001. Das Programm basiert auf Java und unterstützt eigene, vom User programmierte Erweiterungen, um das Programm noch benutzerfreundlicher zu gestalten. Im Zuge unserer Arbeit wurde *Integrity* hauptsächlich zur Verwaltung von mehreren Dokumenten in unterschiedlichen Domänen verwendet, das ist allerdings nur ein sehr kleiner Teil der Funktionen des Programms.

8.2. Application Lifecycle Management System

Der Begiff *Application Lifecycle Management System* beschreibt ein Programm, das den kompletten Entwicklungsprozess sowie die weitere Betreuung nach der Fertigstellung einer Applikation verwaltet. [16]

8.3. Domain

Domänen sind Gruppen innerhalb eines Workflows. Diese Gruppen sind:

- *Input*: Lastenhefte
- *Requirement*: Pflichtenhefte
- *Specification*: Beschreibung eines Designs von elektromechanischen Systemen oder von Software
- *Test*: Test Spezifikation

In diesen Gruppen befinden sich gleichartige Dokumente wie z.B.: Input-, Requirementdokumente, …

8.4. Document

Ein *Document* besteht aus mehreren *Items*, die wiederum den Inhalt des *Documents* darstellen. Beispielsweise besteht ein Pflichtenheft aus mehreren *Items*, in denen wiederum der Text oder Bilder des Dokuments gespeichert werden.

8.5. Item

Ein *Item* kann mit einer Zeile in einer SQL-Datenbank verglichen werden. Es besteht aus mehreren Feldern wie beispielsweise *Text, Comments* oder *Category*.

8.6. Itemfelder

In diesen Feldern wird der jeweilige Inhalt der *Items* gespeichert.

8.7. Workflow

Ein Workflow beschreibt eine bestimmte Abfolge von Aktivitäten oder Prozessen in einem System. [15]

8.8. ECSDomainCopy

ECSDomainCopy beschreibt den Namen der Diplomarbeit, also *Integrity-DomainCopy*, in der Firma Magna Powertrain.

8.9. Trace

Der Begriff *Trace* wird in *Integrity* für das Wort Beziehung oder Verknüpfung verwendet. Falls Änderungen, die bei einem *Item* durchgeführt werden, bei einem anderen *Item* ebenfalls erledigt werden sollen, wird ein *Trace* benötigt.

8.10. Field-Mapping

Damit der Kopiervorgang erfolgreich durchgeführt werden kann, müssen die Itemfelder der verschiedenen Domänen einander zugeordnet werden, da sich diese von Domäne zu Domäne unterscheiden.

8.11. GUI (Graphical User Interface)

Die grafische Benutzeroberfläche ist das Design des Fensters, mit dem der Benutzer arbeitet.

8.12. Integration

Eine Integration ist eine zusätzliche, von einem Entwickler ausprogrammierte Funktion, die von *PTC Integrity* nicht ausprogrammiert worden ist.

9. Quellverzeichnis

9.1. Dokumente und PDFs

1. AdminCLIReference_Integrity_10_4.pdf
2. Application_Lifecycle_Management_Solution_Guide_10_1.pdf
3. ClientGettingStartedGuide_Integrity_10_4.pdf
4. ClientUserGuide_Integrity_10_4.pdf
5. IntegrationsBuilderGuide_Integrity_10_4.pdf
6. IntegrityALMSolutionOverview.pptx
7. WorkflowCLIReference_Integrity_10_4.pdf
8. IntegrityJava API 4.12.4301
9. CheatSheet_ReleaseAndPublishIntegrations.vsd

9.2. Webseiten

9.2.1. Wikipedia

10. PTC Integrity:
    https://en.wikipedia.org/wiki/PTC_Integrity
11. NetBeans IDE:
    https://de.wikipedia.org/wiki/NetBeans_IDE
12. TortoiseHG:
    https://de.wikipedia.org/wiki/TortoiseHg
13. Mock-Objekt:
    https://de.wikipedia.org/wiki/Mock-Objekt
    https://de.wikipedia.org/wiki/Mocking_Framework
14. Unit-Tests:
    https://de.wikipedia.org/wiki/Modultest
15. Testcases:
    https://de.wikipedia.org/wiki/Testfall
16. Workflow
    https://de.wikipedia.org/wiki/Arbeitsablauf
17. Application Lifecycle Management System
    https://de.wikipedia.org/wiki/Application-Management

9.2.2. Diverse sonstige Seiten

18. StackOverflow:
http://stackoverflow.com/
19. CodeRanch:
http://www.coderanch.com/forums
20. PTC Integrity:
http://de.ptc.com/product/integrity
21. Magna Powertrain (ECS):
https://www.ecs.steyr.com/
22. NetBeans:
https://netbeans.org/
23. TortoiseHg:
http://tortoisehg.bitbucket.org/

10. Abbildungsverzeichnis

11. Auszug Projekthandbuch

11.1. Meilensteine

12. Resümee und persönliche Erfahrungen

12.1. Resümee Hannes Windischhofer

Durch die Diplomarbeit konnte ich viele Erfahrungen sammeln, angefangen vom Arbeiten mit Technologien, von denen man bis dahin noch nichts gehört hat, bis zum Entwickeln und dem darauffolgendem Publizieren einer Software innerhalb einer großen, branchenführenden Firma.

Weiters ist mir im Verlauf der Diplomarbeit klar geworden, wie viel es, abgesehen vom Programmieren, an einem solchen Projekt zu tun gibt. Pflichtenheft und andere Dokumente, genaue Planung des weiteren Vorgehens oder regelmäßige Treffen mit dem Auftraggeber, nur um ein paar Beispiele zu nennen.

