Manuscript Title

This manuscript (permalink) was automatically generated from Nebucatnetzer/manubot_test@33f330d on December 16, 2019.

Authors

- **Andreas Zweili**
  XXXX-XXXX-XXXX-XXXX · Nebucatnetzer · -
  Contria GmbH · Funded by -

Abstract

1 Über dieses Dokument

1.1 Beschreibung

Diese Arbeit hat zum Ziel, die Planung und Erstellung einer grafischen Oberfläche zum einfachen Bedienen der Software borg [borgbackup], durchzuführen sowie zu dokumentieren.

1.2 Zweck und Inhalt

Zweck dieses Dokumentes ist die vollständige und nachvollziehbare Dokumentation zur Diplomarbeit von Andreas Zweili.

1.3 Aufbau

Inhalte sind in der Regel chronologisch sortiert, vom ältesten zum jüngsten Ereignis, und nach Kapiteln getrennt. An gewissen Stellen kann die chronologische Reihenfolge allenfalls nicht gewährleistet werden.

1.4 Lizenz

2 Initialisierung

2.1 Vision

Die Software soll borg für den durchschnittlichen Computer User zugänglich machen. Backups sollen dabei schnell und unkompliziert erstellt werden können. Auch die Möglichkeit automatischer im Hintergrund laufender Backups soll dem User gegeben sein, damit die Hürde für Backups so tief wie möglich gehalten wird.

Die besten Backups sind solche, bei denen man gar nicht mehr weiss, dass man sie hat bis man sie braucht.

2.2 Ausgangslage

borg ist deshalb interessant, weil es während einem Backup relativ wenig Ressource im Vergleich zu anderen Systemen benötigt und schon relativ lange aktiv entwickelt wird. Dadurch ist es im Alltag geprüft worden. Des Weiteren bietet borg die Funktion für Verschlüsselung, was es einem User ermöglicht die Daten auf einem unsicheren Cloud Speicher abzulegen.

Des Weiteren speichert borg die Daten mit Block basierter dedup ab. Dies hat den riesigen Vorteil, dass bei einem Backup nur die Änderungen auf Block-Ebene gespeichert werden und nicht jedes Mal die ganze Datei kopiert werden muss.

Damit ermöglicht die Software auch Backups von sehr grossen Dateien, wie Videos oder Disk Images von virtuellen Maschinen, in mehreren Versionen. Ohne dabei jedoch signifikant mehr an Speicher zu benötigen. Zusätzlich werden die Backups dadurch rasend schnell ausgeführt. Gerade dieses Feature macht borg in den Augen des Autors besonders interessant, da sich der durchschnittliche User möglichst wenig mit Dingen wie Backups auseinandersetzen möchte. Umso besser also, wenn sie schnell gehen und so wenig Speicherplatz wie möglich verbrauchen.

borg wird jedoch komplett über die Kommandozeile bedient. Somit ist es für normale Benutzer eher schwierig den Zugang zu der Software zu finden, geschweige denn sie zu bedienen.

borg bietet Entwicklern eine json, api, mit welcher sie, von borg ausgegebenen Dateien einfach weiterverarbeiten können.

borg steht unter einer bsd [bsd] Lizenz zur Verfügung und ist somit gemäss den Richtlinien der Free Software Foundation libre [fsflicenses].

Das Projekt muss dabei vom Studenten in Eigenarbeit und einer Zeit von 250 Stunden bis zum 18. März 2019 erarbeitet werden.

2.3 Projektziele

borg ist eine Kommandozeilen basierte Backup Software. Hauptziel dieser Arbeit ist, ein gui für die Software borg zu entwickeln um die Nutzung zu vereinfachen. Da borg selber freie Software ist und mit freier Software viel gute Erfahrungen gemacht wurden, soll das Projekt selber auch wieder libre sein. Zum einen, um der Community etwas zurückzugeben, des weiteren, um anderen Entwicklern die Möglichkeit zu geben die Software zu verbessern und weiter zu entwickeln.

Als Nebenziel soll mit dieser Arbeit auch die Verbreitung von freier Software gefördert werden. Dies wird insbesondere dadurch erreicht, dass die Software selbst unter der gpl Version 3 [gplv3] veröffentlicht wird. Wenn möglich soll während der Entwicklung auch hauptsächlich freie Software verwendet werden, um aufzuzeigen das ein solches Projekte nicht zwingend von proprietärer Software abhängig ist. Die gesamte Arbeit wird zudem zu jedem Zeitpunkt öffentlich einsehbar sein. Der Quelltext der Dokumentation ist unter diesem Link erreichbar:
https://git.2li.ch/Nebucatnetzer/thesis

Die Entwicklung wird hauptsächlich auf einem Linux System stattfinden. Da borg einerseits hauptsächlich auf Unix Systeme ausgelegt ist und anderseits die Hauptzielgruppe des Projektes auch auf Linux Usern liegt. Trotzdem sollen im Projekt cross-plattform fähige Technologien eingesetzt werden, damit es in der Zukunft möglich ist das Projekt auf andere Plattformen auszuweiten.

2.3.1 Ziele inklusive Gewichtung

Im Projektantrag wurden vorgängig folgende Ziele definiert und entsprechend gewichtet. Die Gewichtung wurde dabei so vorgenommen, dass Ziele mit einer Muss-Gewichtung den

Minimalanforderungen der zu entwickelnden Software entsprechen. Die weiteren Ziele wurden von 5 bis 1 gewichtet. Die Bewertung 5 bedeutet, dass die Umsetzung sehr nützlich und oder wichtig für die Software ist und daher in naher Zukunft zu implementieren ist. Ein Ziel mit einer tiefen Bewertung sollte, wenn möglich, auch einmal in die Software integriert werden und ist nicht unwichtig.

2.4 Projektabgrenzung

Die Anwendung beschränkt sich darauf Funktionen von borg grafisch darzustellen oder nützlich zu erweitern, soweit dies über die api möglich ist. Wie in Abbildung:(1) zu sehen ist, werden die Aktionen effektiv immer vom Borg Binary ausgeführt und nicht von der grafischen Oberfläche. Eine Erweiterung von borg ist nicht vorgesehen. Dies aus dem Grund das Backups, Deduplikation und Verschlüsselung sowie deren korrekte Implementation komplexe Themen sind und unbedingt nur von Experten angegangen werden sollten. Die Auswirkungen von Fehlern sind sind schlicht zu gross.

Des Weiteren wird die Grundlage für eine kollaborative Entwicklung geschaffen. Während der Laufzeit der Diplomarbeit werden jedoch keine Inputs aus der Borg Community im Bezug auf die Entwicklung entgegengenommen.

Bugs von borg welche während der Dauer der Diplomarbeit vom Studenten entdeckt werden, wird dieser dem Projekt melden, jedoch nicht selber beheben.

file:pictures/kontextdiagramm.pdf

2.5 Projektmethode

Für das Projekt wurde die wasserfall gewählt. Da nur eine einzige Person am Projekt arbeitet, kann nur ein Task nach dem anderen abgearbeitet werden und viele Aufgaben stehen in Abhängigkeit zueinander. Somit macht das iterative Vorgehen der wasserfall für dieses Projekt am meisten Sinn.

2.6 Konfigurationsmanagement

2.6.1 Versionskontrolle

Die komplette Dokumentation, der Quellcode der Applikation sowie jegliche zusätzlichen Dokumente, wie etwa die Zeitplanung, werden mittels der Software git versioniert. Thematisch zusammengehörende Änderungen werden in einem commit zusammengefasst. Somit ist jederzeit nachvollziehbar, was wann geändert hat. Ein commit sollte dabei gemäss dem Artikel von Chris Beams „How to write a Git commit Message“ [commit] und in englischer Sprache geschrieben sein.

Versionsnummern sind für die Applikation zum jetzigen Zeitpunkt noch nicht vorgesehen. Sollten sie zukünftig einmal verwendet werden, soll eine semantische Versionierung [semver] verwendet werden. Dabei ist eine Versionsnummer immer nach diesem Schema aufgebaut, MAJOR.MINOR.PATCH. Bei Änderungen wird die: 1. MAJOR Version erhöht, wenn man inkompatible Änderungen an der api macht. 2. MINOR Version erhöht, wenn man Funktionalität hinzufügt, die abwärtskompatibel ist. 3. PATCH Version erhöht, wenn man abwärtskompatibel Bug-Fixes hinzufügt. Eine Versionsnummer würde dann so aussehen Version 1.2.3.

Auf jeden Fall sollte, sofern möglich, immer nur lauffähiger Code im Master Branch eingecheckt sein, damit der Master Branch immer eine funktionierende Software repräsentiert. Dies gilt auch für das Repository der Dokumentation. Der Master Branch der Dokumentation sollte maximal mit zwei Befehlen `make clean' und` make' „kompilierbar“ sein.

Als Software für die Versionskontrolle wurde git [git] aus den folgenden Gründen ausgewählt:

- Ist der de facto Standard bei Versionskontrollsoftware
- Läuft auf allen gängigen Betriebssystemen
- Es gibt gratis Services, die man nutzen kann (Github, Gitlab)
- Ermöglicht es offline zu arbeiten und commit erstellen
- Es steht bereits ein eigener git Server zur Verfügung
- git ist aus vorhergehenden Projekten vertraut, dadurch muss kein Aufwand betrieben werden, eine neue Software zu lernen. Zusätzlich hat sich git in den vorhergehenden Projekten als robuste und schnelle Software erwiesen.
- git ist libre unter der gpl v2.

2.6.2 Editor

Sowohl bei der Dokumentation wie auch bei der Programmierung wurde hauptsächlich der Editor GNU Emacs [emacs] verwendet. GNU Emacs ist mit 32 Jahren (seine Wurzeln reichen bis ins Jahre 1976 zurück) wohl eines der ältesten noch aktiven Software Projekte. Emacs ist libre unter der gpl v3. Emacs wurde gewählt, da es ein schneller, schlanker und sehr flexibler Texteditor ist. Von normaler Textmanipulation über Taskmanagement bis zu Emails schreiben ist alles möglich.

2.6.3 Dokumentation

Diese Dokumentation wurde in Org-mode [orgmode], einer Erweiterung für den Text Editor Emacs, geschrieben. Die Syntax von Org-mode erinnert an Markdown. Org-mode bietet einem eine Vielzahl an Hilfen, inklusive dem Erstellen von Tabellen und Spreadsheet Funktionen. Für die finale Version des Dokuments kann Org-mode die ursprünglich Textdatei über LaTeX in eine PDF Datei exportieren.

LaTeX [latex] ist eine Software, welche einem die Benutzung des Textsatzsystems TeXs vereinfacht. LaTeX wurde gegenüber einem „What You See Is What You Get“ (z.Bsp. MS. Word) Editor gewählt, weil es einem mit seiner Markup Sprache erlaubt das Dokument in Textdateien zu erstellen, gerade für Programmierer ist dies eine sehr interessante Lösung. Dadurch, dass LaTeX auch nur aus reinen Textdateien besteht, können die Dokumente auch ohne weiteres in die Versionskontrollsoftware einchecken und die Entwicklung im Log zurückverfolgen. LaTeX ist libre unter der LaTeX Project Public License.

Die Grafiken in diesem Dokument wurden hauptsächlich mit dem Vektor Grafik Editor Inkscape [inkscape] erstellt. Inkscape ist libre unter der GNU Public License v3.

Die Diagramme wurden mit Draw.io [draw] erstellt. Draw.io ist libre unter Apache Lizenz Version 2.0 [apache] und kann sowohl als Desktop Applikation wie auch als Webanwendung genutzt werden.

