# KeePass

Password Safe

Version 0.3

| | |
|---|---|
| Jeroen Brandsma | s2178702 |
| Marco Gunnink | s2170248 |
| Daan Mauritsz | s2586274 |
| Ynte Tijsma | s2253216 |

**university of groningen**

# Revision History

| Version | Author | Date | Description |
| --- | --- | --- | --- |
| 0.1 | Daan Mauritsz | 12-12-2015 | Initial setup document |
| 0.1 | Jeroen Brandsma | 13-12-2015 | Introduction |
| 0.1 | Daan Mauritsz, Ynte Tijsma & Marco Gunnink | 13-12-2015 | Requirements |
| 0.2 | Daan Mauritsz | 3-1-2016 | Requirements feedback, rewrite |
| 0.2 | Jeroen Brandsma | 3-1-2016 | Introduction feedback processed |
| 0.2 | Ynte Tijsma & Marco Gunnink | 3-1-2016 | Start software architecture |
| 0.3 | Daan Mauritsz & Jeroen Brandsma | 10-1-2016 | Software architecture |
| 0.3 | Ynte Tijsma & Marco Gunnink | 10-1-2016 | Patterns |
| 0.3 | Marco Gunnink | 10-1-2016 | Review |

# Contents

# 1 Introduction

This section outlines the KeePass system. In the following sections the system will be explained in more detail. The system will also be evaluated based on the patterns used.

## 1.1 The system

Nowadays people need to remember many passwords. A password is needed to log in a PC, your e-mail, social media accounts and many more. Even though security experts urge people to use different passwords for each service, this is often neglected due to the sheer number of passwords needed. This causes a great danger when a service is attacked and user information is stolen, since the attackers can then gain access to any account that uses the same password.

To overcome this problem, KeePass was developed. KeePass is a free open source password manager, which helps people to manage their passwords in a secure way. All passwords can be stored in just one database, which can be secured with multiple credential methods. This means only one password needs to be remembered. The database is encrypted with the most secure encryption algorithms (AES and Twofish) currently known.

## 1.2 Versions

KeePass is under development in 2 different versions. The versions 1.x and 2.x are developed and maintained in parallel. They are not based on each other and offer different functionality and portability.[1] The most recent version of the 1.x branch is version 1.30 was released on the 2nd of January 2016. Version 1.x is a lightweight version that offers less functionality than 2.x. Version 2.30 was released on the 9th of August 2015. This version is able to run on multiple operation systems and offers extended functionality. Both versions are published under the GNU General Public License version 2.0 (GPLv2). This means that other developers are free to change and redistribute the software.

## 1.3 Community

According to the founder and main developer of KeePass, Dominik Reichl, the main reason his program became popular is the easiness of improving and adding functionality by the community. He didn't develop every part of the program himself. He worked together with other developers and also gave the community opportunities to contribute to the program. The community is very active in the forums on the SourceForge page of KeePass. [3] Here people who need help can get support with their problems. Also people with new ideas or suggestions can freely discuss here with the developers of KeePass.

# 2 Requirements

In this section the stakeholders will be identified and their concerns will be described. Secondly the key drivers will be determined.

## 2.1 Quality attributes

The stakeholders will now be identified. The concerns of each stakeholder will be described using the quality attributes defined in ISO 25010 *citation needed*. The quality attributes that will be used are defined in Table 2.

| Standard | Quality attribute |
|----------|-------------------|
| ISO 25010 | Performance Efficiency |
| | Compatibility |
| | Usability |
| | Reliability |
| | Security |
| | Maintainability |
| | Portability |

Table 2: Used definitions of quality attributes.

## 2.2 Stakeholders

The following stakeholders are distinguished and their concerns are described with the quality attributes listed in Table 2.

### 2.2.1 High priority stakeholders

The high priority stakeholders are the stakeholders that have a big impact on the system.

**Dominik Reichl** - Dominik Reichl is the creator, owner and main developer of the KeePass system. His main concerns are security, usability, compatibility and maintainability of the system.

**Bill Rubin** - Bill Rubin is a main developer of the system. His main concerns are security, usability and maintainability of the system.

**Other KeePass developers** - The developers are responsible for the development of the product and maintenance of the system. Just like the end users and main developers, they are mainly concerned with the usability, security, compatibility and portability of the product. Furthermore they are also concerned with the maintainability, because they will have to adapt and maintain the product.

**End users** - The end users will make use of the software product to store their passwords safely. Their main concerns are usability, security, compatibility and portability of the product.