Die größte Herausforderung des Projekts war aber ohne Zweifel die Tatsache, dass nur in der Firma an dem Programm gearbeitet werden konnte. Grund dafür war, dass unser Programm auf Werte, Variablen und Kommandos zugreifen musste, die wegen Integrity nur in der Firma zur Verfügung standen. Gelöst haben wir das Problem durch eine gute Zeiteinteilung sowie durch die Verwendung von Unit-Tests, die ich mitunter sehr zu schätzen gelernt habe.

Alles in allem bin ich aber durchaus froh, dass ich an dieser Diplomarbeit mitgearbeitet habe. Dank dieser habe ich etwa neun Monate Firmenluft schnuppern und Erfahrungen sammeln dürfen, die im späteren Berufsleben durchaus nützlich werden können.

12.2. Resümee Christoph Seiler

Diese Diplomarbeit war für mich reich an Erfahrung und eine große Herausforderung. Ich lernte wie wichtig es ist eine gute Zeiteinteilung zu haben, vor allem weil wir lediglich in der Firma an unserem Programm arbeiten konnten, sowie diese strikt einzuhalten und dass man eine solche Arbeit nicht unterschätzen darf.

Da wir zuhause keinen Integrity-Client hatten, war es bei unserer Diplomarbeit besonders wichtig, viel Kontakt zum Auftraggeber zu haben, deshalb war die Zeiteinteilung auch nicht immer so leicht, da dieser ein bis zwei Mal wöchentlich Home-Office betrieben hatte.

Ich bin froh, dass ich die Möglichkeit hatte diese Diplomarbeit in der Firma Magna Powertrain zu absolvieren, da ich hoffe, dass mir die gemachten Erfahrungen im späteren Berufsleben helfen werden und dass ich später einmal in dieser Firma arbeiten kann. Außerdem konnte ich so sehen, wie viele Arbeitsstunden in so einem, für unsere Verhältnisse, größeren Projekt stecken, was ich bisher unterschätzt hatte.

13. Verteilung der Aufgabengebiete

13.1. Hannes Windischhofer

- Kurzbeschreibung, Punkt 1
- Abstract, Punkt 2
- Entstehung und Planung, Punkt 5
  - Release einer Integrity ECS Integration
  - Hinzufügen einer Integration in Integrity
- Umsetzung, Punkt 6
  - Allgemeines
  - Funktionalitäten
- Aufwandsverteilung, Punkt 7
- Begriffe, Punkt 8
- Auszug Projekthandbuch, Punkt 11
- Resümee Hannes Windischhofer, Punkt 12.1
- Verteilung der Aufgabengebiete, Punkt 13
- Anhang, Punkt 14
  - Ablaufdiagramm Gesamtsystem
  - Verwendete Kommandos und Klassen

13.2. Christoph Seiler

- Kurzbeschreibung, Punkt 1
- Stakeholder im Projekt, Punkt 3
- Beschreibung des Umfelds, Punkt 4
- Entstehung und Planung, Punkt 5
  - Thema und Aufgabenstellung
  - Zielsetzung
  - Vorgehensweise
  - Funktionsumfang
  - Ressourcenplanung
- Umsetzung, Punkt 6
  - Testing
- Begriffe, Punkt 8
- Quellverzeichnis, Punkt 0
- Abbildungsverzeichnis, Punkt 10
- Resümee Christoph Seiler, Punkt 12.2
- Verteilung der Aufgabengebiete, Punkt 13
- Anhang, Punkt 14
  - Das Team

14. Anhang

14.1. Das Team

14.1.1. Hannes Windischhofer

Persönliche Daten	
Geburtsdaten:	Linz, 04.Mai 1997
Staatsbürgerschaft:	Österreich
Religionsbekenntnis:	röm.-kath.
Familienstand:	ledig
Eltern:	Manfred Windischhofer, Beamter Erika Windischhofer, Ärztliche Assistentin
Geschwister:	Philipp Windischhofer, Student

Abbildung 49:
Hannes Windischhofer

Schulbildung	
4 Jahre	Volksschule in Grein
4 Jahre	Hauptschule für IT Grein
seit September 2011	HTL für Informatik Perg

Erfahrungen		
Praktikum	Juli 2013	Straßenmeisterei Grein
Praktikum	Juli 2014	Gasokol GmbH in Saxen

Kenntnisse und Fähigkeiten	
Sprachen:	• Englisch (Schulkenntnisse)
EDV-Kenntnisse:	• Sehr gute Office-Kenntnisse (MS Office 2010/2013) • Programmieren in Java, C, C#, Python • Bildbearbeitung (GIMP) • Sound- und Videobearbeitung • Webdesign (PHP, JavaScript, HTML, CSS) • Gute Hardware-Kenntnisse • 10-Finger-System(250 Anschläge/min)
Sonstiges:	• Erste-Hilfe-Kurs • ECDL (Europäischer Computerführerschein) • Börni in Bronze und Silbermedaille vom Computer Contest 2010/11 • Führerschein der Klasse B

14.1.2. Christoph Seiler

Persönliche Daten	
Geburtsdaten:	Amstetten, 10. Jänner 1997
Staatsbürgerschaft:	Österreich
Religionsbekenntnis:	röm.-kath.
Familienstand:	ledig
Eltern:	Ronald Marton, Bankangestellter Sabine Seiler-Stubauer, Bankangestellte
Geschwister:	Pia Stubauer, Schülerin Paul Stubauer, Baby

Abbildung 50: Christoph Seiler

Schulbildung	
4 Jahre	Volksschule in Langenhart, St. Valentin
4 Jahre	Hauptschule in Langenhart, St. Valentin
seit Sept. 2011	HTL für Informatik in Perg