Beim Design der Arbeit wurden soweit als möglich die typographischen Regeln aus dem Buch „Practical Typography“ von Matthew Butterick [typo] angewandt. Bei den Diagrammen wurden ausschliesslich Farben aus der von Google entwickelten Design Sprache „Material“ [material] eingesetzt.

2.7 Zeitplanung

Die detaillierte Zeitplanung ist dem Ganttchart in der Datei [02_Zeitplanung_Andreas_Zweili.html] zu entnehmen. Bei der Zeitplanung wurde darauf geachtet, dass die Arbeit soweit als möglich nicht mit dem Berufsleben kollidiert. An einem normalen Arbeitstag wurde dabei damit gerechnet das ca. 2 Stunden Arbeit am Abend möglich sein sollten. An einem arbeitsfreien Tag wurde mit 6 Stunden Arbeit gerechnet. Über die Festtage wurden diverse Tage von der Planung ausgenommen, da es nicht realistisch schien, dass an diesen Tagen die Arbeit signifikant vorwärts gehen würde. Auch Schultage wurde nicht als Arbeitstage gerechnet.

Um die Arbeitslast zu verteilen, wurde vom 14. Januar bis zum 11. März auf der Arbeitsstelle jeder Montag als frei eingegeben. Dadurch steht während des Projektes etwas mehr Zeit zur Verfügung, als mit einer 100 Prozent Arbeitsstelle möglich wäre.

[02_Zeitplanung_Andreas_Zweili.html] file:02_Zeitplanung_Andreas_Zweili.html

2.8 Controlling

Mit dem Controlling wird die Planung mit den effektiv verwendeten Ressourcen verglichen und ausgewertet. Somit können für zukünftige Projekte Lehren gezogen werden.

2.8.1 Zeitaufwand

Um den geschätzten Zeitaufwand mit dem effektiv geleisteten Aufwand zu vergleichen wurde die Tabelle:(28) erstellt. Darin werden die beiden Aufwände einander gegenübergestellt und grössere Abweichungen begründet. Die Nummer vor jeder Aufgabe in der Tabelle korreliert dabei mit den Aufgabennummern im Ganttchart.

2.8.2 Ressourcen

In der Tabelle:(26) wurden die für die Arbeit benötigten Materialen erfasst. Da es sich beim Projekt um ein reines Software Projekt handelt ist der Material Aufwand entsprechend gering. Im Abschluss des Projektes werden die geplanten Ressourcen den effektiv verwendeten gegenübergestellt.

2.8.3 Kosten

Werden die internen Lohnkosten des Projektleiters auf ca. 60 CHF pro Stunde geschätzt (dies entspricht in etwa dem doppelten reelen Stundenlohn des Projektleiters), ergeben sich gemäss der Berechnung in der Tabelle:(27), theoretische Kosten von 19080 CHF für die Umsetzung dieser Arbeit. Die Kosten für die Entwicklung werden im Projekt jedoch nicht berücksichtigt. Somit sind diese nur ein rein theoretischer Faktor.

2.9 Projektrisiken

Das Risikomanagement dient dazu Risiken im Projekt zu erkennen und Massnahmen zur Vermeidung zu definieren. Dadurch steht man Risiken nicht unvorbereitet gegenüber, sollten sie eintreffen.

2.9.1 Risikobeschreibung

In der Tabelle: (6) sind die Risiken des Projektes gemeinsam mit ihren Gegenmassnahmen aufgelistet. Somit können gewisse Risiken bereits vorher vermieden werden.

3 Analyse

3.1 SWOT-Analyse

Die SWOT-Analyse ist eine Methode die Stärken, Schwächen, Chancen und Gefahren zu erkennen, indem eine 4-Felder-Matrix ausgefüllt wird.

Grundlage einer guten SWOT Analyse ist eine klare Zieldefinition und Fragestellung.. Die ausgefüllte SWOT-Analyse für dieses Projekt ist in der Abbildung:(2) zu sehen.

file:pictures/swot_analyse.pdf

3.2 Umweltanalyse

Die Projektumwelt-Analyse ist eine Methode die Beziehungen, Erwartungshaltungen und Einflüsse auf das Projekt durch interne und externe soziale Umwelt zu betrachten und zu bewerten. Auf Grundlage der Analyseergebnisse werden erforderliche Massnahmen zur Gestaltung der Umweltbeziehungen abgeleitet. Die Gestaltung der Projektumweltbeziehungen ist eine Projektmanagementaufgabe. In der Tabelle:(3) wurden die Anforderungen und Wünsche mit Einschätzung der Wahrscheinlichkeit und der Einflussnahme aufgenommen. Zusätzlich ist die Beziehung der Stakeholder zum Projekt noch in der Abbildung:(3) grafisch dargestellt.

Da das Projekt so ausgelegt ist, dass der Projektleiter es in Eigenarbeit verwirklichen kann, ist der Einfluss der Stakeholder während der Umsetzung sehr gering. Die User werden bei der Entwicklung mittels einer Usability-Studie miteinbezogen und die borg Community wird mit regelmässigen Posts auf dem offiziellen Github Repository auf dem Laufenden gehalten. Nach Ende der Diplomarbeit soll das Projekt für interessierte Entwickler jedoch offen sein. Der Quellcode wird bereits während der Arbeit öffentlich zur Verfügung gestellt.

file:pictures/stakeholder_diagramm.pdf

3.3 Risiko-Analyse

Bei der Risiko-Analyse wird von einem durchschnittlichen Benutzer ausgegangen, der zur Zeit noch keine Backups macht und beginnen möchte borg zu nutzen, um auf einer externen Harddisk seine Backups zu speichern.

Es wird eine Ist/Soll Analyse gemacht. Jedes Risiko wurde entsprechend der Tabelle: (4) nach der Wahrscheinlichkeit des Eintreffens bewertet und entsprechend der Tabelle: (5) nach seiner

Auswirkung im Bezug auf die Nützlichkeit der gemachten Backups.

In der Tabelle: (6) sind dabei die Risiken für das Szenario aufgelistet und nummeriert. In der Abbildung:(4) ist die Bewertung des Ist-Risikos grafisch dargestellt und in der Abbildung:(5) ist das Soll-Risiko, welches mit dieser Arbeit angestrebt wird, ebenfalls grafisch dargestellt.

Es sollte im Rahmen der Arbeit möglich sein die meisten Risiken zu verringern. Da automatische Hintergrundbackups jedoch ein Kann-Ziel sind wir in dieser Analyse nicht davon ausgegangen, dass man das Risiko Nr. 5 im Rahmen dieser Arbeit reduzieren kann.

file:pictures/istrisiko.pdf

file:pictures/sollrisiko.pdf

3.4 Anforderungskatalog

Der Anforderungskatalog entspricht 1:1 den Zielen, welche in der Tabelle 1 definiert wurden. Im Zeitplan wurde der Fokus hauptsächlich auf die Muss-Ziele gelegt. Ein paar der Kann-Ziele sind im Konzept jedoch auch abgebildet.

3.5 Use Cases

Ein Use Case sammelt alle möglichen Szenarien, die eintreten können, wenn ein Akteur versucht, mithilfe des betrachteten Systems ein bestimmtes Ziel zu erreichen. Dabei beschreibt er, was beim Versuch der Zielerreichung passieren kann. Je nach Ablauf kann auch ein Fehlschlag ein Ergebnis eines Anwendungsfalls sein (e.g. falsches Passwort beim Login). Dabei wird die technische Lösung nicht konkret beschrieben. Die Detailstufe kann dabei sehr unterschiedlich sein.[usecase]

3.5.1 Anwendungsfalldiagramm

„Ein Anwendungsfalldiagramm … ist eine der 14 Diagrammarten der Unified Modelling Language (UML), einer Sprache für die Modellierung der Strukturen und des Verhaltens von Software- und anderen Systemen. Es stellt Anwendungsfälle und Akteure mit ihren jeweiligen Abhängigkeiten und Beziehungen dar.“

Das Anwendungsfalldiagramm für das borg gui ist in der Abbildung: (6) zu sehen.

file:pictures/use_case.pdf

3.5.2 Use Cases Detailbeschreibung

Use Cases werden in der Regel mithilfe einer sogenannten Use Case Schablone im Detail beschrieben, damit klar ist, wie der Ablauf jeweils genau aussieht. Die in diesem Projekt verwendete Schablone wurde von Alistair Cockburn definiert.

Die nachfolgend aufgeführten Use Cases, Tabellen:(7, 8, 9, 10, 11, 12, 13) wurden dem Anwendungsfalldiagramm, Abbildung:(6), entnommen und zusätzlich noch um jeweils ein

Aktivitätsdiagramm, Abbildungen: (7, 8, 9, 10, 11, 12), erweitert um den Ablauf verständlicher zu machen.

Ein Aktivitätsdiagramm ist dabei ein hilfreiches UML Diagramm zum Erweitern von Use Cases und zeigt einem gut die Zuständigkeiten der Aktoren auf.

3.6 Benötigte Funktionalität von Borg

Damit nachvollziehbar ist welche Funktionen von borg verwendet wurden um die Use Cases umsetzen zu können, werden diese hier in Beziehung zur jeweiligen Funktion des gui aufgelistet: * Für das Erstellen von Archiven `borg create' [borgcreate]. * Für das Anzeigen der Archiven` borg list' [borginfo]. * Für das Wiederherstellen der Archive `borg extract' [borgextract]. * Für das Löschen der Archive` borg delete' [borgdelete]. * Zum Mounten der Archive `borg mount' [borgmount]. * Zum Unmounten der Archive` borg umount' [borgumount]. * Zum anzeigen der Repository Statistik ~borg info~ [borginfo].

Die detaillierte Implementation wird in der Sektion [Realisierung] beschrieben.

[Realisierung] siehe Abschnitt 5

4 Konzept

4.1 Varianten

Mit der json api von borg stehen einem diverse Möglichkeiten zur Verfügung, um das Programm anzubinden. Da das Ziel ist, das Programm normalen Nutzern zugänglicher zu machen, bietet sich ein normales Desktop Programm am ehesten an. Desktop Programme werden von allen Computer Usern täglich genutzt und sind somit etwas was sie kennen. Zudem ist es für die User auch viel einfacher zu verstehen, als wenn sie vor der Nutzung einen lokalen Webserver starten und diesen im Anschluss zur Nutzung wieder beenden müssten.

4.1.1 Bewertung

Mit der Idee aus der „Einleitung zu den Varianten“ wurde dann eine Tabelle, mit Anforderungen an die Technologien, erstellt. Die Bewertungspunkte setzen sich einerseits aus Projektzielen anderseits aus für das Projekt sinnvollen Punkten zusammen. Dadurch ergeben sich dann die Bewertungen, welche in der Tabelle:(14) aufgenommen wurden. Die möglichen Varianten wurden danach bewertet. Die effektive Berechnung des Resultats wird nach folgender Formel durchgeführt.

$$G * EP = KE$$

Also die Gewichtung(/G/) multipliziert mit der erreichten Punktzahl(/EP/) ergibt das Kriteriumsergebnis(/KE/). Für das Endresultat wird dann die Summe über alle Kriterien gebildet. Die Variante mit der höchsten Summe wurde für das Projekt ausgewählt.

Mussziele erhalten dabei eine Gewichtung von 10 und Wunschziele eine Gewichtung entsprechend der Bewertung in der Tabelle Projektziele (1).