### 2.2.2 Lower priority stakeholders

Lower priority is given to the less important stakeholders of the system, the system is less reliable on these stakeholders and their concerns.

**Plug-in developers** - KeePass promotes plug-in development for the system, the plug-in developers are the people that work on plug-ins for the KeePass system. Their main concerns are security, compatibility and maintainability of the system.

## 2.3 Concerns

Table 3 shows the concerns of the stakeholders. The first column shows the stakeholders relevant for this software product. Each stakeholder is assigned a number of points. These points are used to give priority to the relevant quality attributes. The more points are given to a certain quality attribute the more important this quality attribute is. An important thing to note is that only the high priority stakeholders are taken into account and not every stakeholder has the same amount of points to spend, because the stakeholders are equally significant. This is our own interpretation of the concerns.

| Stakeholders | Points | Performance Efficiency | Compatibility | Usability | Reliability | Security | Maintainability | Portability |
|---|---|---|---|---|---|---|---|---|
| Dominik Reichl | 100 | 5 | 20 | 15 | | 50 | 5 | 5 |
| Bill Rubin | 50 | | 10 | | | 20 | 10 | 10 |
| Other KeePass developers | 50 | | 5 | 5 | | 20 | 20 | |
| End users | 100 | | 20 | 30 | 10 | 30 | | 10 |
| **Total:** | 300 | 5 | **55** | **50** | 10 | **120** | 35 | 25 |

Table 3: Table containing point distribution and stakeholder concerns.

## 2.4 Key drivers

Using Table 3 we can determine the key drivers for this software product. The key drivers are the quality attributes that are assigned the most points. The key drivers we derived are:

1. **Security** - the main functionality of the KeePass system is providing security for the user by managing passwords. This is sensible information and therefore logically security is a key driver for the system.

2. **Usability** - the system should be easy in use since the frequency that it will be used, every time a user enters a password somewhere the KeePass system has to take over. Thus the usability is an important aspect of the system.

3. **Compatibility** - it is important that the system works with many different websites, browsers, etc. Therefore compatibility is a key driver.

# 3 Software Architecture

This section will explain the software architecture of KeePass. This architecture will be outlined using 2 views, the logical and the process view.

## 3.1 Logical view

The logical view shows the functionality that is provided to the users of the system. First an overview of all the subsystems that make up for the entire system is shown and then each subsystem will be explained into detail.
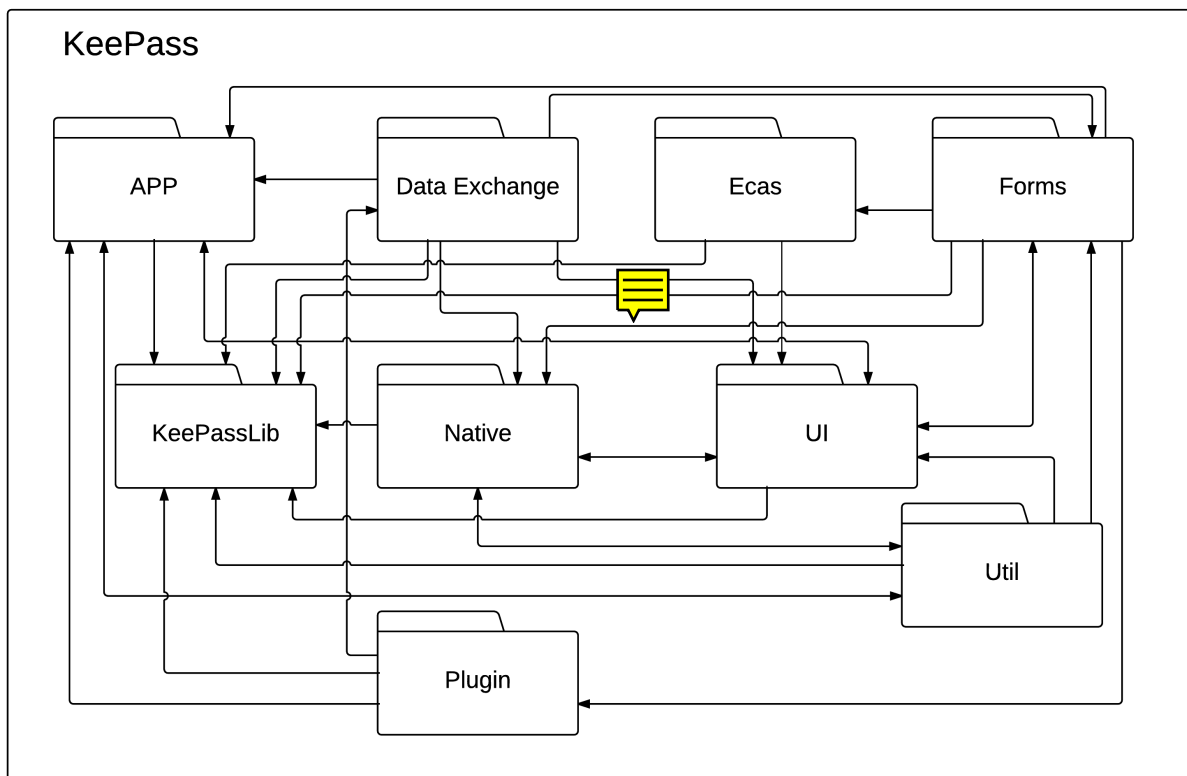


