Voranalysebericht

Bash Vi, Shylux, Kaleb Tschabolt

March 1, 2011

Status	**In Arbeit**/In Prüfung/Abgeschlossen
Projektname	Projektexplorer
Projektleiter	Niklaus Hofer
Auftraggeber	M. Frieden, GIBB
Autoren	Niklaus Hofer
Verteiler	Lukas Knöpfel, Kaleb Tschabolt, Niklaus Hofer

Änderungskontrolle, Prüfung, Genehmigung

Version	Datum	Beschreibung, Bemerkung	Name oder Rolle
1	March 1, 2011		Niklaus Hofer, Projektleiter

Definitionen und Abkürzungen

Begriff/ Abkürzung	Bedeutung

References

[1] New module decisions for 3.0. http://permalink.gmane.org/gmane.comp.gnome.devel.announce/101, Jun 2010.

[2] Gnome foundation. Gnome activity journal. http://live.gnome.org/action/show/GnomeActivityJournal?action=show&redirect=GnomeZeitgeist, 2011.

[3] Wikimedia Foundation. Sqlite features. http://en.wikipedia.org/w/index.php?title=SQLite&oldid=415319458#Features, 2011.

[4] Apple Inc. Time machine. http://www.apple.com/macosx/what-is-macosx/time-machine.html, 2011.

[5] Google Inc. Google desktop. http://desktop.google.com/linux/, 2011.

[6] json.org. Xml definition. http://www.json.org/xml.html, 2011.

[7] orelly. Compiling python into java. http://docstore.mik.ua/orelly/other/python/0596001886_pythonian-chp-25-sect-3.html, 2011.

Contents

1 Zweck des Dokuments

In der Voranalyse geben wir einen groben Einblick in die künftigen Funkionen unseres Programmes. Hier werden die Ziele festgesetzt und Entscheidungen über die zu verwendeten Technologien begründet und festgehalten.

2 Ist-Aufnahme und Ist-Analyse

2.1 Beschreibung des Ist-Zustands (Ist-Analyse)

- Da das Projekt neu gestartet wird, existiert keine Vorarbeit, auf der wir aufbauen könnten.
- Einige der Ideen können aber mit denjenigen von Gnome Zeitgeist[2] und Apple timemachine[4] verglichen werden.
- Als bestehende Lösung könnte man die herkömmliche, hierarchische Struktur von Dateisystemen ansehen. Allerdings wollen wir diese nicht ersetzen, sondern lediglich um weitere Möglichkeiten ergänzen.

2.2 Schwachstellenanalyse

- Es existiert zwar kein Vorsystem, dessen Schwächen wir hier beschreiben könnten. Ich möchte hier aber auf die Schwächen herkömmlicher Dateiverwaltungssysteme eingehen.
- Diese haben vorallem das Problem, dass sich die Anwender herkömmlicher Dateisysteme eine Dateistruktur ausdenken und merken müssen. Sollten sie die Struktur jemals vergessen, so ist es schwierig Dateien zu finden. Auch ist das System recht unflexibel, wenn es darum geht Dateien, die zum gleiche Thema gehören, die aber weit über die hierarchische Struktur verteilt sind, zusammen zu verwalten.

2.3 Sicherheit

Es existieren keine Vorsysteme, die daher auch keine Sicherheitslücken aufweisen können.
Erwähnenswert scheint hier aber, einmal mehr, das cloud computing. Alle Daten bei einem storage Anbieter zu lagern löst nicht nur das Problem der Backups, sondern erlaubt auch das blitzschnelle Durchsuchen der Dokumente, da viele der online Anbieter (insbesondere Google mit seinem online Office) die Dokumente indexieren und so das Durchsuchen mit komplexen Algorithmen erlauben. All diese Vorteile haben aber einen entscheidenden Nachteil: Die Kontroller über die Dateien geht verloren. Der Storage-Anbiter hat, sofern man die Daten nicht manuell verschlüsselt, vollen Zugriff darauf. Zudem besteht hier der Nachteil dass ein Angreiffer alle die Daten bequem einsehen wenn er an das Passwort kommt.

2.4 Zukünftige Entwicklung

Das Gnome Zeitgeist Projekt[2] setzt einige innovative Ideen um, die das verwalten von Dateien komfortabler machen sollen. Dabei geht es darum, dass dokumentiert wird wann welche Dateien geöffnet wurden und welche anderen zur selben Zeit geöffnet waren. Es ermöglicht dann nicht nur die chronologische Ansicht der Dateien, sondern auch Abfragen wie “welches war der Song, den ich während dem Betrachten jener Bilder gehört habe?”.

- Welche Trends zeichnen sich ab und sind sie relevant?
  - Ein Trend ist dahingehend fest zu stellen, dass immer mehr Volltextsuchen für den Desktop zu Einsatz kommen, zum Beispiel Google Desktop search[5].
- Wie ist die Bereitschaft zu Veränderungen?
  - Die Bereitschaft grosse Änderunge ein zu gehen scheint nicht besonders gross zu sein.
    So wird das aktuelle System schon seit Jahrzehnten nahezu unverändert verwendet.
    Um die User nicht zu verwirren, vermeiden die Hersteller grundlegende Änderungen und verhalten sich eher zurückhaltend.
    Auch Gnome Zeitgeist wurde aus dem Gnome 3.0 Release gestrichen[1] und wird wohl vom Anwender selbst installiert werden müssen.
- Welche Anforderungen muss das System in Zukunft erfüllen?
  - Die User möchten Files mit anderen über die Ordner hinaus verknüpfen. Weg von der hierarchischen Struktur.