4.1.2 Backend

Für die Backend Programmierung bieten sich die folgende drei Sprachen an: [C#], [C++] und [Python]. Dies vor allem, weil alle drei Allrounder Sprachen sind und sich gut für Desktop Applikationen eignen.

[C#] siehe Abschnitt 4.1.2.1

[C++] siehe Abschnitt 4.1.2.2

[Python] siehe Abschnitt 4.1.2.3

4.1.2.1 C#

C# ist eine von Microsoft entwickelte Programmiersprache, welche viele Frameworks zur Verfügung stellt. Insbesondere aufgrund der grossen kommerziellen Nutzung und der guten Integration mit Microsoft Windows hat C# eine relative grosse Verbreitung. Bei Linux und OS X ist es jedoch schwieriger C# zu integrieren und zu nutzen da es nicht standardmässig installiert ist und der Fokus von C# hauptsächlich auf Microsoft Windows liegt.

Sie ist zu Teilen libre. Die Common Language Runtime, welche für das Ausführen von Software zuständig ist, ist unter der MIT Lizenz lizenziert [csharp], der aktuelle Compiler Roslyn ist unter der Apache Lizenz verfügbar [roslyn]. Da es sehr viele offizielle Teile um die Sprache C# gibt, kann im Rahmen des Projektes nicht direkt abgeschätzt werden, ob alle benötigten Teile libre sind. Für die Bewertung wird deshalb ein kleinerer Wert als bei C++ und Python genommen.

C# ist die Programmiersprache, welche an der IBZ hauptsächlich gelehrt wird. Dadurch sind die Kenntnisse der Sprache und ihrer Anwendung bereits vorhanden. Ausserhalb der Schule wurde die Sprache jedoch noch nie eingesetzt.

Entwickelt wird C# hauptsächlich mit der ide Microsoft Visual Studio. Dies ist eine sehr umfangreiche und komplexe Software. Visual Studio ist dabei nur für Windows und OS X erhältlich. Es ist auch möglich C# Projekte ausserhalb von Visual Studio zu erstellen, dies ist jedoch nicht sehr einfach.

Der Code ist gut lesbar und es gibt offizielle Styleguides von Microsoft was den Code über Projekte hinaus einheitlich aussehen lässt. Zudem hilft hier auch Visual Studio stark den Code entsprechend zu formatieren. Besonders angenehm sind die Klassen- und Methodennamen der offiziellen Frameworks. Insgesamt sehr gut gelöst aber in Sachen Lesbarkeit noch etwas hinter Python.

Unter Windows ist das Setup von C# relativ einfach. Allerdings ist es auch dort im Vergleich zu Python eine umfangreiche Angelegenheit Visual Studio sauber zu installieren und nutzbar zu machen. Auf anderen Plattform wird dies leider nicht einfacher und unter Linux ist es bereits schwierig eine funktionierende Umgebung in Gang zu bringen.

Da C# bereits an der IBZ gelehrt wird, ist der Lernfaktor hier, im Vergleich zu den anderen Sprachen, sicher am kleinsten. Allerdings gibt es noch keinerlei Kenntnisse beim Einbinden eines der unten aufgeführten gui Frameworks. Daher gibt es auf jeden Fall noch genügend zu lernen.

Die borg Community hat vor relativ kurzer Zeit die offizielle Unterstützung von Windows zurückgezogen. Da C# eine sehr Windows lastige Sprache ist, wird daher davon ausgegangen, dass die Sprache innerhalb der borg Community nicht sehr verbreitet ist.

C# ist eine stark typisiert Sprache und kompilierte Sprache. Des Weiteren ist Visual Studio der Erfahrung nach nicht die schnellste Software. Dies alles führt dazu das C# nicht gerade die schnellste Sprache zum Programmieren ist. Jedoch aufgrund des moderneren Unterbaus ist sie sicher schneller als C++.

4.1.2.2 C++

C++ ist eine stark typisierte und kompilierte Programmiersprache. Sie ist seit 1998 Teil des ISO Standards [cpp98]. ISO/IEC 14882:2017 [cpp17] ist zurzeit die aktuellste Variante. Die Sprache existiert seit ca. 33 Jahren und hat eine weitreichende Verbreitung gefunden. C++ ist auf allen Betriebssystemen gut unterstützt muss jedoch für jedes System separat kompiliert werden.

Von C++ sind innerhalb des Projektes keinerlei Vorkenntnisse vorhanden. Dies ist ein sehr hoher Risikofaktor.

C++ kompiliert direkt zu Maschinensprache und ist dadurch sehr performant und läuft sehr gut auf jedem System. C++ ist im Vergleich zu modernen Sprachen jedoch relativ komplex und bietet diverse Stolpersteine für Programmierer.

Zum Entwickeln braucht es verhältnismässig wenig Werkzeuge. Da die Sprache bereits sehr alt ist, stammt sie noch aus einer Zeit, wo man noch etwas rudimentärer programmierte. Allerdings braucht man in jedem Fall einen compiler, um ein Programm zu erzeugen. Bei komplexeren Programmen wird man, um mindestens so etwas wie makefile auch nicht herumkommen

Im Vergleich zu Python oder C# ist C++ wohl die am schwersten lesbare Sprache. Zudem gibt es auch keinen zentralen Styleguide, welcher einem vorgeben würde wie der Code am besten ausschauen sollte. Somit haben sich über die Jahre mehrere Standards etabliert.

Der Lernfaktor wäre aufgrund der mangelnden Vorkenntnisse hier ganz klar am Grössten.

Da C++ eine alte Sprache ist, geniesst sie auch eine dementsprechende Verbreitung. Daher ist anzunehmens dass sicher mindestens ein grösserer Teil der älteren borg Entwickler C++ oder C gelernt haben.

Da C++ auch heute noch zu den meistgenutzten Sprachen gehört, gibt es entsprechend viele Ressourcen dazu und Beispielprojekte, von denen man ableiten kann. Auch hilfreiche Libraries gibt es sehr viele, welche den Programmierer unterstützen können. Die Sprache selber ist jedoch eher umständlich zu schreiben. Hinzu kommt noch, dass man, während der Entwicklung immer wieder den Code kompilieren muss. In einem Projekt mit dieser begrenzten Zeitspanne eher ungeeignet.

4.1.2.3 Python

Der Python Interpreter ist für eine Vielzahl an Betriebssystemen erhältlich, inklusive Windows, OS X und Linux. Nahezu jedes Desktop Linux System kommt mit Python vor installiert. Auch OS X

kommt bereits ab Werk mit Python Version 2. Version 3 lässt sich sehr einfach nachinstallieren und ist einfach nutzbar. Unter Windows gestaltetet sich die Installation etwas aufwendiger aber auch nicht sehr kompliziert. Python integriert sich in Windows jedoch etwas weniger elegant als C#.

Python ist freie Software unter der Python Software Foundation License [python] und wird durch die Python Software Foundation in einem Community basierten Modell entwickelt.

Die Vorkenntnisse sind im Vergleich zu C++ relativ gross und zu C# etwas weniger ausgeprägt. Es wurden damit im Rahmen der Ausbildung schon ein grösseres Projekt realisiert und ansonsten mehrere kleine Projekte im Privaten erstellen.

Für Python gibt es ein paar ide welchen den Programmierer bei seiner Arbeit unterstützen können. Keine davon ist allerdings ein Muss, um Python programmieren zu können. Im einfachsten Fall wäre dies mit Notepad möglich. Ein Editor mit etwas fortgeschritteneren Features wäre jedoch empfehlenswert.

Python unterstützt mehrere Programmierungsparadigmen wie etwa objektorientiert, funktionale oder prozedurale Paradigmen. Bei der Entwicklung von Python wurde sehr grossen Wert auf die Lesbarkeit der Sprache gelegt. Dies mit dem Hintergedanken das eine Programmiersprache viel häufiger gelesen als effektiv geschrieben wird [pep8].

Um ein Python Programm zu starten, braucht es eigentlich kein grosses Setup. Solange die Abhängigkeiten vorhanden sind, kann man ein Skript mit einem einfachen Befehl, Code Snippet (1) starten.

```
python3 example.py
```

Programmlisting 1 Minimal Python Setup

Da Python schon eine etwas bekanntere Sprache ist, ist der Lernfaktor der Sprache selber nicht mehr so hoch. Allerdings gibt es noch viele interessante Konzepte, die man im Zusammenhang mit der Sprache lernen kann. Wie etwa zum Beispiel multiple Vererbung von Klassen.

borg selber wurde in Python geschrieben. Daher ist davon auszugehen, dass Python innerhalb dieser Community eine sehr hohe Verbreitung geniesst.

Python ist eine dynamisch typisierte und interpretierte Sprache. Dies bedeutet, dass man bei Variablen nicht explizit den Typ angeben muss und die Programme zur Laufzeit für den Computer übersetzt werden. Interpretierte Sprachen haben den Vorteil, dass man mit ihnen in der Regel sehr schnell und unkompliziert entwickeln kann, dies jedoch zulasten der Performance.

4.1.3 Frontend

Fürs Frontend sind folgende Projekte interessant: [Qt], [Gtk] und [Electron]. Alle drei sind cross-plattform fähige gui Frameworks und nicht von einer spezifischen Sprache abhängig. Da nahezu keine Erfahrung mit den aufgeführten Frameworks vorhanden ist, werden bei den Frontend Frameworks die Punkte der Verbreitung in der Community und Geschwindigkeit der Entwicklung ausgeschlossen. In beiden Fällen wäre nicht mal eine ungenaue Schätzung wirklich möglich.

[Qt] siehe Abschnitt 4.1.3.1

[Gtk] siehe Abschnitt 4.1.3.2

[Electron] siehe Abschnitt 4.1.3.3

4.1.3.1 Qt

Qt [qt], „cute“ ausgesprochen, ist ein Framework zum Entwickeln von grafischen Oberflächen, welche auf verschiedenen Systemen ohne grosse Änderungen laufen sollen und sich dabei soweit als möglich wie eine native Applikation verhalten und „anfühlen“ soll.

Die Rechte an Qt hält die Firma „The Qt Company“. Das Framework Qt wird jedoch offen entwickelt und die Community hat ein Mitspracherecht. Die Linux desktopumgebung KDE nutzt das Qt Framework intensiv. Qt ist libre und der gpl v3 [qtlicense] oder mit einer kostenpflichtigen proprietären Lizenz erhältlich, falls die gpl nicht genutzt werden kann.

Vorkenntnisse zu Qt sind nur sehr wenig vorhanden. Mehr als ein paar Tests wurden damit noch nicht gemacht.

Eine Qt Oberfläche kann direkt in der jeweiligen Sprache des Backends geschrieben werden oder Mittels des Qt Designers als xml Datei gespeichert und dann in die eigentliche Applikation importiert werden. Somit ist keine spezielle Software nötig.

xml ist nicht übermässig gut lesbar, allerdings kann man Qt in der verwendeten Sprache programmiert werden, somit ist es hauptsächlich von der Sprache im Backend abhängig. Die Dokumentation ist in C++ geschrieben, was für einen Entwickler ohne C++ Kenntnisse die Software etwas unzugänglich macht.

Qt scheint, soweit dies bis jetzt abgeschätzt werden kann, sehr leicht in ein Projekt integrierbar zu sein.

Da noch sehr wenig Kenntnisse vorhanden sind, ist der Lernfaktor entsprechend gross.