Erfahrungen		
Praktikum	Juli 2013	G. Stubauer Werbe -& PR-Agentur in St. Valentin
Praktikum	Juli 2014	Magna Powertrain in St. Valentin

Kenntnisse und Fähigkeiten	
Sprachen:	• Englisch (Schulkenntnisse)
EDV-Kenntnisse:	• gute Office-Kenntnisse (MS Office 2010) • Programmieren in Java, C und C# • Bildbearbeitung (GIMP) • Sound- und Videobearbeitung • Webdesign • Hardware-Grundkenntnisse • 10-Finger-System
Sonstiges:	• ECDL (Europäischer Computerführerschein) • Führerschein der Klasse B

14.2. Ablaufdiagramm Gesamtsystem

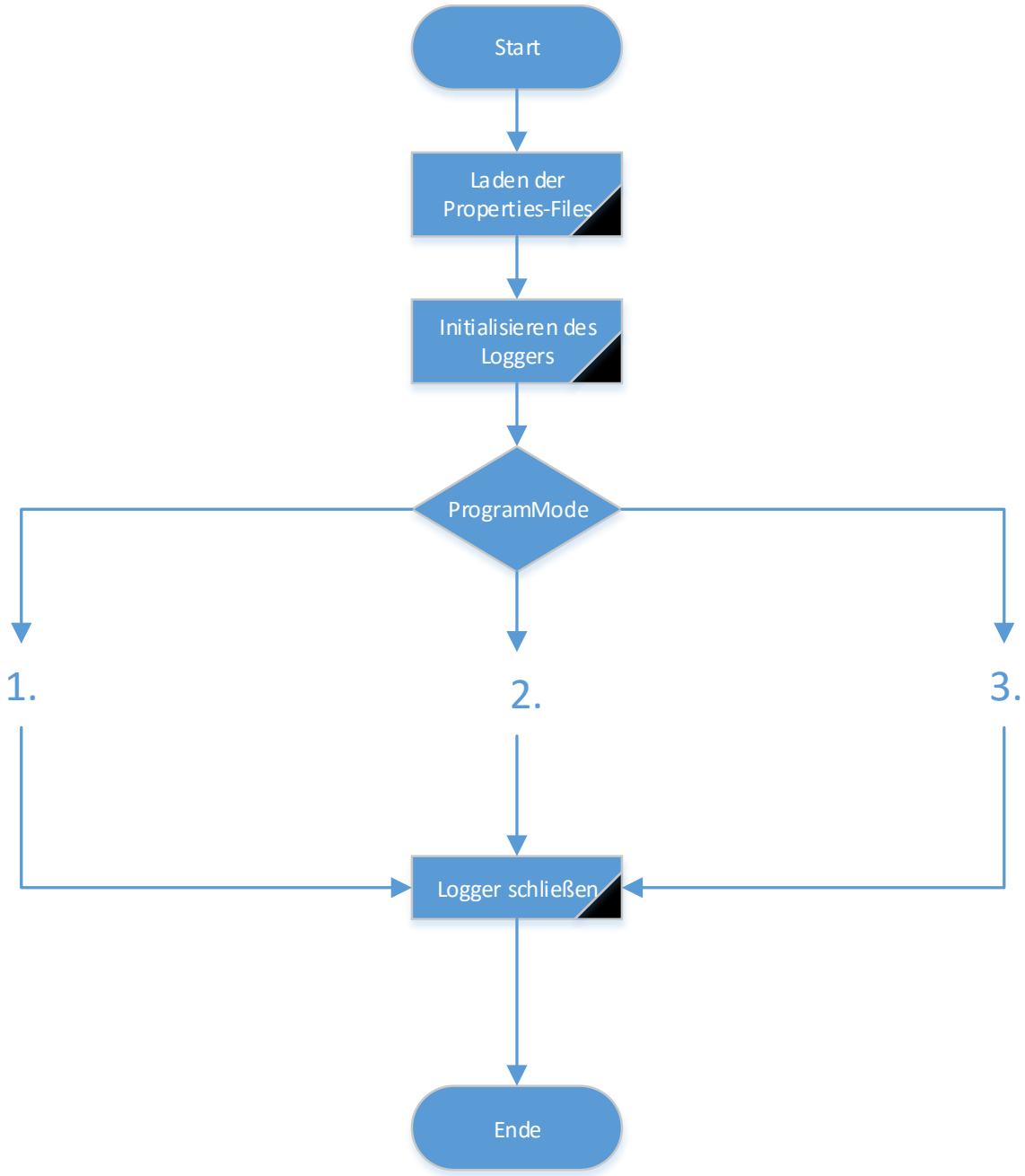


Abbildung 51: Ablaufdiagramm 1