  - Das System muss in der Lage sein auch sehr grosse Mengen an Dateien so zu verwalten, dass einzelne Dateien schnell und ohne Umwege gefunden werden können, ohne dass sich der User lange Dateipfade merken muss.

- Wie wird sich das System entwickeln, wenn nichts geändert wird?
  - Die Betriebsysteme werden eingebaute Funktionen zum tagen mitbringen. Diese Funktion gibt es bereits in Gnome Zeitgeist.
  - Zum Versionieren gibt es heute schon einige einige Tools (meist Kostenpflichtig), die aber meist sehr undurchsichtig sind.
  - Bequeme backup-timeline-Lösungen ähnlich Apple's Zeitgeist werden wohl vermehrt eingesetzt werden, da sie das Erstellen und Verwalten von Backups vergleichsweise sehr einfach machen.
- Wo wurden ähnliche Probleme schon gelöst?
  - Windows 7 hat ein neues System zum erstellen 'virtueller' Ordner mitgebracht. Hier können Diese fassen mehrere Ordner und Dateien, unabhängig von deren Speicherort, in einem virtuellen Verzeichnis zusammen.
  - Es gibt schon einige Taggingsysteme, auch auf Basis des Filesystems.
  - Abgesehen von Gnome Zeitgeist hat sich seit Jahren nur sehr wenige getan bei der Verwaltung von Dateien.
- Welche Vorstellungen hat man von einer neuen Lösung?
  - In dem Bereich gibt es einige interessante Experimente und Untersuchungen, ein allgemeiner Tenor über eine 'Lösung der Zukunft' ist aber nicht in Sicht.
  - Evtl. wird sich das Problem auch von selbst lösen, falls die Verschiebung in die Cloud so von statten geht wie es sich einige Internetdienstleiste (Stichwort Google) vorstellen.
  - Auch aufwendige Dokumentenverwaltungssysteme könnten in Zukunft öfters zum Einsatz kommen, besonders beim Einsatz in Firmen, die grosse Datenmengen sinnvoll verwalten müssen.
- Was oder wer kann das zukünftige System beeinflussen?
  - Das System könnte durch zwei Dinge stark beeinflusst werden. Eine Möglichkeit wäre, dass ein Startup, oder ein freies Projekt eine Lösung bringt, die derart genial ist, dass sie sich durchsetzt und dann auch in der Industrie (Windows, OS X) integriert wird.
  - Die andere Möglichkeit wäre, dass der Branchenriese Microsoft eine Lösung in künftige Versionen von Windows integriert und so die Anwender auf ein neues System 'zwingt'.
- Welche kritischen Erfolgsfaktoren gibt es?
  - Das System darf nicht kompliziert sein und die Nutzer verwirren. Viele Nutzer hatten bereits Probleme damit das aktuelle System zu erlernen. Falls ein neues nicht sofort verständlich ist, so werden sie es ablehnen.

3 Systemziele

Bezeichnung	Beschreibung	Priorität (0-3)	Zielhierarchie	Zielkategorie	Kriterien zur Bewertung
Performance	Läuft das program schnell?	2			Das Programm sollte ähnlich schnell reagieren, wie die von den verschiedenen Systemen mitgelieferten Dateimanager. Keinenfalls darf sich die Benutzung so träge anfühlen, dass man lieber auf die Vorteile des Systems verzichtet.
Qualität	Kann das Programm einfach erweitert werden?	2			Das Programm ist objektorientiert aufgebaut. Die Benennung von Objekten und Variablen ist einheitlich. Das Programm ist klar und verständlich dokumentiert.
Sicherheit	Das Programm reist keine Sicherheitslücken in die Mechanismen der jeweiligen Betriebssysteme.	3			Firewalls und Rechteverwaltungssysteme werden nicht umgangen.
Kompatibilität	Läuft das Programm auf allen wichtigen Betriebssystemen.	2			Das Programm soll auf folgenden OS laufen: Windows, Linux, OS X
Datenkonsistenz	Die Metadaten werden ohne überflüssige Redundanze	1			Die Metadaten Ablage verhindert vom Design her widersprüchliche oder doppelte Angaben.

4 Fachlicher Soll-Zustand

4.1 Hauptaufgaben der neuen Lösung

- Hauptaufgaben
  - Dateien verwalten
  - Dateien können mit Tags versehen werden.
  - Dateien können nach Tags sortiert werden.
  - Operationen können auf eine Auswahl an Dateien (z.B. alle Dateien eines bestimmten Tags) angewandt werden.
  - Das Programm kann Dateien Versioniren.
- Informationsflüsse
  - Informationen zu den Dateien (Tags) werden dauerhaft abgelegt.
  - Werden Dateien mit Hilfe anderer Programme verschoben/erstellt, sollte das Programm (zumindest falls es läuft, vorzugsweise aber immer) davon in Kenntnis gesetzt werden.
- Schnittstellen nach aussen, zu externen Systemen
  - Das Programm muss Zugriff auf ein System haben, in dem es die Tags ablegen kann.
  - Zugriff auf das Dateisystem muss vorhanden sein, um das Verschieben von Dateien zu registrieren und um Dateien zu archivieren/versioniren.
  - Falls das Programm ein CLI erhält, so kann es natürlich mit Scripts bedient und automatisiert werden.