4.1.3.2 Gtk

Gtk ist sowohl für Linux wie auch für Windows und OS X erhältlich. Gtk hat als Projekt der Gnome Foundation seine Wurzeln jedoch ganz klar in der Linux Welt. Gtk ist libre unter der Lesser General Public Lizenz [gtklicense]. Gtk ist ein Projekt der GNOME Foundation einer nicht für Profit Organisation, welche die Entwicklung diverser freier Software Projekte koordiniert.

Zu Gtk gibt es keinerlei Vorkenntnisse als Programmierer. Gtk wurde bis jetzt nur intensiv als User verwendet.

Gtk integriert sich nur unter Linux wirklich gut ins System. Unter Windows und OS X können die Applikationen schnell etwas fremd wirken. Dies ist gut bei der Applikation Meld [meld] zu sehen, wenn man eine Datei auswählen möchte, Abbildung (13).

file:pictures/meld.png Die Gtk Dokumentation empfiehlt [gtk_setup], dass man unter Microsoft Windows das Programm MSYS2 installiert, um Gtk einzurichten. Zum Programmieren an sich braucht es nicht zwingend weitere Tools aus einem Editor. Wie auch bei Qt hat man jedoch die Möglichkeit das gui mit einem gui Designer grafisch zu erstellen.

Wie auch Qt kann man Gtk entweder direkt in der Backend Sprache programmieren oder aus dem gui Designer, dann als xml exportieren. Der Code in der Dokumentation ist in C geschrieben, welches auch nicht die zugänglichste Sprache ist.

Die Verwendung von Gtk innerhalb des Programms scheint ähnlich einfach zu sein wie bei Qt. Die Installation ist allerdings unter Windows eher das Gegenteil von einfach.

Da die Kenntnisse gleich null sind, ist der Lernfaktor auf dem Maximum.

4.1.3.3 Electron

Electron ist ein cross-plattform Framework zum Entwickeln von gui, welches dabei jedoch auf Technologien aus der Webentwicklung benutzt. Entwickelt wird Electron von der Firma Github und ist libre unter der MIT Lizenz [electronlicense].

Da Electron auf Technologien aus der Webentwicklung setzt, sind hier im Vergleich zu den anderen Frameworks bereit gute Kenntnisse vorhanden. Über die genau Funktion und Implementierung sind noch keine Kenntnisse vorhanden.

Die Verwendung von Webtechnologien macht Electron zwar sehr kompatibel auf den unterstützten Systemen, oftmals sehen die Applikationen jedoch eher wie eine Webseite als wie eine Desktop Applikation aus. Ein weiterer Nachteil ist der hohe Ressourcenverbrauch, da jede Applikation nahezu einer eigenen Instanz des Google Chrome Browsers gleich kommt.

Bei der Installation muss Node.js und der Paket Manager von Node.js, NPM, vorhanden sein. Zum Programmieren selber braucht es keine speziellen Tools. Ein Editor und ein Webbrowser sollten ausreichend sein.

Electron Applikationen bestehen hauptsächlich aus html, css und JavaScript Code. Wenn man sich die komplette Applikation in Node.js programmieren möchte, kommt dann noch eine zusätzliche Sprache hinzu. html ist ähnlich mühsam zu lesen wie xml. css und JavaScript sind relativ angenehm zu lesen, wobei es für beide keine offiziellen Styleguides gibt. Was bei Webanwendungen jedoch immer das schwierigste ist, ist der Wechsel zwischen verschiedenen Sprachen und Konzepten. Dieses Problem hat man bei Electron leider auch.

Das Setup von Electron ist etwa ähnlich kompliziert wie das Setup von Gtk und ist sehr ähnlich dem Entwickeln einer normalen Webapplikation.

Da an der IBZ Webtechnologien bereits intensiv behandelt worden sind und man in diesem Rahmen bereits ein paar Webapplikationen erstellt hat, wäre der Lernfaktor bei Electron wohl nicht so gross wie etwa bei Qt oder Gtk.

4.1.4 Ergebnis

Aufgrund der erreichten Punktzahl, Tabelle:(21), bei den vorhergehenden Variantenbewertungen, wurde entschieden für das Backend der Applikation auf Python zu setzen und fürs Frontend Qt zu benutzen.

4.2 Applikationsname

Da die einzusetzende Technologie nun feststeht lässt sich auch gut ein Name für die Applikation ableiten. Oftmals werden die grafischen Applikationen gleich benannt wie die Kommandozeilen Applikation aber mit dem Namen des gui Frameworks als Suffix. Somit wird das zu erstellende gui für borg im weiteren Verlauf der Arbeit nun Borg-Qt genannt

4.3 Testing

Die Anwendung wird während der Realisierung soweit als möglich mit automatischen unittest und funktionstest überprüft. Dies hauptsächlich, um die Erfahrung in diesem Bereich zu erweitern um ein gutes Fundament für die Zukunft des Projektes zu bauen.

Aufgrund der Unerfahrenheit im Bereich des automatisierten Testings wurden noch die Testfälle in der Tabelle:(29), erstellt. Diese werden final von Hand überprüft. Somit kann vermieden werden, dass nicht funktionierende automatische Tests den Abschluss des Projektes verhindern. Da die Testfälle sich hauptsächlich an den Use Cases orientieren, gibt es ein paar Ziele die, dadurch nicht getestet werden können. Zudem sind zurzeit nur ca. 20 der Ziele durch die Use Cases abgedeckt. Die weiteren Ziele lassen sich erst sinnvoll integrieren, wenn die Basis für das Programm geschaffen wurde. Somit werden diese Ziele erst im Anschluss zur Diplomarbeit umgesetzt.

Getestet wird die Applikation jeweils auf dem Computer des Projektleiters. Auf diesem läuft die aktuelle Langzeitsupport Version (18.04) von Ubuntu [ubuntu] Linux, mit der GNOME Desktop Umgebung [gnome], als Betriebssystem. Die Tests werden jeweils gegen eine von PyInstaller generierte Binärdatei ausgeführt. Der genaue Vorgang der Erstellung dieser Datei wird in der Sektion: [Releases] beschrieben. Somit werden die Tests immer gegen eine veröffentlichbare Version gemacht.

Als Testdateien wird jeweils das Code Repository von Borg-Qt selber verwendet. Der Pfad des borg Repository für lokale Backups soll `/tmp/test-borgqt' sein, in den Testfällen „Lokales Repository“ genannt und das Passwort` foo’. Im Makefile des Repository wird dieses Setup definiert. Somit kann man als Entwickler nur `make init’ ausführen und hat eine funktionsfähige Testumgebung.

Um Backups über ssh testen zu können, wird eine virtuelle Maschine mit Ubuntu 18.04 verwendet. Die Konfiguration der virtuellen Maschine sieht dabei wie folgt aus: * 2 CPU Kerne * 1024 MB RAM * IP: 10.7.89.117 * Ein User `borg' mit Passwort` borg’ * borg Repository unter `/home/borg/backup/diplom' mit Passwort` foo’, in den Testfällen „Server Repository“ genannt * Der ssh Key des Entwicklers wird in den User `borg’ importiert. Dies ermöglicht Passwort freie Logins.

Die Testfälle werden während der Entwicklung kontinuierlich durchgeführt. Am Ende der Diplomarbeit wird das finale Ergebnis des jeweiligen Testfalles erfasst. Allfällige Besonderheiten werden im Kapitel [Realisierung] beschrieben.

[Releases] siehe Abschnitt 5.11

[Realisierung] siehe Abschnitt 5

4.4 Klassendiagramm

Um die Abhängigkeiten zwischen den einzelnen Klassen der Anwendung aufzuzeigen, wurde ein Klassendiagramm, Abbildung:(34), erstellt. Das Klassendiagramm basiert auf dem UML Standard. Im Diagramm wurden nicht alle „Properties“ und Methoden alles Klassen aufgezeichnet, sondern nur solche, die auf eine andere Klasse verweisen. Dadurch bleibt das Diagramm übersichtlicher. Die Klassennamen, welche in fetter Schrift gehalten sind, wurden dabei vom Projektleiter erstellt. Die Klassennamen, welche kursiv sind, sind Klassen, welche entweder von Python oder Qt bereitgestellt werden.

4.5 Usability-Studie

Um Borg-Qt auf seine Nutzerfreundlichkeit zu testen, wird im Rahmen der Diplomarbeit noch eine kleine Usability-Studie gemacht. Bei einer solchen Studie erhalten die Probanden, Tabelle:(22), ein paar Aufgaben, Sektion 4.5.1, welche sie in einer begrenzten Zeit zu erledigen haben. Die Aufsichtsperson gibt ihnen dabei keinerlei Hilfestellungen. Die Probanden sollen die Aufgaben alleine mithilfe der Tipps und Hinweisen in der Anwendung lösen. Im Anschluss bewerten die Probanden dann die einzelnen Aufgaben nach ihrer Schwierigkeit, Tabelle:(23). Daraus lässt sich dann eine sogenannte Heatmap erstellen. Aus der Heatmap kann man anschaulich herauslesen, welche Bereiche für die User noch zu kompliziert sind und Nacharbeit benötigen.

Die Probanden wurden aus dem Umfeld des Projektleiters ausgewählt. Es wurde dabei versucht ein einigermassen breites Spektrum an Computerkenntnissen abzudecken. Da die Anwendung allen Erfahrungsstufen behilflich sein soll. Die Angaben in der Tabelle:(22) sind jedoch die Selbsteinschätzung der Probanden und nicht die des Projektleiters.

4.5.1 Aufgaben

1. Du möchtest deine Dateien sichern. Erstelle dazu eine Datensicherung des Ordners `/home/testuser/Downloads’.
2. Du hast aus Versehen die Datei `/home/testuser/Downloads/Example.pdf' gelöscht. Stelle die Datei wieder her. Am Ende soll sie unter` /home/testuser/Documents/Example.pdf’ zu finden sein.
3. Stelle ein beliebiges Archiv wieder her. Der Zielpfad ist `/home/testuser/Documents/’.
4. Lösche ein Archiv deiner Wahl.
5. Du möchtest, dass der Ordner `/home/testuser/Pictures/’ nicht mehr gesichert wird. Konfiguriere die Applikation entsprechend.

4.5.2 Resultate

4.5.2.1 Proband 1

Der Proband fand die Aufgaben grundsätzlich einfach zu lösen. Dass die „Mount“ Funktion zum Wiederherstellen einzelner Dateien gedacht war, hat er nicht erkannt.

4.5.2.2 Proband 2

Der Proband kam mit den Aufgaben insgesamt gut klar. Bei der ersten Aufgabe hätte er sich eine Meldung gewünscht, wenn das Backup erfolgreich durchgelaufen ist. Wie Proband 1 hat auch er die „Mount“ Funktion nicht genutzt zum Wiederherstellen einer einzelnen Datei. Text Hinweise wurden nur bedingt wahrgenommen.

4.5.2.3 Proband 3

Proband 3 kam mit der Anwendung an sich gut klar. Die Aufgabe Zwei fand er über alles gesehen auch am schwierigsten, da er mit der Materie nahezu nicht vertraut ist. Als zusätzlichen Input gab er an, dass ein Kontextmenü, welches sich mit Rechtsklick auf ein Element öffnet, etwas sei was er gerne hätte, da er andere Anwendungen oft so steuert. Aufgabe 5 war auch etwas herausfordernder als 1,3 und 4, insbesondere war unklar wie der Ordner zu der Liste hinzugefügt werden sollte.

Während des Tests ist in der Anwendung noch ein Bug aufgetaucht, welcher unter gewissen Umständen Probleme beim Erstellen von Archiven machte. Die detaillierte Lösung dafür ist im Kapitel 5 beschrieben.