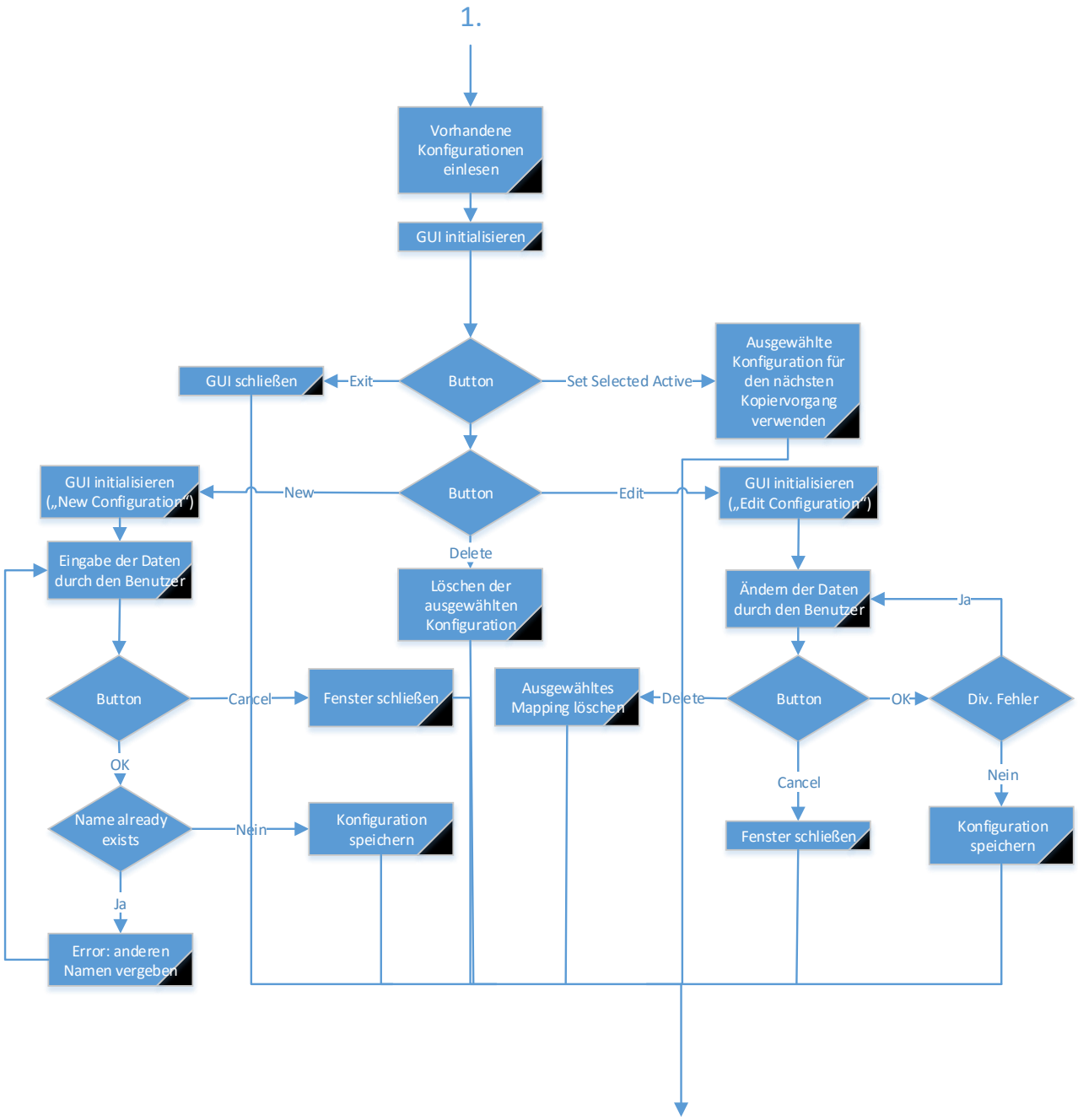


Abbildung 52: Ablaufdiagramm 2

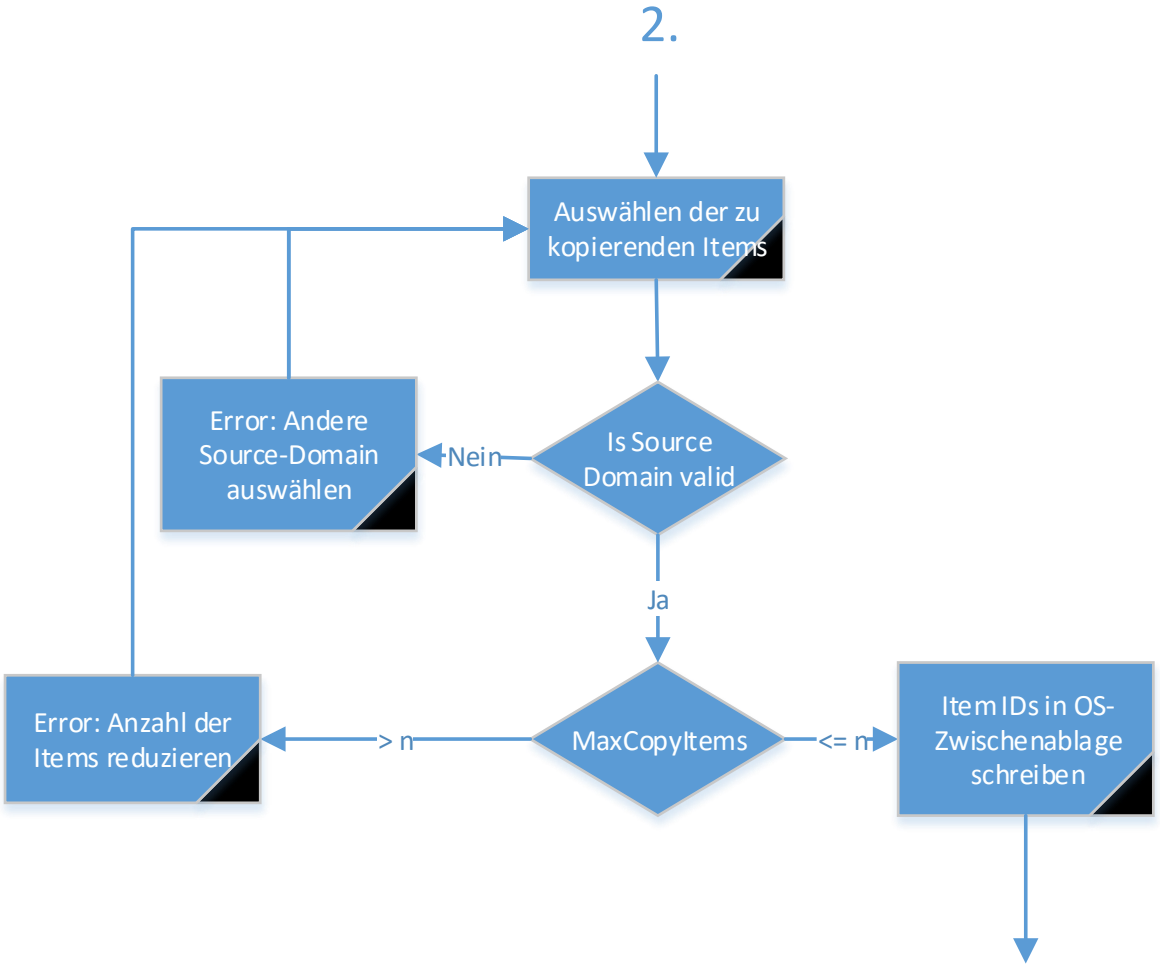


Abbildung 53: Ablaufdiagramm 3

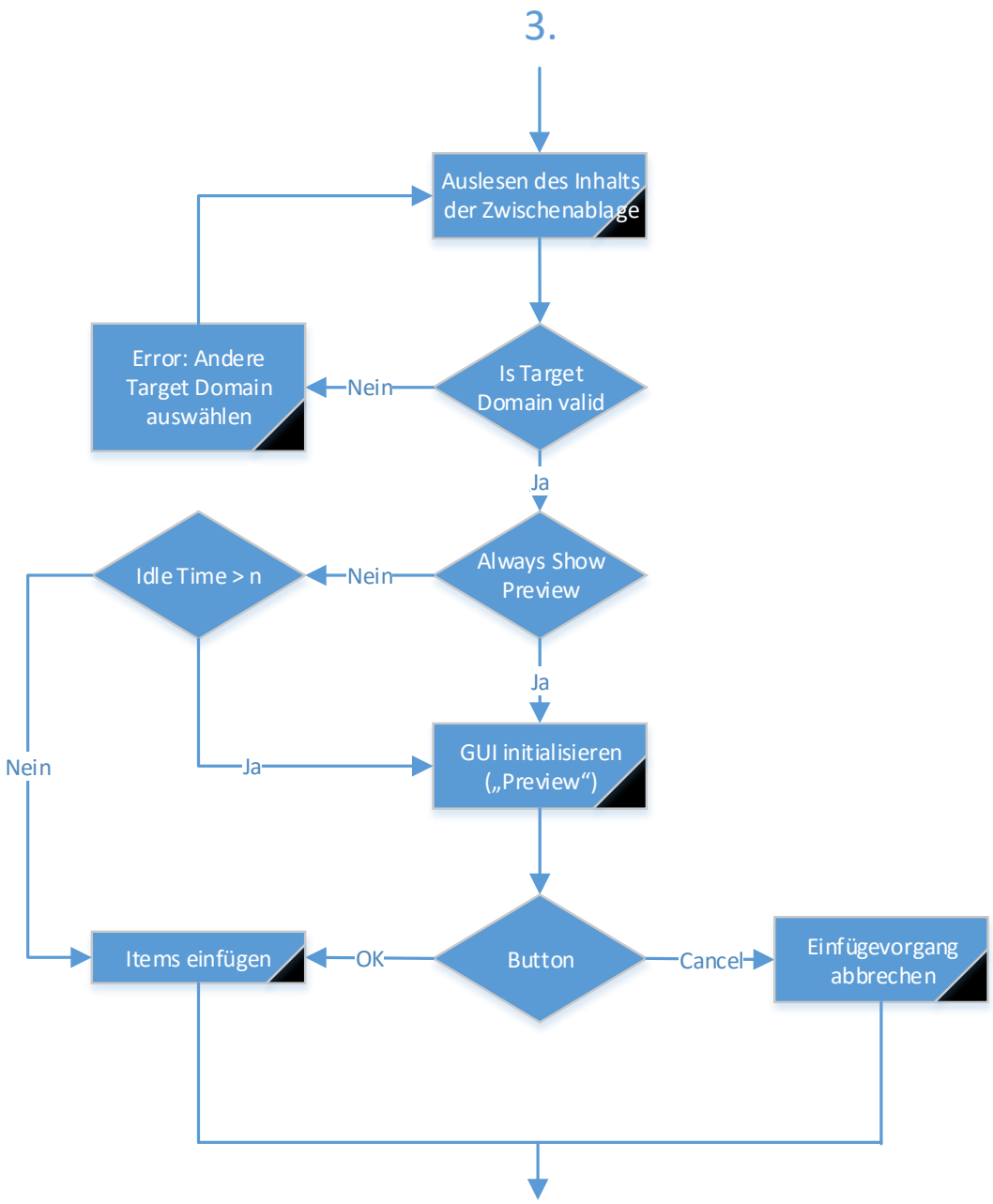


Abbildung 54: Ablaufdiagramm 4

14.3. Verwendete Kommandos und Klassen

14.3.1. Kommandos

14.3.1.1 im viewissue

Syntax

```
im viewissue [--asOf=[<date>|label:<label>] [--[no]showAttachments]
[--[no]showAttachmentDetails] [--[no]showAnnotations] [--[no]showBranches]
[--[no]showLabels] [--[no]showChangePackages] [--[no]showHistory]
[--[no]showHistoryAscending] [--[no]showRelationships] [--[no]showWorkflow]
[--[no]showTimeEntries] [--[no]showHistoryWithComputedField] [--[no]showRichContent]
[--[no]showXHTML] [--[no]showTestResults] [--[no]substituteParams]
[--[no]showSourceTraceDetails] [--[no]showSourceLinkDetails] [--height=value]
[--width=value] [-x value] [-y value] [--hostname=value] [--port=value] [--password=value]
[--user=value] [(-?|--usage)] [(-g|--gui)] [(-F value|--selectionFile=value)] [--quiet]
[--settingsUI=[gui|default]] [--status=[none|gui|default]] [(-N|--no)] [(-Y|--yes)]
[--[no]batch] [--cwd=value] [--forceConfirm=[yes|no]] issue id...
```