- Datenfluss

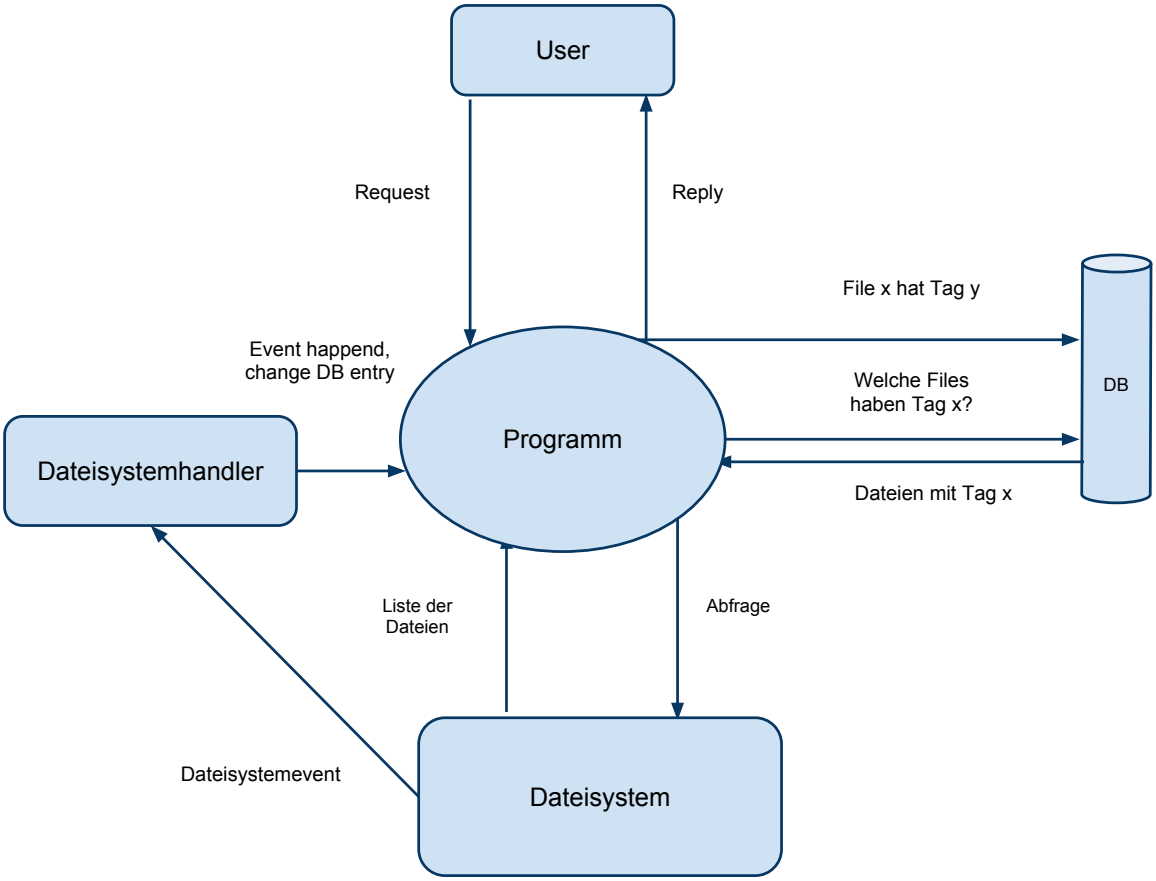

Figure 1: Datenflussdiagram

4.2 Übersicht über die erforderlichen Informationen / Daten

- vom Dateisystem
  - Name der Datei
  - Wo liegt die Datein (Verzeichnis)
  - Andere Metadaten wie Zugriffsrechte, Datum der letzten Veränderung, . . .
- von der Metadaten-Ablage
  - Welche Tags wurden der Datei X zugeordnet
- Dateisystem-Events
  - Datei(en) wird/werden gelöscht
  - Datei(en) wird/werden verschoben
  - Datei wird verändert

4.3 Anforderungen bezüglich Informationssicherheit und Datenschutz

- Informationssicherheit: Verfügbarkeit, Vertraulichkeit, Integrität
  - Die Verfügbarkeit des Programms hängt lediglich von der des Dateisystems ab und kann mit dieser gleichgesetzt werden.
  - Das Vertrauen des Users wird gewonnen, indem nichts nach aussen (internet) geschickt wird.

  - Die Integrität der Daten zu garantieren ist von daher eine Herausforderung, als dass das Programm mitbekommen muss, wenn Dateien verschoben werden und das registrieren muss. Ansonsten verweisen Metadaten ins Leere und Dateien verlieren die Ihnen zugeordneten Metadaten.
    Dieses Problem kann durch das Abhören von Dateisystem Events behoben werden.
    Umgehen liesse sich das Problem nur dann, wenn die Tags in den Filesystem-metadaten gespeichert würden.

- Datenschutz
  - Da das Programm lokal auf dem Computer läuft und nicht in die Zugriffsmechanismen des vorhandenen Systems eingreift geht die einzige Gefahr davon aus, dass der Speicherort der Tags für andere User frei zugänglich ist.
    Um das zu verhindern müssen die Tags entweder direkt in der jeweiligen Datei abgelegt werden oder, falls eine Datenbank zum Einsatz kommt muss diese in einem Verzeichnis abgelegt werden, auf das nur der jeweilige Benutzer Zugriff hat.

5 Lösungsvarianten

5.1 Zu verwendende Programmiersprache

5.1.1 Lösungsvariante 1: Java

5.1.1.1 Beschreibung der Lösungsvariante Bei dieser Lösung würde Java als Programmiersprache für das Projekt verwendet. Java hat den Vorteil, dass der Interpreter eine sehr grosse Verbreitung geniesst und alle beteiligten Entwickler mit der Sprache vertraut sind.