4.5.2.4 Proband 4

Bei Proband 4 war die grösste Hürde, dass das Interface nur in Englisch verfügbar war. Bei Aufgabe Zwei hatte er sich nach eigenen Angaben etwas verloren gefühlt und hätte sich auch ein Kontextmenü auf dem Rechtsklick gewünscht. Mit etwas Hilfe bei der Übersetzung waren die restlichen Aufgaben jedoch gut zu meistern.

4.5.2.5 Probandin 5

Probandin 5 mit der Anwendung insgesamt sehr gut klar und hat auch als Einzige die Tooltips auf den Buttons entdeckt und dann genutzt. Aufgabe 2 war jedoch auch schwierig zu lösen, danach ging es jedoch ohne Probleme.

4.5.3 Auswertung

Alle Testpersonen konnten die Applikation nach anfänglichen Bedienungsschwierigkeiten sehr gut bedienen. Um Hilfestellung zu leisten, wird im Rahmen der Diplomarbeit noch ein Hilfefenster eingebaut, welches den Benutzern beim ersten Starten der Anwendung angezeigt wird und kurz die jeweiligen Elemente des Interfaces anzeigt. Somit sollte auch das Problem bei der Aufgabe Zwei etwas abgeschwächt werden. Eines der Hauptprobleme war dort, dass die Probanden nicht herausgefunden haben, dass der schnellste Weg eine einzelne Datei wieder herzustellen über die „Mount“ Funktion ginge. Die Einarbeitung in die Thematik von Backups würde sich jedoch wohl nur sehr schwer über das gui realisieren lassen. Hier müsste auf jeden Fall eine Dokumentation oder im Idealfall eine Schulung Abhilfe schaffen.

Der von zwei Usern geäusserte wertvolle Hinweis, ein Kontextmenü anzubieten, wird in die künftige Weiterentwicklung der Applikation eingepflegt. Aus Ressourcengründen allerdings erst nach der Diplomarbeit.

Ein Pop-Up, welches ein erfolgreiches Erstellen eines Archivs bestätigt, wird nicht eingebaut. Bei erfolgreicher Durchführung verschwindet der Fortschrittsdialog und in der Archivlist erscheint ein weiterer Eintrag. Das sind zwar nicht die offensichtlichsten Hinweise im Falle eines Fehlers, erscheint jedoch sofort ein Dialog, der darauf hinweist. Somit sollten die beiden Vorgänge genügend unterschieden sein und es hat auch kein anderer Proband das Bedürfnis nach einer Bestätigung.

Eine Deutschübersetzung, eine weitere Anforderung der Usability Tester, wird auch für zukünftige Entwicklungen aufgenommen und nicht im Rahmen der Diplomarbeit umgesetzt.

Im Rahmen der Diplomarbeit werden noch einige Texte angepasst. An gewissen Stellen war die Rede von „Backups“ und an anderen von „Archives“. Da borg sie selber „Archives“ nennt, sollte Borg-Qt noch so angepasst werden das überall von „Archives“ die Rede ist. Zudem wird bei den „Include“ und „Exclude“ Optionen über der Liste noch ein Label hinzugefügt, um die Elemente zu beschreiben. Schlussendlich werden die Buttons „Add file“ und „Add folder“ zu „Exclude file“ und „Exclude folder“ sowie „Include file“ und „Include folder“ umbenannt. Somit zeigen die Buttons dann auch direkt, dass sie Dateien respektive Ordner ein-/ausschliessen. Ein paar der Probanden hatten es zuerst über den „Remove“ Button versucht.

5 Realisierung

5.1 Cross-plattform Kompatibilität

Um sicherzugehen, dass die gewählten Technologien auch den Anforderungen entsprechen wurde ein kleines „Hello World“ Programm mit Python3 und Qt geschrieben. Dieses läuft ohne jegliche Probleme und Anpassung auf Windows, Linux und OS X. Wie in den Screenshots in Abbildung:(14) zu sehen ist.

file:pictures/hello_world.png

5.2 Benutzerinterface

5.2.1 Inspiration

In der Vorstudie zur Diplomarbeit wurde borg mit der Software „Back in Time“[backintime] verglichen. „Back in Time“ setzt auf Rsync zum Kopieren der Dateien. Dies erlaubt es „Back in Time“ auch schnelle Backups über ssh zu machen allerdings ohne dedup.

Das übersichtliche Userinterface in Abbildung:(15), wurde für Borg-Qt als Vorlage genommen. Insbesondere die einfache und direkte Art ein Backup eines spezifischen Pfades zu machen ist sehr gelungen. Da sie es dem User so einfach wie möglich macht ein Backup zu erstellen.

file:pictures/bit_main.png

5.2.2 Erste Umsetzung

Qt bietet einem mehrere Möglichkeiten zum Erstellen der grafischen Oberfläche. Zum einen kann die ganze Oberfläche programmatisch erstellt werden. Dies gibt dem Entwickler ein grosses Mass an Kontrolle, ist allerdings nicht sehr intuitiv.

Die angenehmere Variante ist es den Qt Designer, Abbildung:(16), zu nutzen. Mit diesem lassen sich die Oberflächen in einer grafischen Oberfläche designen und auch gleich starten. Damit ist direkt zu sehen wie sich die Oberflächen auf dem System verhalten.

file:pictures/qt_designer.png Mit der ersten gui Version wurden die ersten Basisziele der Projektarbeit umgesetzt. Im Hauptfenster, Abbildung:(17), befinden sich wie auch bei „Back in Time“ in der einen Hälfte eine Liste der vorhandenen Archive und in der anderen Hälfte ein Dateimanager. Dieser dient zur Auswahl des zu sichernden Pfades. Im oberen Bereich findet sich die Toolbar mit den Aktionen, die der User ausführen kann. Gemäss den Use Cases sind dies „Backup, Restore, Mount, Delete und Settings“.

Bei den Icons wurde zuerst versucht diese nach der „Icon Naming Specification“[iconnamespec] auszuwählen. Diese Spezifikationen würden es erlauben einfach den definierten Namen des Icons anzugeben. Qt würde dann jeweils das passende Icon basierend auf dem System anzeigen. Somit wären die Icons passend zum jeweiligen Betriebssystem. Allerdings gab es für die Aktionen keine passenden Icons in der Spezifikation. Deshalb wurden schlussendlich das „Feather“ Icon Theme Set [feathericons] ausgewählt. Dabei handelt es sich um ein freies Icon Theme unter der MIT Lizenz, welches die Icons als svg Dateien bereitstellt. Dadurch können die Icons frei skalieren und funktionieren auch auf Geräten mit einer hohen Auflösung.

file:pictures/borgqt_main_v1.png

Im Einstellungsfenster gibt es drei Tabs zur Auswahl. Einmal den „General“ Tab, Abbildung:(18), dieser zeigt allgemeine Optionen an. Im zweiten Tab „Include“, Abbildung:(19), kann der User die Ordner und Dateien auswählen, die er sichern will. Der dritte Tab „Exclude“, Abbildung:(20), gibt dem User die Möglichkeit einzelne Ordner oder Dateien von den Backups auszuschliessen.

file:pictures/borgqt_settings_general_v1.png

file:pictures/borgqt_settings_include_v1.png

file:pictures/borgqt_settings_exclude_v1.png

Das „Progress“ Dialogfenster, Abbildung:(21), zeigt dem User einen Fortschrittsbalken und einen „Cancel“ Button zum Abbrechen der Aktion an. Das Fenster ist generisch gehalten, damit es von verschiedenen Tasks gleichermassen genutzt werden kann.

file:pictures/borgqt_progress_v1.png

5.3 Einstellungen

Die Einstellungen werden von der Applikation benötigt, um die vom User definierten Vorgaben auszuführen, das Backup Repository zu finden, etc. Diese Einstellungen sollen in einer Klar-Text Datei gespeichert werden. Dies hat zum einen den Vorteil, dass man die Einstellungen sehr einfach sichern kann. Zum anderen kann man die Einstellungen der Applikation auch anpassen, ohne dass man die Applikation selber starten muss.

5.3.1 Backend

Zum Erstellen und Auslesen der Konfigurationsdatei wurde das Python Standard Modul `configparser' [configparser] verwendet. Dieses macht es einem sehr einfach eine Datei im „INI“ Stil zu erstellen und parsen.

„INI“ Stil bedeutet dabei das die Einstellungen in „Key/Value“ Paaren gespeichert werden. Somit kann man einfach auf den benötigten Wert zugreifen, in dem man seinen Schlüssel angibt. Ein Beispiel ist im Code Snippet: (2) zu sehen.

```
# docs/borg_qt.conf.example
[borgqt]
includes = [
       "/home/username/",
        "/home/otheruser/Downloads"
   ]
repository_path = /tmp/test-borgqt
password = foo
prefix = muster
```

Programmlisting 2 Ein Beispiel einer INI Datei.

Das Auslesen und Schreiben der Konfigurationsdatei liess sicher relativ einfach realisieren. Die grösste Herausforderung dabei war, dass `Configparser' keinen Support für eine Liste von Werten hat. Die wurde insbesondere für` include' und `exclude' Pfade benötigt. Also für die Pfade, welche gesichert werden oder von einem Backup ausgeschlossen werden sollen.

Abhilfe schaffte hier ein Stackexchange Post [configlist]. Dieser schlug vor, dass man die Liste im json Format speichern soll. Da `Configparser' alle Werte im Format „String“ zurückgibt, können dann die json Listen sehr einfach von einem json Parser umgewandelt werden. Im Projekt wurde dies dann unter anderem als Methode der` Config' Klasse, Code Snippet:(3), implementiert. Somit muss man jeweils nur die `_return_list_option()' Methode mit der benötigten Option als Argument aufrufen und bekommt als Resultat eine funktionierende Python Liste zurück.

Beim Schreiben der Konfigurationsdatei macht man dann einfach das Umgekehrte. Man konvertiert eine Python Liste in einen json String.

```
# borg_qt/config.py

def _return_list_option(self, option):
    """Reads the provided option from the configparser object and
       returns
    it as a list."""
    if option in self.config['borgqt']:
        return json.loads(self.config['borgqt'][option])
    else:
        return []
```

Programmlisting 3 Methode zum Parsen von JSON Listen in Konfigurationsdateien.

Die Datei wird jeweils beim Start der Applikation gelesen und angewendet. Somit weiss die Applikation bereits nach dem Start wo das Repository liegen sollte und wie die Login Daten dafür sind. Dies geschieht mittels der Methode `_get_path', Codesnippet:(4). Es gibt dabei zwei mögliche Pfade, wo die Konfigurationsdatei liegen könnte. Befindet sich die Datei nicht am vorgegeben Pfad` ~/.config/borg_qt/borg_qt.conf' oder direkt „neben" dem Binary, gibt die Applikation eine entsprechende Meldung, Abbildung:(22), aus. Der Hauptpfad unter `~/.config/borg_qt/borg_qt.conf' wird dabei gemäss dem Ziel Nr. 21 über die Umgebungsvariable` HOME' zusammengesetzt

```
# borg_qt/config.py

def _get_path(self):
    """searches for the configuration file and returns its full
       path."""
    home = os.environ['HOME']
    dir_path = os.path.dirname(os.path.realpath(__file__))

    if os.path.exists(os.path.join(home,
       '.config/borg_qt/borg_qt.conf')):
        return os.path.join(home, '.config/borg_qt/borg_qt.conf')
    elif os.path.exists(os.path.join(dir_path, 'borg_qt.conf')):
        return os.path.join(dir_path, 'borg_qt.conf')
    else:
        raise BorgException("Configuration file not found!")
```

Programmlisting 4 Methode zum Suchen der Konfigurationsdatei

file:pictures/borgqt_missing_config.png

5.3.2 Frontend

Zur Vereinfachung der Bedienbarkeit wurde die Applikation, um eine grafische Konfigurationsmöglichkeit erweitert. Diese stellt dabei hauptsächlich die Werte aus der Konfigurationsdatei grafisch dar und übergibt allenfalls geänderte Werte ans Backend, welches die Konfiguration, dann wieder in der Datei speichert.