Relevante Parameter

- --[no] showAttachments
  Gibt an, ob angehängte Dateien (z.B.: Bilder) angezeigt werden sollen.
- --[no] showAttachmentDetails
  Gibt an, ob Details der angehängten Dateien (z.B.: Name des Users, der die Datei hinzugefügt hat, Datum und Zeit des Hinzufügens der Datei, Name der Datei, Größe der Datei) angezeigt werden sollen.
- *issue id*
  Beschreibt die ID des Items, das der User genauer betrachten will [7]

Verwendung

Dieses Kommando wird bei der Methode *getFieldContent(String issueID, String fieldName)* verwendet, um den Inhalt des markierten Items auszulesen.

```
Command myCmd = new Command(Command.IM, "viewissue");
myCmd.addSelection(issueID);
Response myRes = myCmdRunner.execute(myCmd);
WorkItemIterator wii = myRes.getWorkItems();
fieldContent = wii.next().getField(fieldName).getValueAsString();
return fieldContent;
```

Abbildung 55: im viewissue

Dem Kommando wird lediglich die ID des Items, das betrachtet werden will, als Parameter mitgegeben. Der Rückgabewert des Kommandos enthält die Inhalte aller Felder des Items. Nun wird noch der Inhalt des gewünschten Feldes, identifiziert durch den Parameter *fieldName*, ausgelesen und für weitere Verarbeitung zurückgegeben.

14.3.1.2 im createcontent

Syntax

```
im createcontent [--addRelationships=value] [--addSourceTrace=value]
[--addSourceLink=value] [--addAttachments=value]
[--type=type] [--field=value] [--richContentField=value]
[--insertLocation=[first|last|before:ID|after:ID|[0,...]]] [--quiet]
[--user=name] [--hostname=server] [--password=password]
[--port=number] [(-?|--usage)] [(-F file|--selectionFile=file)] [(-N|--no)]
[(-Y|--yes)] [--[no]batch] [--cwd=directory] [--forceConfirm=[yes|no]] [-g|--gui]
[--settingsUI=[gui|default]] [--status=[none|gui|default]] [--parentID=value]
```

Relevante Parameter

- --field=value
  Setzt ein Feld und dessen Wert für das Item.
  z.B.: --field=Severity=Critical → für das Feld *Severity* wird der Wert *Critical* spezifiziert
- --addAttachment=value
  Fügt eine Datei dem Item hinzu.
  z.B.: --addAttachment=Severity, ./Folder/FileToAdd.jpg → dem Feld *Severity* wird das Bild *FileToAdd.jpg* hinzugefügt. Absolute sowie relative Pfade sind möglich.
- --parentID=value
  Gibt die ID des Dokuments, in dem das Item erstellt werden soll, an.
- --type=type
  Gibt an, welchem Itemtyp das Item angehören soll. Die Itemtypen sind von *Integrity* vorgegeben, bzw. werden vom Administrator erstellt. [7]

Verwendung

Dieses Kommando wird im Zuge der Methode *createTargetNode* verwendet, um ein neues Item anzulegen.

```
Command myCmd = new Command("im", "createcontent");
myCmd.addOption(new Option("parentID", parentID));
myCmd.addOption(new Option("insertLocation", "after:" + targetID));

for(int i=0; i<mapping.length;i++){
    myCmd.addOption(new Option("field", targetFields[i] + "=" + fieldContent));
}
Response myRes = myCmdRunner.execute(myCmd);
Result result = myRes.getResult();
targetNodeID = result.getField("resultant").getItem().getId();
return targetNodeID;
```

Abbildung 56: im createcontent

Dem Kommando werden folgende Parameter hinzugefügt:

- Die ID des Dokuments, in dem das Item angelegt werden soll
- Der Ort, an dem das Item angelegt werden soll. In diesem Fall nach dem Item mit der ID *targetID*

- Das Feld und der entsprechende Inhalt desselben. Dieser Parameter wird durch eine Schleife solange mit unterschiedlichen Feldern und Inhalten hinzugefügt, bis alle gemappten Felder angelegt wurden.

Anschließend wird die ID des eben erstellten Items zur weiteren Bearbeitung bzw. zur Fehlerbehandlung zurückgegeben.