5.1.1.2 Struktur (Grobe Architektur) Java hat sehr viele Module, die nicht nur die Entwicklung erleichtern, sondern auf den Zugriff auf verschiedene Formate, die zum Abspeichern der Daten verwendet werden können.

5.1.1.3 Externe und interne Schnittstellen Wie bereits erwähnt bietet Java viele Schnittstellen zu gängigen Datenbanken und Dateiformaten an.
Zudem hat Java verschiedene Frameworks die zur Erstellung des GUIs verwendet werden können und sich zum Teil stark an den nativen Look des jeweiligen Systems anpassen.
Java's Ausgabe in die Konsole hingegen ist eher langsam.

5.1.1.4 Abdeckung der Anforderungen

- **Performance:** Java gilt heute als recht schnell.
- **Flexibilität:** Java gilt zur Zeit als die meist verwendete Programmiersprache überhaupt, was den Vorteil mit sich bringt, dass sehr viele Programmierer in der Lage sind damit geschriebene Programme weiter zu entwickeln oder an zu passen.
- **Sicherheit:** Dadurch dass Java code in der JVM ausgeführt wird, wird das ohnehin geringe Sicherheitsrisiko, dass durch den Einsatz der Software entsteht, noch weiter verringert.
- **kompatibilität:** Java ist für nahezu alle computer (egal ob Desktop oder handheld) verfügbar, in verschiedenen Varianten, die teils selbst den hohen Offentheitsansprüchen des Debian Projektes genügen. Die Verwendung vieler verschiedener Interpreter bringt aber auch Probleme. So sehen etwa Programme die für Sun/Oracle Java entwickelt wurden im Betrieb unter IcedTea etwas anders aus.

5.1.1.5 Realisierbarkeitsbetrachtung Die beteiligten Programmierer haben alle bereits Erfahrungen mit Java. Zudem findet man im Internet schnell Hilfe zu fast allen Bereichen der Programmiersprache.
Die Realisierbarkeit ist also durchaus gesichert wenn Java zum Einsatz kommt.

5.1.2 Lösungsvariante 2: Python

5.1.2.1 Beschreibung der Lösungsvariante Zur Implementierung des Programmes würde hier Python verwendet. Die moderne Programmiersprache kommt heute immer öfters zum Einsatz und läuft auf allen gängigen Betriebssystemen.

5.1.2.2 Struktur (Grobe Architektur) Auch Python bietet Schnitstellen zu fast allen wichtigen Formaten und Datenbanken. Zudem bietet Python aber noch den Vorteil, dass es auch sehr viele low-level-Schnittstellen gibt, die auch sehr hardware (oder, hier von grösserem Interesse, Dateisystem)nahe Operationen erlauben.

5.1.2.3 Externe und interne Schnittstellen Python braucht ebenfalls einen Interpreter. Dieser ist aber für alle gängigen Betriebssysteme verfügbar. Zudem kann Python code auch in der weit verbreiteten Java Virtual Machine ausgeführt werden[7].

5.1.2.4 Abdeckung der Anforderungen

- **Performance** Python gilt als sehr performante Programmiersprache. Die Performance würde für unseren Einsatzzweck vollauf genügen.
- **Flexibilität** Python hat ein ausgeprägtes Objekt Modell und ist gut Verständlich. Zudem steigt die Zahl an Python Entwicklern seit einger Zeit sehr rasch. Auch eine in Python geschriebene Implementierung könnte also später gut von einem anderen Programmierer erweitert oder geändert werden.

- **Kompatibilität** Der Python Interpreter hat eine deutlich weniger grosse Verbreitung als Java, ist aber auf Unix-artigen Systemen meist enthalten und kann unter Windows einfach installiert werden. Es lässt sich zudem spekulieren, dass Oracles aktueller Umgang mit Java und insbesondere mit der community der Verbreitung von Java starch schaden wird, wodurch dessen Verbreitung evtl. hinter die von Python zurückfallen könnte.

5.1.2.5 Realisierbarkeitsbetrachtung Der Einsatz von Python hat den Nachteil, dass die beteiligten Entwickler damit weniger vertraut sind als mit Java. Allerdings ist dadurch der Lerneffekt grösser. Zudem bietet Python mächtige Module, die die Arbeit vereinfachen.
Python ist auch insbesondere für die Entwicklung von GUIs gut geignet. Unter Linux gibt es eine Vielzahl von Projekten, die lediglich ein Python GUI für Tools darstellen, die ansonsten nur für die Konsole verfügbar sind.

5.2 Zu verwendende Ablage für die Metadaten

5.2.1 Lösungsvariante 1: SQLite

5.2.1.1 Beschreibung der Lösungsvariante ese Lösungsvariante sieht vor, dass die Daten in einer SQLite Datenbank abgelegt würden. SQLite scheint als Datenbank besonders gut geeignet, da alle Daten in lediglich einem File abgelegt werden. SQLite ist sehr schnell, unterstützt aber trotzdem alle standard SQL Befehle. SQLite wird auch in populären Software Projekten wie Firefox und Adobe Air eingesetzt[3].