Qt kennt keinen Mechanismus zum Auslesen aller Elemente aus einem sogenannten `QListWidget', einem gui Element, welches Listen darstellt. Die Elemente müssen somit zuerst in einer Zwischenliste gespeichert werden, bevor sie zurück in das` Configparser' Objekt geschrieben. Im Code sieht dies dann wie in Codesnippet:(5) aus. Dabei wird jedes Element einzeln aus dem `QListWidget' geholt und in die Zwischenliste geschoben. Im zweiten Teil wird die Liste dann wieder zu einem json String konvertiert und im` Configparser' Objekt gespeichert. Die Option `indent=4' dient dabei der Lesbarkeit, damit nicht der ganze json String auf ein Zeile in der Konfigurationsdatei gespeichert wird, sondern jedes Listenelement seine eigene Zeile erhält.

```
# borg_qt/config.py

# Workaraound to get all items of a QListWidget as a list
includes = []
for index in range(self.list_include.count()):
    includes.append(self.list_include.item(index).text())

# Configparser doesn't know about list therefore we store them as
       json
# strings
self.config['borgqt']['includes'] = json.dumps(includes,
                                                indent=4,
                                                sort_keys=True)
```

Programmlisting 5 Workaround zum Auslesen aller Elemente in QListWidgets.

5.4 Borg Interface

Zuerst erschien es sinnvoll die Kommunikation zwischen borg und Borg-Qt über einfache Funktionen laufen zu lassen. Dieser Ansatz hatte allerdings zwei Probleme. Zum einen wurde es relativ umständlich Informationen zu verarbeiten und weiterzugeben, zum anderen führte es zu dem unschönen Nebeneffekt, dass das gui eingefroren ist. Eine Recherche ergab, dass Threads hier Abhilfe schaffen könnten.

Python liefert für Threads das Modul `threading.Thread' [threading], mit. In der Praxis lies sich der Fortschrittsdialog und der Thread jedoch nicht so verknüpfen das sich der Dialog schliesst, wenn das Backup durchgelaufen ist und der Thread wieder entfernt wird. Aus diesem`

Grund wurde dann ein erfolgreicher Test mit dem PyQt Modul QThread' [qthread] gemacht. Nach Beendigung des Backups wird der Fortschrittsdialog automatisch geschlossen. Auch das Stoppen des Threads mit einem Klick auf den „Cancel" Button funktioniert einwandfrei.

Damit borg aus der Anwendung angesteuert werden kann wird das Python Modul subprocess' [subprocess] verwendet. Dieses erlaubt einem neue Prozesse zu erstellen, welche man oftmals benötigt um etwa, wie im Fall von Borg-Qt, externe Applikationen zu starten, zu steuern und ihre Ausgabewerte auszulesen. Das effektive Kommando wird dann aus dem Property self.command' gelesen.

Damit borg die Ausgabe im json Format ausgibt, muss man man noch die Parameter --log-json' und –json' mitgeben. Der erste Parameter ändert hauptsächlich das Format von Errormeldungen und der zweite formatiert dann die finale Ausgabe. Die Ausgaben werden jeweils an Variablen weitergegeben (json_output' und json_error') welche im weiteren Code verarbeitet werden.

Insbesondere json_error' ist für den weiteren Programmablauf von grosser Wichtigkeit. Wenn Borg ein Problem feststellt, wird die Error Meldung von borg an json_error' weitergegeben. Mittels der Methode im Codesnippet:(6), wird die Variabel ausgewertet und im Falle eines Fehlers wirft der Code eine Exception, welche im Hauptprogramm abgefangen wird. Dabei wird eine Fehlermeldung in einem separaten Fenster ausgegeben. Die Methode wurde dabei auf der Klasse BorgQtThread' umgesetzt und steht somit allen Funktionen zur Verfügung. Die Fehlermeldung bei einer fehlenden Konfigurationsdatei, Abbildung: (22), funktioniert nach dem gleichen Prinzip und konnte somit zum grössten Teil wiederverwendet werden. Der restliche json Output kann dann einfach mit dem json' Modul geparst werden. Somit werden dem User, gemäss Ziel Nr. 14, direkt die Fehlermeldungen von borg angezeigt und es muss nur an gewissen Stellen noch applikationsspezifisches Exception Handling betrieben werden.

```python
# borg_qt/borg_interface.py

def process_json_error(self, json_err):
    if json_err:
        error_list = json_err.split('\n')
        if "borg.locking" in error_list[0]:
            pass
        else:
            err = json.loads(error_list[0])
            raise BorgException(err['message'])
```

Programmlisting 6 Auswertung der json err Variabel.

Die ganze Funktionalität wurde dann in der Klasse `BorgQtThread' zusammengefasst. Somit kann für jede Funktion von borg eine einzelne Klasse geschrieben werden, welche dann von` BorgQtThread' die Funktionen erbt. Die Funktionsklassen müssen dann jeweils nur die Methode `self.create_command(self)' implementieren, welche das Property` self.command' erstellt und die einfachen Funktionen von borg sollten direkt funktionieren.

5.5 Backup

Daten zu sichern ist die primäre Funktion von Borg-Qt. Deshalb soll das Erstellen eines Backups so schnell und unkompliziert wie möglich vonstattengehen.

5.5.1 Backend

Um Backups erstellen zu können, wurde die Klasse `BackupThread' erstellt, welche von` BorgQtThread' erbt. Die Klasse `BackupThread' nimmt beim instantiieren 3 Argumente auf:` includes', `excludes',` prefix'. Wobei `excludes' und` prefix' beide optional sind. Im Hauptcode werden diese Argumente aus der Konfigurationsdatei ausgelesen und übergeben. Die Includes werden im Falle eines Backups im Hintergrund aus der Konfigurationsdatei gelesen. Wenn es der User manuell ausführt, wird der im Frontend ausgewählte Pfad mitgegeben.

Die „Excludes“ haben lange nicht funktioniert. Der Grund dafür waren zusätzliche Anführungszeichen um die Exclude Pfade. Diese wurden aus Versehen hinzugefügt, da borg normalerweise auf der Kommandozeile ausgeführt wird und die Anführungszeichen dort notwendig sind, um allfällige Leer- oder Sonderzeichen abzufangen. Es wurde davon ausgegangen, dass das `subprocess' Modul ähnlich funktioniert wie die Kommandozeile. Da man an das Modul direkt einen String übergibt, sind die zusätzlichen Anführungszeichen nicht notwendig und führen sogar dazu, dass die Pfade gar nicht funktionieren. Somit werden die „Excludes“ mittels der Methode` _process_excludes' mit dem entsprechenden Parameter gepaart und als gesamte Liste an das finale Kommando angehängt. Die „Includes“ funktionieren auf die gleiche Weise, benötigen jedoch keine zusätzlichen Parameter. Zu sehen ist dies im Codesnippet:(7).

```
# borg_qt/borg_interface.py
# Funktion zum Verarbeiten der "Excludes"
def _process_excludes(self, excludes):
    processed_items = []
    if excludes:
        for item in excludes:
            processed_items.extend(['-e', item])
        return processed_items
    else:
        return processed_items

# Methode zum Erstellen des gls:borg Kommandos.
def create_command(self):
    self.command = ['borg', 'create', '--log-json', '--json',
                    ('::'
                        + self.prefix
                        + '{now:%Y-%m-%d_%H:%M:%S}')]
    self.command.extend(self.includes)
    if self.excludes:
        self.command.extend(self.excludes)
```

Programmlisting 7 Erstellen des „borg create“ Kommandos fürs erstellen von Backups.

5.5.2 Frontend

Damit die Backups im Frontend funktionieren, musste zum einen der „Backup“ Knopf mit der Methode `create_backup' verknüpft werden. Des Weiteren wurde ein Dateibaum, in Abbildung:(23) grün umrahmt, eingefügt. Dieser gibt den Pfad des angewählten Objektes and die` create_backup' Methode weiter.

file:pictures/borgqt_file_tree.png

Während dem ein Archiv erstellt wird, wird ein kleiner Dialog mit Ladebalken angezeigt, Abbildung: (24). Dieser dient hauptsächlich dazu dem User das Gefühl zu geben, dass die Applikation noch am Arbeiten ist.

Der Dialog musste gegenüber der ersten Version in Sektion: [Erste Umsetzung] noch etwas angepasst werden. borg gibt, während dem Erstellen eines Archivs keine Informationen zurück, welche es einem erlauben würden einen Fortschrittsbalken zu generieren, welcher den effektiven Fortschritt anzeigt. borg gibt einzig die Anzahl der verarbeiteten Dateien in regelmässigen Abständen zurück. Da borg jedoch zu Beginn nicht meldet, wie viele Dateien gesichert werden, lässt sich damit keine Prozentzahl erstellen. Ein paar Experimente, bei denen die zu sichernden Dateien zuerst von Borg-Qt gezählt werden sollten, wurden verworfen. Einerseits weil keine Methode gefunden werden konnte, welche die gleiche Anzahl Dateien zurückgab wie borg. Anderseits, weil es den Backup Vorgang unnötig in die Länge zieht. Dies ist insbesondere der Fall, wenn sich sehr viele

Dateien im Quellverzeichnis befinden. Es kann sogar soweit kommen, dass das Zählen länger als das eigentliche Sichern dauert. Aus diesem Grund wurde der Fortschrittsbalken mit Prozentanzeige durch einen sich wiederholenden Ladebalken ersetzt.

file:pictures/borgqt_progress_v2.png

Wurde das Archiv erfolgreich erstellt, wird die Liste mit den Archiven sowie die Repository Statistik aktualisiert. Beide Elemente sind in der Abbildung:(25), grün respektive rot umrahmt. Für die beiden Funktionen wurde jeweils eine eigene Klasse, `ListThread' und` InfoThread', erstellt. Beide erben von `BorgQtThread'. In den Klassen wird wie bei` BackupThread' borg über einen `subprocess' aufgerufen, um die Archiv Liste respektive Statistik zurückzuerhalten Die json Strings werden wieder auf die jeweilige Information geparst und die Archive in eine Python Liste, die Repository Statistik, in Zahlen umgewandelt.

Da borg die Repository Grössen in Bytes zurückgibt, sollten diese zur Anzeige noch in eine Menschen lesbarses Format umgerechnet werden. In Borg-Qt geschieht dies mit der Helferfunktion `convert_size'. Die Funktion wurde von Stackoverflow [sizeformat] übernommen.