14.3.1.3 im issues

Syntax

```
im issues [--[no]applyDisplayPattern] [--asOf=[<date>|label:<label>]
[--fields=field[:width[:rich|plain]],field[:width[:rich|plain]],...] [--fieldsDelim=value]
[--[no]sortAscending] [--sortField=field] [--fieldFilter=field=[value,value...]]
[--[no]inlineEditMode] [--[no]ApplyDisplayPattern] [--[no]showTallRows]
[--structureFieldIconDisplayField=field] ][--structureFieldDisplayFormat=value]
[--focusIssueID=value] [--expandLevel=value] [--traverseFields=field,field,...]
[--[no]substituteParams] [--[no]showXHTML][--queryDefinition=query] [--hostname=value]
[--port=value] [--password=value] [--user=value] [--query=[user:]query] [(-?|--usage)] [(-g|--gui)]
[--height=value] [--width=value] [-x value] [-y value] [(-F value|--selectionFile=value)]
[--quiet] [--settingsUI=[gui|default]] [--status=[none|gui|default]] [(-N|--no)] [(-Y|--yes)]
[--[no]batch] [--cwd=value] [--forceConfirm=[yes|no]] [--hostname=value] issue id
```

Relevante Parameter

- --fields = field
  Gibt an, welche Felder eines Items angezeigt werden sollen.
  Falls eines der Felder ein *RichContent*-Feld ist, kann mit *::rich* oder *::plain* angegeben werden, ob der Inhalt der Felder formatiert oder im Plaintext angezeigt werden. [7]

Verwendung

Verwendet wird dieses Kommando in der Methode *getIssueType*, um den Typ eines Items auslesen zu können.

```
Command myCmd = new Command(Command.IM, "issues");
myCmd.addOption(new Option("fields", "Type"));
myCmd.addSelection(ID);

Response myRes = myCmdRunner.execute(myCmd);
WorkItemIterator wii = myRes.getWorkItems();
issueType = wii.next().getField("Type").getItem().getId();
return issueType;
```

Abbildung 57: im issues

Dem Kommando wird die Option hinzugefügt, dass nur das Feld *Type* in der Antwort des Kommandos zurückgegeben wird. Weiters wird noch die ID des gesuchten Items angefügt.

Nach dem Ausführen des Kommandos wird der Inhalt des *Type*-Felds ausgelesen und zur weiteren Bearbeitung weitergegeben.

14.3.1.4 im editissue

Syntax

```
im editissue [--[no]showWorkflow] [--[no]batchEdit] [--query=[user:]query]
[--removeAttachment=value] [--removeRelationships=value] [--addAttachment=value]
[--updateAttachment=value] [--addRelationships=value][--addSourceTrace=value]
[--addSourceLink=value] [--removeSourceTrace=value] [--removeSourceLink=value] [--field=value]
[--richContentField=value] [--hostname=value] [--port=value] [--password=value]
[--user=value] [(-?|--usage)] [(-g|--gui)][(-F value|--selectionFile=value)] [--quiet]
[--settingsUI=[gui|default]] [--status=[none|gui|default]] [(-N|--no)] [(-Y|--yes)]
[--[no]batch] [--cwd=value] [--forceConfirm=[yes|no]] issue id...
```

Relevante Parameter

- --addAttachment = value
  Fügt eine Datei dem Item hinzu.
  z.B.: --addAttachment="field=Attachments, path=./Folder/FileToAdd.jpg" → dem Feld *Attachments* wird das Bild *FileToAdd.jpg* hinzugefügt. Absolute sowie relative Pfade sind möglich.
- --removeAttachment = value
  Entfernt den Anhang eines Feldes. Falls kein Feld definiert wird, wird das Standardanhangfeld benutzt.
  z.B.: --removeAttachment = "field=Attachments, name=AttachmentName" [7]

Verwendung

Verwendet wird dieses Kommando in der Methode *addRelationship2Src*, um einem Item einen *Trace* hinzuzufügen.

```
Command myCmd = new Command(Command.IM, "editissue");
myCmd.addOption(new Option("addRelationships", relationType + ":" + srcIssueID));
myCmd.addSelection(targetNodeID);
Response myRes = myCmdRunner.execute(myCmd);
```

Abbildung 58: im editissue

Dem Kommando werden der Typ der Beziehung, die ID des Quellitems sowie die ID des Zielitems angefügt. Wurde das Kommando erfolgreich durchgeführt, wird eine Beziehung zwischen den beiden Items, identifiziert durch die Quell- sowie die Ziel-ID, erstellt.

14.3.1.5 im viewtype

Syntax

```
im viewtype [--height=value] [--width=value] [-x value] [-y value]
[--[no]showHistory] [--[no]showReferences][--[no]showProperties] [--quiet]
[--user=name] [--hostname=server] [--password=password] [--port=number]
[(-?|--usage)] [(-F file|--selectionFile=file)] [(-N|--no)] [(-Y|--yes)]
[--[no]batch] [--cwd=directory][--forceConfirm=[yes|no]] [-g|--gui]
[--settingsUI=[gui|default]] [--status=[none|gui|default]] type...
```

Relevante Parameter

- *type*
  Beschreibt den Namen eines Items der Domäne, deren Name angezeigt werden soll [1]

Verwendung

Dieses Kommando wird in der Methode *getAssociatedType* verwendet, um die Domäne des Zieldokuments herauszufinden. Diese wird dann beim Einfügevorgang eines Items benötigt.

```
Command myCmd = new Command("im", "viewtype");
myCmd.addSelection(mainType);
Response myRes = myCmdRunner.execute(myCmd);
WorkItemIterator wii = myRes.getWorkItems();
typen= wii.next().getField("associatedType").getItem().getId();
return type;
```

Abbildung 59: im viewtype

Dem Kommando wird der Name eines Dokuments der Domäne, in der das neue Item erstellt wird, als Parameter angehängt. Wird beispielsweise *Input Document* angegeben, wird das Kommando *Input*, also den Namen der *Input*-Domäne zurückgeben.

Um den Namen der Domäne aus der Antwort des Kommandos herauszubekommen, wird der Inhalt des *associatedType*-Felds ausgelesen.