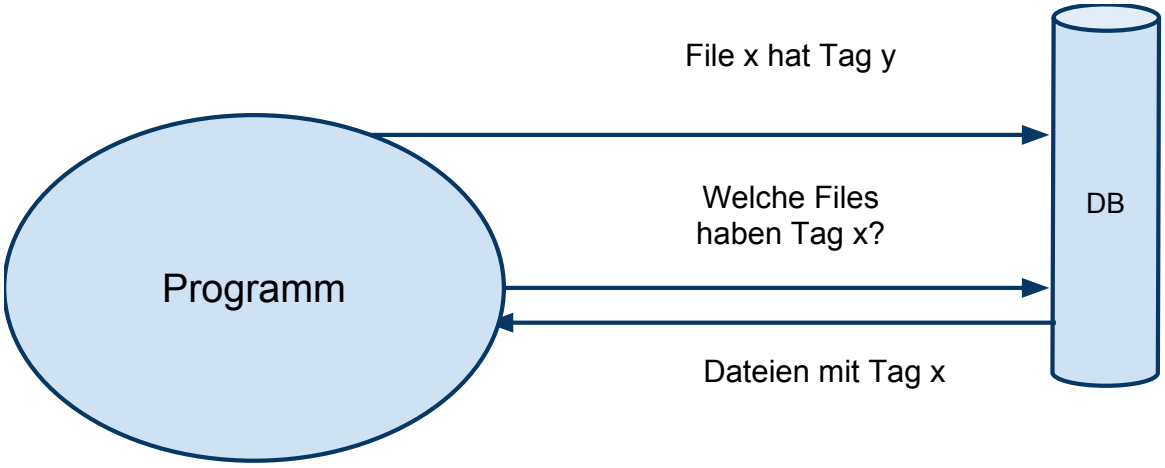

Figure 2: Datenflussdiagram beim Einsatz von SQLite

5.2.1.2 Struktur (Grobe Architektur)

5.2.1.3 Externe und interne Schnittstellen Dank der weiten Verbreitung von SQLite wird die leichtgewichtige Datenbank von fast allen Programmiersprachen unterstützt. Wie bei anderen Datenbanken auch, lassen sich darüber SQL Befehle absetzen, was das Speichern und Abfragen von Daten verhältnismässig sehr einfach macht.

5.2.1.4 Abdeckung der Anforderungen

- **Performance** Performance: SQLite gilt als sehr schnelle Datenbank mit guten Caching. SQLite kann als die schnellste der hier vorgestellten Lösungen angesehen werden.
- **Flexibilität** Flexibilität: Ein sauberes Datenbank-Layout lässt sich einfach erweitern, zudem wird SQLite von sehr vielen Programmiersprachen unterstützt, das macht diese Lösung sehr flexibel.
- **Sicherheit** Sicherheit: Durch die Verwendung von SQLite würde kein weiteres Sicherheitsrisiko entstehen. Zwar besteht hier theoretisch das Risiko der SQL-injection. Da aber jeder User eine eigene Datenbank verwenden würde könnte auch niemand sonst code einschleusen. Umgekehrt würde das Einschleusen von Code wenig sinn machen, da ja kein anderer User die Datenbank jemals verwenden würde.
- **Kompatibilität** Kompatibilität: SQLite ist für alle Betriebssyseme verfügbar.

- **Datenkonsistenz** Datenkonsistenz: Da Datenbanken speziell für das Vermeiden von Datenkonsistenz entworfen wurden, kann diese Lösung hier besonders punkten. Natürlich müsste ein vernünftigen Datenbank-layout angelegt werden.

5.2.1.5 Realisierbarkeitsbetrachtung Die beteiligten Programmierer sind mit SQL vertraut. Zudem ist das Ansteuern von SQLite mit Python und Java sehr einfach. SQLite wäre sicher am einfachsten um zu setzen.

5.2.2 Lösungsvariante 2: XML

5.2.2.1 Beschreibung der Lösungsvariante Diese Lösungsvariante sieht vor, dass die Metadaten in einem XML file abgelegt würden.
Dieses würde festhalten welche Datei (path und filename) welche Tags zugeordnet bekommen hat.

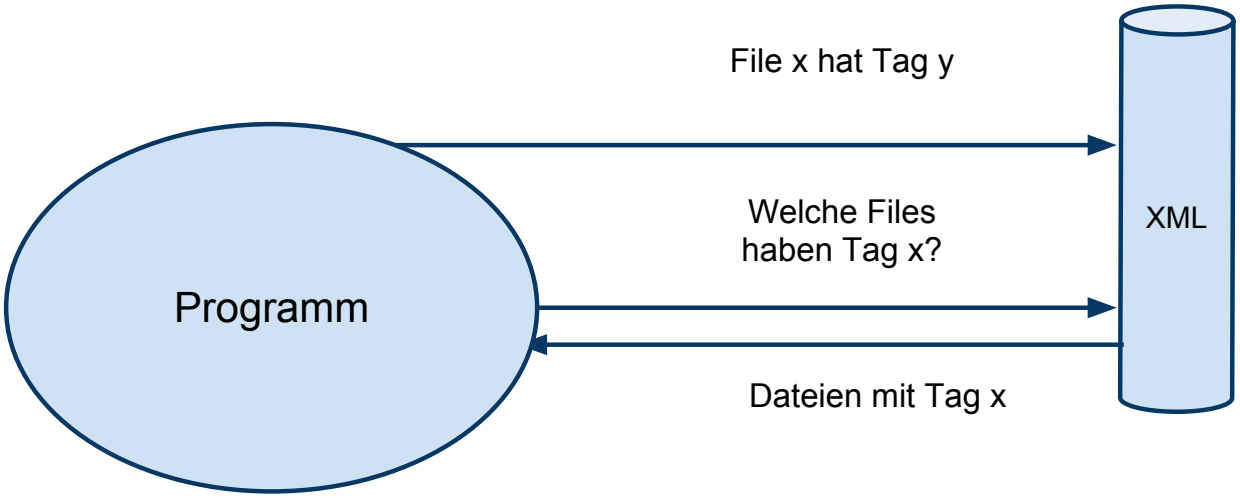

Figure 3: Datenflussdiagram beim Einsatz von XML

5.2.2.2 Struktur (Grobe Architektur)

5.2.2.3 Externe und interne Schnittstellen So gut wie alle modernen Programmiersprachen haben Schnittstellen zu XML files die das Lesen und Schreiben der in XML Form abgelegten Daten einfach gestalten. XML gilt aber auch als schwerfällig und umständlich[6].