Beim Durchführen der Usability-Studie wurde noch ein Bug entdeckt welcher die Anwendung zum Abstürzen brachte. Der Bug, der entdeckt wurde, tritt immer dann auf, wenn ein Archiv gemountet ist während man ein Archiv erstellen möchte. Dies ist jedoch offenbar eine Funktion die von borg nicht unterstützt wird [borgmountissue]. borg kann mehrere Archive gleichzeitig mounten. Der User müsste jedoch jedes der Archive zuerst wieder unmounten bevor er eine neue Datensicherung erstellen kann. Das Problem wurde dadurch gelöst, dass dem User ein Dialog angezeigt wird, über welchen er vor einer Datensicherung zuerst die gemounteten Archive aushängen kann. Anschliessend startet die Datensicherung, wie wenn kein Archiv gemountet gewesen wäre.

file:pictures/borgqt_archive_list.png

[Erste Umsetzung] siehe Abschnitt 5.2.2

5.6 Restore

Der Code für das Wiederherstellen eines Archivs ist sehr ähnlich wie der Code für das Erstellen. Die Besonderheiten bei dieser Funktion sind vor allem die Kontrolle, dass ein Archiv angewählt wurde, bevor man die Wiederherstellung startet, das Erstellen des Zielpfades sowie das Aufräumen bei einem Fehler.

Wird der „Restore“ Knopf gedrückt ohne das ein Archiv angewählt wurde, erscheint folgende Fehlermeldung, Abbildung:(26), um den Benutzer darauf hinzuweisen, das er dies noch tun sollte.

file:pictures/borgqt_no_archive_selected.png

Für die Wiederherstellung einer Datensicherung, selektiert der User das gewünschte Archiv. Als zweiten Schritt startet er den Prozess mit Klick auf „Restore“. Im sich automatisch öffnenden Dialogfenster, ist der gewünschte Zielort auszuwählen. Ist der Zielort festgelegt, erstellt Borg-Qt ein Subverzeichnis mit dem Namen des Archivs und beginnt mit der eigentlichen Wiederherstellung. Ist der Zielort für die Applikation nicht beschreibbar erscheint die Fehlermeldung, Abbildung:(27), und

der Vorgang wird abgebrochen. Nach der erfolgreichen Wiederherstellung öffnet die Applikation den Zielort in einem Dateimanager, damit der User gleich mit den Dateien weiterarbeiten kann.

file:pictures/borgqt_not_writeable.png

Gibt es, während dem Wiederherstellen, einen Fehler gibt die Anwendung den entsprechenden Fehler aus und löscht zusätzlich noch den zu Beginn erstellten Archiv Ordner.

Wird das gleiche Archiv nochmal an den gleichen Zielort wiederhergestellt, werden bereits vorhandene Dateien überschrieben.

5.7 Mount

Die „Mount“ Funktion prüft zuerst ob der Benutzer ein Archiv angewählt hat und gibt, falls dies nicht der Fall ist, eine entsprechende Fehlermeldung aus. Im Gegensatz zur „Restore“ Funktion zeigt die „Mount“ Funktion jedoch keinen Dialog zum Auswählen des Zielpfades. Die Funktion erstellt sich diesen selbst. Der Zielpfad ist dabei kombiniert aus dem `/tmp' Verzeichnis und dem Namen des Archivs

borg mountet jedes Archiv nur mit Leserechten. Es ist relativ unwahrscheinlich, dass der Zielpfad in unbeschreibbarer Form bereits vor dem Ausführen der `mount_backup' Methode bereits vorhanden ist. Ist dies der Fall kann davon ausgegangen werden, dass der Benutzer das Archiv bereits einmal gemountet hat. Genau dies wird in der Applikation auch so überprüft. Hat die Applikation Schreibrechte auf den Zielpfad, wird das ausgewählte Archiv auf diesem Pfad gemountet. Anschliessend wird der Pfad in einem Dateimanager geöffnet, damit der Benutzer direkt mit den Dateien weiterarbeiten kann. Wurde erkannt, dass das Archiv bereits gemountet wurde, also der Pfad nicht schreibbar ist, öffnet die Applikation direkt den Dateimanager, ohne zu versuchen das Archiv noch einmal zu mounten.

Zusätzlich wird der Pfad jedes gemounteten Archivs in einer Liste gespeichert. Beim Beenden der Applikation iteriert die Applikation über jeden Pfad in der Liste unmountet das Archiv und löscht den Ordner. Somit befindet sich das System wieder im gleichen Zustand wie vor dem Start der Applikation.

5.8 Delete

Soll ein Archiv gelöscht werden wird, wie bei der „Restore“ und „Mount“ Funktion, überprüft ob eines angewählt ist. Ist dies gegeben, zeigt die Applikation dem Benutzer einen Dialog, Abbildung: (28). Bestätigt er diesen mit „Yes“ wird der Vorgang fortgesetzt und das Archiv gelöscht mit „No“ wird der Vorgang abgebrochen. Nach der Löschung werden die Archivliste und die Repository Statistik aktualisiert, um den neuen Zustand wiederzugeben.

file:pictures/borgqt_yes_no.png

5.9 Automatische Backups

Damit der Benutzer die Backups nicht von Hand machen muss, ist es sinnvoll eine Funktion bereitzustellen, welche die Backups automatisch im Hintergrund erledigt. Dadurch ist sichergestellt

das die Backups im allgemeinen Trubel des Lebens nicht vergessen gehen.

Voraussetzung für automatisierte Backups ist, dass die Datensicherung ohne gui gestartet werden kann. Bei Borg-Qt wird dies über einen Kommandozeilen Parameter realisiert. Hierfür wurde das Python Standard Paket `argparser' verwendet. Konkret bedeutet dies, dass wenn die Applikation auf der Kommandozeile mit folgendem Befehl ausgeführt wird:` borg_qt -B'. Wird die grafische Oberfläche nicht angezeigt und es wird direkt die Methode `background_backup' der Klasse` MainWindow' ausgeführt. Dabei werden alle Ordner, welche in den Einstellungen unter „Include" sowie „Exclude" gespeichert wurden, im Archiv gesichert, respektive davon ausgeschlossen. Damit sind die Voraussetzungen für automatische Backups gegeben.

Um die Backups in regelmässigen Intervallen auszuführen, gibt es zwei Möglichkeiten, wie man dies implementieren könnte.

Variante a), die Applikation permanent im Hintergrund laufen lassen, etwa als Trayicon wie man das von anderen Applikationen wie etwas Dropbox kennt.

Variante b) über Werkzeuge des Betriebssystems. Die drei Desktop Betriebsysteme, Windows, OS X und Linux bringen alle drei Werkzeuge mit, um periodisch ein Programm auszuführen.

Für Borg-Qt wurde beschlossen Variante b) zu realisieren. Die Anwendung soll dem Benutzer soweit als möglich aus dem Weg gehen. Eine Anwendung welche permanent in der Taskleiste lebt, erfüllt das Kriterum nicht.

Unter Linux wurden sich wiederholende Aufgaben, früher mit sogenannten Cron Jobs umgesetzt. Die moderne Lösung sind heutzutage jedoch Systemd Timer. Also konkret Systemd. Dies aus dem Grund das Systemd genau für das Managen von Systemdiensten programmiert wurde. Ein Grossteil der benötigten Funktion ist bereits in Systemd enthalten, somit kann bei der Entwicklung zusätzlich Zeit gespart werden.

Systemd ist ein init System, welches dazu dient dazu die Benutzerumgebung und die dazugehörigen Prozesse zu starten und zu verwalten [systemd]. Die Prozesse werden über sogenannte „Services" gestartet. Die Services werden dabei einfach in Klartextdateien mit der Dateiendung `.service' definiert. Der Inhalt orientiert sich dabei praktischerweise am „INI" Stil. In Borg-Qt wurde das INI Format bereits bei den Konfigurationsdateien verwendet. Somit können die dort gesammelte Erfahrungen zur Implementation von` configparser' wiederverwendet werden. Soll ein Service in einem gewissen Zeitintervall ausgeführt werden, benötigt Systemd eine weitere Datei mit dem gleichen Namen jedoch mit der Dateiendung `.timer' . Der Inhalt ist auch wieder im INI Stil gehalten. Systemd versteht eine Vielzahl an Datumsformaten [systemddate]. In Borg-Qt wurden zwei Varianten in den Einstellungen umgesetzt. Eine, welche „Predefined Schedule" genannt wurde und eine mit dem Namen „Custom Schedule", zu sehen in, Abbildung:(29). Die Predefined Option wird dabei in die von Systemd unterstützten Formate „hourly, daily, weekly und monthly" übersetzt. Entsprechend der gewählten Wiederholung wird automatisch ein Archiv erstellt. Mit der Custom Option kann der Benutzer sich den Zeitplan`

individueller gestalten. Etwa „jeden Mittwoch um 12:00 Uhr“ für Systemd übersetzt würde dieser Zeitplan dann so aussehen: Wednesday --* 12:00:00'. Für spätere Versionen von Borg-Qt wäre es allenfalls auch möglich die Auswahl von mehreren Wochentagen zu ermöglichen, damit der Benutzer etwa folgenden Zeitplan erstellen könnte „Montag, Mittwoch, Freitag stündliche Backups.“ (`Monday, Wednesday, Friday --* *:00:00').

file:pictures/borgqt_settings_schedule.png

Das Erstellen der eigentlichen Systemd Konfiguration passiert in Borg-Qt in der Config' Klasse zum gleichen Zeitpunkt, wie die eigentliche Konfigurationsdatei geschrieben wird. Zum Schreiben und de-/aktivieren des Systemd Services, respektive Timers wurde eine Klasse SystemdFile' erstellt. Somit könnte die Funktion auch einfach in einem anderen Projekt verwendet werden.

Systemd benötigt zum Starten der Anwendung den absoluten Pfad in der Service Datei. Da davon ausgegangen werden kann, das Borg-Qt im PATH' des Systems abgelegt wird, wurde das Unix Tool which' verwendet, um den exakten Speicherort zu erhalten. Mittels des Befehls which borg-qt' erhält man den absoluten Speicherort der Datei. Zusammen mit den Daten aus den Einstellungen wird diese Information in einem Configparser' Objekt gespeichert, welches dann mithilfe der SystemdFile' Klasse in eine borg_qt.service', Codesnippet:(8), respektive borg_qt.timer', Codesnippet:(9), Datei, im Systemd Pfad /home/username/.config/systemd/user/' geschrieben und aktiviert wird.

Eine Option in der Datei borg_qt.timer', die noch erwähnenswert ist, ist Persistent = true'. Ist Persistent' auf true' gesetzt, holt Systemd den Tasks nach sollte er eine Ausführung verpasst haben. Dies ist insbesondere dann hilfreich, wenn etwa der Zeitplan auf daily' oder weekly' gesetzt wurde. Sollte also etwa jeden Mittwoch ein Backup gemacht werden aber der Computer lief an diesem Tag nicht, startet Systemd Borg-Qt, sobald der Computer das nächste Mal eingeschaltet wird kommt.

```
#~/.config/systemd/user/borg_qt.service
[Unit]
Description = Runs Borg-Qt once in the backround to take a backup
according to the configuration.

[Service]
Type = oneshot
ExecStart = /home/andreas/bin/borg_qt -B
```

Programmlisting 8 Systemd Service Datei für Borg-Qt

```
 #~/.config/systemd/user/borg_qt.timer
 [Unit]
 Description = Starts the borg_qt.service according to the
configured schedule.

 [Timer]
 OnCalendar = hourly
 Persistent = true

 [Install]
 WantedBy = timers.target
```

Programmlisting 9 Systemd Timer Datei für Borg-Qt

5.10 GUI Anpassungen nach Usability-Studie

Im Rahmen der durchgeführten [Usability-Studie] machten die User folgenden Feststellungen: * „Include“ und „Exclude“ Funktionen sind unklar beschriftet. * Die Funktionen sind unklar. * Texte sind nicht einheitlich.

Die Korrekturen wurden im Rahmen der Diplomarbeit angepasst.