14.3.1.6 im types

Syntax

```
im types [--fields=field1[:width1],field2[:width2]... ] [--fieldsDelim=value]
[--height=value] [--width=value] [-x value] [-y value][--user=name]
[--hostname=server] [--password=password] [--port=number] [(-?|--usage)]
[(-F file|--selectionFile=file)][(-N|--no)] [(-Y|--yes)] [--[no]batch]
[--cwd=directory] [--forceConfirm=[yes|no]] [(-F file|--selectionFile=file)]
[--[no]asAdmin] [-g|--gui] [--settingsUI=[gui|default]] [--quiet]
[--status=[none|gui|default]] type...
```

Relevante Parameter

- --fields=*field*
  field kann etliche Werte annehmen, im Programm wurde allerdings nur mit *field=visibleFields* gearbeitet. So gibt das Programm nur die Namen der sichtbaren, verwendbaren Felder einer Domäne zurück.
- *type*
  Gibt an, von welcher Domäne ausgegangen werden soll. Ist *type* beispielsweise der Wert *Input* zugewiesen, werden alle Felder der *Input*-Domain zurückgegeben. [1]

Verwendung

Verwendet wird dieses Kommando in der *fieldExists*-Methode, um die existierenden Felder einer Domäne auszulesen und diese mit einem Feld zu vergleichen. So kann festgestellt werden, ob dieses Feld in einer Domäne existiert.

```
Command myCmd = new Command(Command.IM, "types");
myCmd.addOption(new Option("fields","visibleFields"));
myCmd.addSelection(domain);

Response myRes = myCmdRunner.execute(myCmd);
WorkItemIterator wii = myRes.getWorkItems();
String cmdResult =  ((Field)(wii.next().getFields().next())).getValueAsString();
```

Abbildung 60: im types

Dem Kommando werden die Option *–fields=visibleFields*, sowie der Name der Domäne angehängt. Anschließend wird das Kommando ausgeführt und alle existierenden und sichtbaren Felder der Domäne ausgelesen.

Danach wird noch das als Methodenparameter mitgegebene Feld mit der Masse an Feldern aus dem Kommando verglichen. Je nach dem Ergebnis wird *true* oder *false* zurückgegeben.

14.3.1.7 im viewfield

Syntax

```
im viewfield [--overrideForType=type] [--height=value] [--width=value] [-x value]
[-y value] [--[no]showHistory][--[no]showReferences] [--quiet] [--user=name]
[--hostname=server] [--password=password] [--port=number][(-?|--usage)]
[(-F file|--selectionFile=file)] [(-N|--no)] [(-Y|--yes)] [--[no]batch]
[--cwd=directory][--forceConfirm=[yes|no]] [-g|--gui] [--settingsUI=[gui|default]]
[--status=[none|gui|default]] field...
```

Relevante Parameter

- *field*
  spezifiziert den Namen des Feldes, das genauer betrachtet werden möchte [1]

Verwendung

Im Programm wird dieses Kommando im Zuge der *isRichContentField*-Methode verwendet, um herauszufinden, ob das mitgegebene Feld *RichText* unterstützt.

```
Command myCmd = new Command(Command.IM, "viewfield");
myCmd.addSelection(fieldname);
Response myRes = myCmdRunner.execute(myCmd);
WorkItemIterator wii = myRes.getWorkItems();
WorkItem wi = wii.next();
String refField = wi.getField("BackedBy").getValueAsString();

Command myCmd2 = new Command(Command.IM, "viewfield");
myCmd2.addSelection(refField);
Response myRes2 = myCmdRunner.execute(myCmd2);
WorkItemIterator wii2 = myRes2.getWorkItems();
WorkItem wi2 = wii2.next();
isRichContent = wi2.getField("richContent").getBoolean();

return isRichContent;
```

Abbildung 61: im viewfield

Dem Kommando wird dabei lediglich der gewünschte Feldname als Parameter angehängt. Um herauszufinden, ob ein Feld *RichText* unterstützt, muss zuerst das Feld *BackedBy* ausgelesen werden.

Danach wird das Kommando erneut aufgerufen, um mithilfe des Inhalts des *BackedBy*-Felds das Feld *richContent* auszulesen. Ist der Inhalt dieses Feldes *true*, unterstützt das Feld, das vom Benutzer übergeben worden ist, *RichText*. So steht dem Kopieren von Formatierungen nichts mehr im Weg.

14.3.2. Klassen und Interfaces

14.3.2.1 CmdRunner

Die *CmdRunner*-Klasse mit der Methode *execute(Command cmd)* wird verwendet, um die verschiedenen Kommandos (unter Punkt 14.3.114.3.1) ausführen zu können. [8]

14.3.2.2 Command

Ein Objekt der *Command*-Klasse repräsentiert ein Kommando, wie man es in der *Commandline* eingibt. Die Klasse hat mehrere Felder, darunter die Konstante *IM,* die für alle während der Diplomarbeit benutzten Kommandos verwendet wird.

Jedes *Command*-Objekt hat ein oder mehrere Optionen (z.B. *im viewissue* ***--parentID=value*** …) sowie eine optionale Selektion (z.B. *im viewfield* ***FieldName***). [8]

14.3.2.3 Field

Das Interface *Field* repräsentiert ein Feld eines Items in *Integrity*, welches in einen Feldnamen und einen dazugehörigen Wert enthält. [8]

14.3.2.4 WorkItem

Das Interface *WorkItem* wird im Rahmen der Diplomarbeit benutzt, um Werte aus der Antwort von Kommandos, die aus *Workitems* besteht, auszulesen. [8]

14.3.2.5 Result

Das Interface *Result* wird dazu benutzt, die Ergebnisse, die manche Kommandos (beispielsweise *im createcontent*) liefern, auszulesen und mithilfe deren weiterzuarbeiten. [8]