5.2.2.4 Abdeckung der Anforderungen

- **Performance** Performance: Die Geschwindigkeit der XML Verarbeitung gilt als sehr gut und kann je nach Szenario durchaus mit der einer SQlite Datenbank mithalten.
- **Flexibilität** Flexibilität: XML ist sehr flexibel und erlaubt auch das Erstellen eigener Definitionen. Zudem gibt es heute sehr viele Programmierer, die mit XML vertraut sind. XML brächte auch den Vorteil, dass die nurtext-Dateien auch ausserhalb des Programms problemlos gelesen und verarbeitet werden könnten.
- **Sicherheit** Sicherheit: Bei der Verwendung von XML entstünden keine weiteren Sicherheitsprobleme.
- **Kompatibilität** Kompatibilität: Da die Daten in XML Files abgelegt wären, würde es keine Rolle spiele auf welchem System das Programm läuft. Die Kompatibilität und Portabilität wird also durch XML durchaus erhalten.
- **Datenkonsistenz** Datenkonsistenz: Bei sauberem Ausarbeiten eines Layouts zum Abspeichern der Daten liesse sich mit XML die Datenkonsistenz problemlos erhalten.

5.2.2.5 Realisierbarkeitsbetrachtung Die beteiligten Programmierer sind mit dem Konzept von XML nicht weiter vertraut und müssten dieses zuerst erlernen. Besonders da XML zur Zeit stark von JSON bedrängt wird scheint das aber kaum sinnvoll.
Trotzdem wäre das Projekt bei Verwendung von XML sicher realisierbar.

5.2.3 Lösungsvariante 3: Metadaten des Dateisystems

5.2.3.1 Beschreibung der Lösungsvariante Eine Möglichkeit wäre es, die Daten zusammen mit der Datei auf das Dateisystem zu schreiben. Das bringt den Vorteil, dass die Informationen fest mit der Datei verbunden sind.

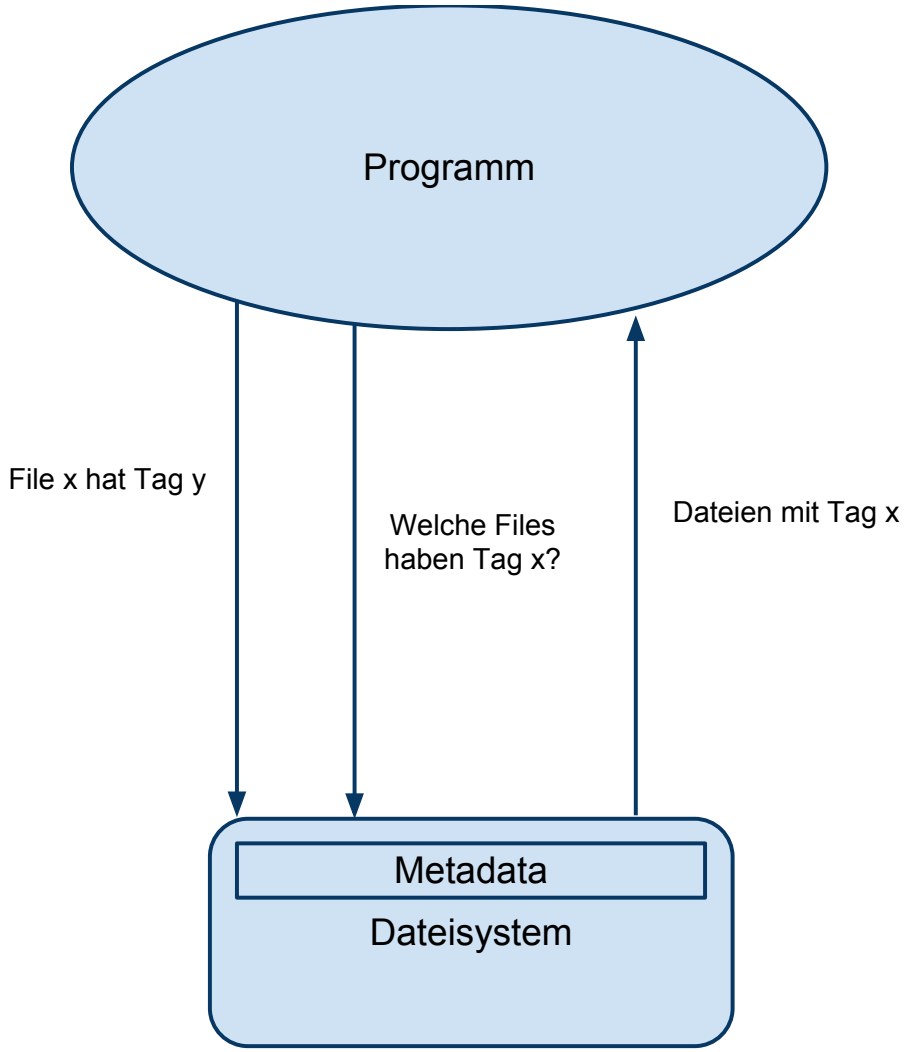

Figure 4: Datenflussdiagram beim Speichern der Metadaten auf das Daeisystem

5.2.3.2 Struktur (Grobe Architektur)

5.2.3.3 Externe und interne Schnittstellen Die Schnittstellen die genutzt würden liegen beim Dateisystem. Das bringt das Problem mit, dass das Programm nicht auf allen Dateisystemen gleich arbeiten kann. Besonders in der Unix/Linux Welt wo es eine grosse Zahl verschiedener Dateisysteme gibt wäre das ein Problem.