In den „Include“ sowie „Exclude“ Optionen wurden einige Buttons neu beschriftet und zwei Labels hinzugefügt, nun wird klarer auf ihre Funktion hingewiesen, Abbildungen:(30) und (31).

file:pictures/borgqt_settings_include_v2.png

file:pictures/borgqt_settings_exclude_v2.png

Um die Funktionen der Applikation zur erklären wurde ein Hilfe Fenster, Abbildung:(32), eingebaut. Dieses wird dem Benutzer beim Start der Applikation angezeigt. Es gibt ihm einen kurzen Überblick welcher Button welche Aktion auslöst und welche Elemente welche Information anzeigen. Optional kann der Benutzer entscheiden, dass er das Fenster beim nächsten Start nicht mehr angezeigt bekommen möchte. Über den Button „Help“ kann das Fenster jederzeit unabhängig der Einstellungen wieder angezeigt werden.

file:pictures/borgqt_help.png

Mit Fertigstellung der Anpassungen wurde die Realisierung erfolgreich abgeschlossen und die Entwicklung neuer Funktionen für den Zeitrahmen der Diplomarbeit gestoppt.

[Usability-Studie] siehe Abschnitt 4.5

5.11 Releases

Für die finale Veröffentlichung wird Borg-Qt als ein sogenanntes ausführbares „Binary“ zur Verfügung gestellt. Man kennt diese auf Windows Systemen etwa als die Dateien mit der Endung

`.exe'. Beim Binary handelt es sich um ein selbst entpackendes Dateiarchiv. Sämtliche benötigten Python Module und sonstige Dateien wie etwa die Icons oder gui Definitionsdateien sind darin enthalten.

Diese Art der Auslieferung hat den Vorteil, dass der User das Programm nicht speziell installieren muss oder dafür irgendwelche zusätzlichen Dinge installieren muss. Der Nachteil ist jedoch, dass ein solches Binary nur auf dem jeweiligen Betriebssystem erstellt und ausgeführt werden kann. Das heisst, dass man unter Linux etwa keine Binaries für Mac erstellen kann oder umgekehrt.

Das Binary wird mit dem Programm „PyInstaller"[pyinstaller] erstellt. PyInstaller wird auf der Kommandozeile mit Angabe der Hauptdatei des Codes ausgeführt. Der Befehl dafür ist relativ einfach, Codesnippet:(10). Pfade zusätzlicher Dateien wie etwa Icons müssen mit angegeben werden. PyInstaller kann diese sonst nicht in das Binary einbinden. Der gezeigte Code wurde dabei in ein Makefile implementiert. Somit kann man in der obersten Ebene des Repository einfach den Befehl `make' ausführen und das Binary wird im Ordner dist' erstellt.

```
pyinstaller --hidden-import=PyQt5.sip \
    --add-data=borg_qt/static/icons:static/icons \
    --add-data=borg_qt/static/UI:static/UI \
    -F borg_qt/borg_qt.py; \
```

Programmlisting 10 Code zum Erstellen der finalen Binaries von Borg-Qt

Auf Github wird jeweils ein Release erstellt und dazu die passenden Binaries hochgeladen. Github packt dabei den Source Code beim Erstellen des Releases in ein Zip Archiv. Somit steht der exakte Source Code zu jedem Binary direkt zu Verfügung. Dies um den Regeln der gpl zu folgen sowie um Benutzern die Möglichkeit zu geben den Code vor einer Nutzung zu überprüfen und als sicher zu befinden.

5.12 Kontrollieren der Testfälle

Am 25.02.2019 würden gemäss dem Zeitplan die Testfälle durchgegangen. Bis auf die Testfälle TC-18 und TC-24 konnten alle Testfälle erfolgreich durchgeführt werden. TC-18 und TC-24 konnten deshalb nicht durchgeführt werden, da diese Funktionen noch nicht implementiert wurden. TC-18 ist zwar technisch bereits möglich wurde im gui jedoch noch nicht umgesetzt. Die Resultate wurden in der Tabelle:(29) erfasst.

6 Ausblick/Fazit

6.1 Zielbewertung

In der nachfolgenden Tabelle:(25) wurden die Ziele nach „Erfüllt", „Nicht erfüllt" oder „Teilweise erfüllt" bewertet. Die Ziel-Nr. entspricht dabei der Ziel-Nr. in der Tabelle:(1). In der Spalte Bemerkung wird noch kurz ausgeführt, in welcher Form das Ziel bewiesen wurde.

6.1.1 Risikoanalyse der neuen Ist-Situation

Das Ist-Riskio in der Sektion [Risiko-Analyse] könnte wie prognostiziert erhehblich gesenkt werden. Entgegen der ursprünglichen Annahme konnten die automatischen Backups doch noch während der Diplomarbeit implementiert werden. Dadurch konnte auch das Risiko Nr. 5 (Der Benutzer vergisst Backups zu machen) erheblich gesenkt werden. Somit hat die neue Ist-Situation, Abbildung:(33), eine bessere Risikobewertung als das geplante [Soll-Risiko].

file:pictures/ist_risiko_neu.pdf

[Risiko-Analyse] siehe Abschnitt 3.3

[Soll-Risiko] See figure 5

6.2 Projektmanagement

Das zu Beginn erstellte Gantt Chart war sehr hilfreich, um den Überblick über das Projekt zu behalten. Anhand davon konnte immer abgeschätzt werden wie das Projekt etwa im Zeitplan steht. Durch die Annahme das an arbeitsfreien Tagen nur 6 Stunden gearbeitet wird, konnte unter der Woche auch mal ein Abend ausgesetzt werden, wenn am Wochenende zuvor einmal 8 Stunden gearbeitet wurde.

Die regelmässigen Arbeitsessions an jedem schulfreien Samstag und an den freien Montagen haben sich als eine gute Variante des Arbeiten erwiesen und haben die Last der Diplomarbeit gut verteilt.

Die konservativen Zeitschätzungen haben sich als korrekte Entscheidung erwiesen. In Zukunft kann noch stärker versucht werden die Zeit, welche für ähnliche Code Teile verwendet wird kürzer zu schätzen. Copy/Paste erlaubt unter Umständen enorme Zeitersparnise. Trotzdem sollte genügend Reserve für allfällige Herausforderungen eingeplant werden. In der Entwicklung kann es immer wieder vorkommen das man an einem Bug oder Ähnlichem hängen bleibt ohne das man es mit mehr Arbeitskräften oder Zeit schneller beheben kann.

Für die Planung der Programmierarbeiten haben sie die Use Cases und Aktivitätsdiagramme als eine grosse Hilfe erwiesen. Besonders um den roten Faden während der Entwicklung zu behalten waren sie sehr hilfreich.

6.2.1 Zeitaufwand

Der Zeitaufwand ist wie in Tabelle:(28) zu sehen, leicht tiefer als geplant. Dies ist vor allem auf die Zeitersparnisse bei der Realisierung zurückzuführen.

6.2.2 Ressourcen

Die im Projekt eingesetzten Ressourcen entsprechen den in der Tabelle:(26) geplanten 1:1.

6.2.3 Kosten

Da die geleisteten Stunden tiefer als geplant waren, sind auch die geschätzten Kosten tiefer. Wie in der Tabelle:(27) zu sehen würden sich die angenommenen, internen Kosten für die Arbeit auf ca. 17302.20 CHF belaufen. Dies stellt eine Einsparung von 1777.8 CHF dar.

6.3 Usability-Studie

Die Studie war eine sehr interessante Erfahrung. Enduser sehen eine Anwendung mit ganz anderen Augen als der Entwickler der Anwendung. Dieser weiss von jedem Element wie der Code dazu aussieht. Es wurde auch klar, dass die Aufgaben in einer Studie richtig gestellt werden müssen. Ansonsten wissen die Probanden schon gar nicht erst was von ihnen gefordert wird.

Auch sollte, wenn möglich, darauf geachtet werden, dass auf einem Betriebssystem getestet mit welchem die Probanden bereits etwas Erfahrung haben. Zwei der Probanden waren ab dem Verhalten und Aussehen des Dateimanagers von Ubuntu 18.04 etwas verwirrt. Sie hatten ihn zuvor weder gesehen noch genutzt. Alternativ könnte auch die Gruppe der Probanden so gewählt werden, dass sie mit dem Betriebssystem bereits vertraut sind.

Eine Usability-Studie ist auf jeden Fall etwas, was man bei zukünftigen Software Projekten, wieder machten sollte.

6.4 Umsetzung

Die Entwicklung mit Qt und Python ging sehr gut von der Hand. Die grafische Erstellung des gui mit dem Qt Designer war sehr hilfreich und hat es ermöglicht mit den Elementen auch einfach mal zu spielen, um zu sehen was passt und was eher weniger geht. Qt selbst ist ein gutes und sehr umfangreiches Framework. Das die Dokumentation den Fokus hauptsächlich auf C++ Code hat, machte es einem zu Beginn etwas schwieriger die gewünschte Information zu finden.

Python als Programmiersprache hat sich in der Entwicklung als gute Entscheidung bewährt. Die Entwicklung ging schnell und unkompliziert von der Hand. Python wurde, als eine sehr flexible Sprache wahrgenommen die einem bei der Entwicklung aus dem Weg geht.

Die eingesetzten unittest waren hilfreiche Werkzeuge bei der Entwicklung. Gegen Ende der Entwicklung wurden sie jedoch weniger eingesetzt da die Entwicklung davon doch etwas Zeit beansprucht und auch ein gewisses Mass an Erfahrung. Wenn man zuerst noch Recherchieren muss wie man den unittest schreibt verliert man wertvolle Zeit die man, während einer Diplomarbeit nicht zur Verfügung hat.

Diejenigen Tests die jedoch geschrieben wurden haben sich, als sehr hilfreich erwiesen einem ein Art Sicherheitsnetz zu bieten in dem sie aufgezeigt haben das nach einer grösseren Änderung immer noch alles so funktionert, wie es sollte. Es ist schade das unittest an der IBZ nicht gelehrt wurden. Sie scheinen definitiv etwas zu sein was jeder Programmierer beherrschen sollte und die Grundlagen für jedes Projekt, das über ein paar Zeilen hinausgeht, sein sollten. unittest werden mit Sicherheit eines der Hauptthemen zum Lernen in den nächsten paar Monaten sein.

6.5 Weiterverwendung von Borg-Qt

Borg-Qt wird als Freizeitprojekt des Projektleiters weiter geführt. Ein paar der Kann-Ziele wären noch sehr nützlich wären sie in der Applikation integriert. Zudem sind ein paar davon interessante Aufgaben um die eigenen Fähigkeiten zu erweitern. Nach Abschluss der Diplomarbeit wird die Entwicklung der Anwendung dann auch für Externe geöffnet. Vonseiten der borg Community kam

zum Zeitpunkt der Arbeit noch kein spezifisches Feedback zurück. Die Hoffnung besteht, dass sich allenfalls noch ein, zwei Interessierte finden, um am Projekt mitzuarbeiten.

Die Applikation wird aber auf jeden Fall vom Projektleiter produktiv eingesetzt, um seine sowie teilweise auch die Daten von Bekannten zu sichern.

7 Anhang

7.1 Ressourcen

7.2 Kosten

7.3 Zeitaufwand

7.4 Testfälle

7.7 Meeting Protokolle

Die Meeting Protokolle sind in der Datei [03_Meeting_Protokolle.pdf] zu finden.

[03_Meeting_Protokolle.pdf] file:03_Meeting_Protokolle.pdf

References