5.2.3.4 Abdeckung der Anforderungen

- **Performance** Performance: Wir haben einige Bedenken der Performance gegenüber. Da die Metadaten auf dem Dateisystem gespeichert würden, hätten wir keinen globalen Überblick darüber wo überall Dateien eines bestimmten Tags abgelegt sind und es müsste jedesmal eine Suche über das ganze Dateisystem laufen. Um das zu umgehen müsste eine zusätzliche Datenbank angelegt werden. Diese müsste aber mit den Daten des Dateisystems übereinstimmen und würde Redundanzen verursachen. Zudem würde das zusätzliche Betreiben einer Datenbank den Programmieraufwand erheblich steigern.
- **Flexibilität** Flexibilität: Diese Lösung hätte den Nachteil, dass sie für verschiedene Dateisysteme angepasst werden müsste.

- **Sicherheit** Sicherheit: Falls diese Lösung verwendet würde würde evtl. auch eine Sicherheitslücke entstehen. Würden Daten auf einen externes Speichermedium kopiert, so würden die Metadaten mitkopiert. Werden aber Daten für eine Drittperson kopiert, so ist es nicht immer Wünschenswert wenn diese die Tags zu Gesicht bekommt.
  Man müsste also auch hier eine Lösung finden wie dieses Problem umgangen werden kann.
- **Kompatibilität** Kompatibilität: Grundsätzlich liesse sich diese Lösung wohl für die meisten Systeme realisieren. Sie müsste aber nicht nur für jedes System, sondern sogar für jedes Dateisystem angepasst werden.
- **Datenkonsistenz** Datenkonsistenz: Die Lösung mit den Metadaten würde eine recht gute Datenkonsistenz schaffen, besonders da die Tags nicht verloren gingen, wenn eine Datei verschoben wird.
  Wie oben bereits erwähnt müsste aber wohl eine zusätzliche Datenbank angelegt werden, damit Suchbefehle auch in einem vernünftigen Zeitrahmen durchgeführt werden könnten. Diese zusätzliche Datenbank würde die Datenkonsistenz gefährden, vorallem da sie laufend mit den Daten des Dateisystems abgeglichen werden müsste.

5.2.3.5 Realisierbarkeitsbetrachtung Der Aufwand zur Realisierung dieser Variante währe weit grösser als bei den anderen Varianten. Nicht nur würde der Zugriff auf das Dateisystem, der sehr tief geht, implementiert werden, dieser müsste auch noch für verschiedene Systeme und Dateisysteme angepasst werden. Zudem müsste eine zusätzliche Datenbank angelegt werden, die laufend mit den Daten des Dateisystems abzugleichen wäre.
Diese Variante würde die Realisierung des Projektes stark gefährden!

6 Entscheid über die Lösungsvarianten

6.1 Entscheidung zur zu verwendenden Programmiersprache

Kriterium	**Wichtigkeit**	**Java**	**=**	**Python**	**=**
Plattformunabhängig	3	3	9	2	6
Kosten	1	3	3	2	2
Verbreitung des Interpreters	2	3	6	1	2
Geschwindigkeit	1	2	2	3	3
Einfachheit des Dateisystemzugriffs	1	1	1	3	3
Sofort ausführbar	1	1	1	3	3
Lernwert	2	1	2	3	6
Offenheit des Systems	2	2	4	3	6
Gesammt	-	-	28	-	31

Anhand der oben abgebildeten Tabellen sind die jeweiligen Kriterien und die zugehörigen Bewertungen der Lösungsvarianten ersichtlich.
Zu trotzdem möchten wir hier noch einige Punkte hervorheben.
Bei den Programmiersprachen fiel die Wahl auf Python. Ausschlaggebend war hier, neben den genannten Kriterien, auch die Popularität der Sprache. Java ist zwar nach wie vor die meistverwendete Sprache, Python gewinnt aber sehr schnell an Beliebtheit und hat eine grosse und aktive User community.
Diese Beliebtheit und die aktuelle Nachfrage nach Python-Programmierern lässt vermuten, dass das Beherrschen der Scripting-Sprache für die spätere Berufssuche ein Vorteil sein könnte.

6.2 Entscheidung zur zu verwendenden Ablage für die Metadaten

Kriterium	Wichtigkeit	SQLite	=	Metadaten	=	XML	=
Plattformunabhängigkeit	3	3	9	1	3	3	9
Datenbeständigkeit	3	0	0	3	9	0	0
APIs	3	2	6	0	0	1	1
Speicherbeanspruchung	1	2	2	3	3	1	1
Geschwindigkeit	1	3	3	1	1	2	2
Gesammt	-	-	20	-	16	-	13

Anhand der oben abgebildeten Tabellen sind die jeweiligen Kriterien und die zugehörigen Bewertungen der Lösungsvarianten ersichtlich.
Die Wahl der Speichermöglichkeit für die Tags fiel wesentlich einfacher als die Entscheidung für eine Programmiersprache. Zwar wäre das Verwenden der Filesystem-Metadaten sicherlich die technisch gesehen beste

Lösung. Grösster Vorteil wäre, dass die Daten beim Kopieren der Datei in ein anderes Verzeichnis nicht verloren gingen.
Der Zugriff auf diese Metadaten ist aber recht kompliziert und erfordert tiefgehendes Verständnis des Filesystems. Auch ist nicht sichergestellt, dass diese Methode auf allen Dateisystemen eingesetzt werden kann.
SQLite bietet den Vorteil, dass gewöhnliche SQL-Befehle verwendet werden können. Die starke Auslegung unserer Ausbildung und unsere beruflichen Tätigkeiten haben dazu geführt, dass wir mit SQL gut vertraut sind.
Zudem gilt SQLite als sehr schnell.

7 Mittelbedarf

7.1 Sachmittel

Sachmittel werden, neben den zur Verfügung stehenden Computern, keine benötigt.

7.2 Personal

Das Personal ist mit drei Programmierern vorgegeben.

7.3 Ausbildung

Mindestens zwei der Programmierer müssen sich Kenntnisse und Fähigkeiten im Umgang mit Python im Selbststudium aneignen. Zudem muss der Umgang mit SQLite erlernt werden.

7.4 Dienstleistungen

Zur Zusammenarbeit und Recherche muss bei jedem Entwickler eine funktionierende Internetverbindung vorhanden sein. Zudem werden zur Kommunikation und Zusammenarbeit die Officedienste und das Codeverwaltungstool von Google in Anspruch genommen. Um reine Textdateien zusammen zu bearbeiten benutzen wir Gobby.

8 Planung und Organisation

8.1 Projektorganisation

Das Team besteht aus zwei Programmierern und einem Projektmanager (PM).
Die Aufgabe des PMs wird im Verlaufe des Projektes weitergegeben.
Sollte der PM mit seiner Tätigkeit nicht ausgelastet sein, so wird er die Programmierer unterstützen.
Wahrscheinlich werden sich auch einige Spezialisierungen herausbilden. Es ist zum Beispiel denkbar, dass eine Person für das Datenbankdesign zuständig sein wird. Auch die Anpassung/Optimierung an die verschiedenen Betriebssysteme wird wohl jeweils einer einzelnen Person zufallen.

8.2 Anwenderorganisation

Die Anwender sind Einzelpersonen, die an einem lokalen Rechner arbeiten. Mehrere Personen könnten über freigegebene Ordner auf die Dateien zugreifen.

8.3 Termine

Die Termine sind von der Lehrkraft vorgegeben und im Terminplan eingetragen.

8.4 Prioritäten

Priorität hat die Schaffung einer GUI Version und die Dokumentation und die Überschaubarkeit des Codes.

9 Wirtschaftlichkeit

Vertrieb des Programms aussprechen wird, wird im Rahmen des Projektes keinerlei Umsatz generiert werden können. Ausserhalb des Projektes ist es natürlich möglich die von diesem Programmierer beigesteuerten Codeteile zu entfernen und das Programm zu vertreiben.
Ziel ist es, das Programm unter einer GPL oder BSD Lizenz zu veröffentlichen.
Es ist aber davon auszugehen, dass das Endprodukt bei der Verwendung einige Vorteile gegenüber herkömmlichen Dateimanagern bietet und dadurch Zeit eingespart werden kann. Das hängt vom Einsatzzweck ab und die dadurch entstehenden Kosteneinsprarungen könne nicht vorhergesagt werden.
Für uns bietet diese Projekt einen hohen Lernwert. Da wir selbst zu der Zielgruppe des Projekts gehören und es auch für uns optimieren können wir das Programm gut einsetzten.

10 Konsequenzen

10.1 Auswirkungen (organisatorisch, personell, baulich, Vorschriften/Weisungen)

Es stehen neue Wege zum Organisieren und Finden von Dateien zur Verfügung.
Das Organisieren von Projekten die sich über mehrere Ordner verteilen wird vereinfacht.
Die Beteiligten sammeln Erfahrungen mit den im Projekt eingesetzten Technologien.

10.2 bei Nichtrealisierung

Schlechte Note.

10.3 bei verspäteter Realisierung (gegenüber Wunschtermin)

Schlechte Note.

10.4 auf Schnittstellen zu anderen Systemen

Das Programm ist für den lokalen offline Einsatz vorgesehen. Onlinemodul falls wir genug Zeit haben.

10.5 Qualitätsverbesserungen

Einfacherer Umgang mit Dateien. Mehr Möglichkeiten zur Verwaltung von Dateien auf dem lokalen Betriebssystem.

10.6 Risikobeurteilung

Da das Team viel neues lernen muss und das Projekt recht viele einzelne Teile beinhaltet ist das Risiko, dass es nicht gänzlich fertiggestellt werden kann als beachtlich einzustufen.
Wenn das Programm keine Vorteile gegenüber anderen Dateiverwaltungstools beinhaltet, gerät das Programm in Vergessenheit.

10.7 Ausweichmöglichkeiten

Herkömmliche Dateimanager (explorer,bridge von adobe,ls,cp,rsync,mv,rm,find,locate).

11 Antrag auf Freigabe der nächsten Projektphasen

11.1 bisherige Entscheide

Als Programmiersprache haben wir Python bestätigt und als Datenbank haben wir SQLite gewählt.

11.2 Formulierung des Antrags

Wir haben uns eingehend mit den Zielen des Projekts befasst und eine realisierbare Lösung gewählt. Wir bitten sie deshalb den Antrag für die neue Phase zu genehmigen.