Figure 1: Package diagram of KeePass

### 3.1.1 App

The App package takes care of the configuration of the program. This means that all parameters are initialised in this package. For example, it sets the terminal options, the default language, standard working directory and many more parameters. Also some additional security settings on top of the operating system are created here. For example locking the program when the user minimizes the window or locking the program when the user didn't use the application for a certain amount of time. These functionalities are created to make the program more secure for users who don't pay attention that much.

### 3.1.2    Data Exchange

KeePass has a lot of import and export possibilities. Of course it has the possibility to import and export databases from another device. But it is also possible to import and export a database from the 1.x version of KeePass. Besides exchanging databases, KeePass also offers the functionality to import passwords from other sources. For example other password managers, but also from a browser like Mozilla Firefox.

### 3.1.3    ECAS

The Event-Condition-Action-System allows the user to define actions to be taken after certain events in the program occur. For example, when the program saves the database an external program can be run to synchronize the database file. The ECAS package contains the classes that provide this functionality.

### 3.1.4    Forms

The forms package contains all the forms that are available in the program. For every form, multiple files exist in this package. One file describes the layout and looks of the specific form. Another files gives actions to buttons on the form. This means that when a button is clicked, code in the other file will be executed. The forms also present data that is fetched from for example the database.

### 3.1.5    KeePassLib

KeePassLib is a big package which, as the name refers to, is the internal library for KeePass. It consists of multiple packages with different responsibilities. Below is a more detailed description of these packages.

- **Collections** The collection package holds a number of collections of different types. For example, dictionaries of various types and object lists.

- **Cryptography** This package implements different security algorithms. It implements a random password generator, but it can also test passwords on their quality.

- **Keys** Keys represent different possibilities of providing a key to certain parts. A user can lock the database with a master password, a file, both or with other entities.

- **Native** In order to give a native feel to all users, also the users of the mono version, this package provides some workarounds to give the same functionality to all users.

- **Security** The security package provides some methods to secure binaries and strings that are used in the internal memory.

- **Serialization** Like the name says, this package provides methods to serialize and de-serialize files. This functionality is necessary in order to save and open database files.

- **Translation** This package provides some supporting functions to translate the program into other languages. However, the actual translated strings are not located in this package.

- **Utility** The utility package provides some basic methods that can be used throughout the entire program. It provides for example a basic hex to byte converter, some non-standard string methods and a lot more useful methods.

### 3.1.6 Native

This package provides some functionality to replace or support the native features of the operating system. This is to make the different versions of the system look like the same. These files make use of different methods that import .dll files. Additionally, some encryption methods are optionally implemented by the host system. The Native package can load these functions when they are available and falls back on a custom implementation when they aren't.

### 3.1.7 Plugins

The developers of KeePass are heavily encouraging third party developers to create plugins. These plugins can easily add functionality to the program. The Plugins package contains base and helper classes for plugin developers as well as the loader classes for KeePass itself.

### 3.1.8 UI

The UI package consists of a lot of files that make up the functionality of the user interface of the program. It heavily works together with the forms package. The forms package shows the actual forms of the application and the UI package delivers the right content.

## 3.2 Process view

The process view gives an overview of the system by using a specific interaction. In this chapter we take a look at two of the basic functionality of the KeePass system; creating a new password and finding your passwords to use them for logging in. These cases are described in two parts first an activity diagram to explain the steps and then a sequence diagram to display how the action travels through the different parts of the system.

### 3.2.1 Creating new passwords

Creating a new password is the action the user takes to save a login set to the KeePass system. Hereby the user is asked in the password form to enter the username, name and possibly password the user wants to use. The password can also be auto generated by the KeePass system with several different methods. For this scenario we assume the user is already connected to the database where the user wants to save the password.
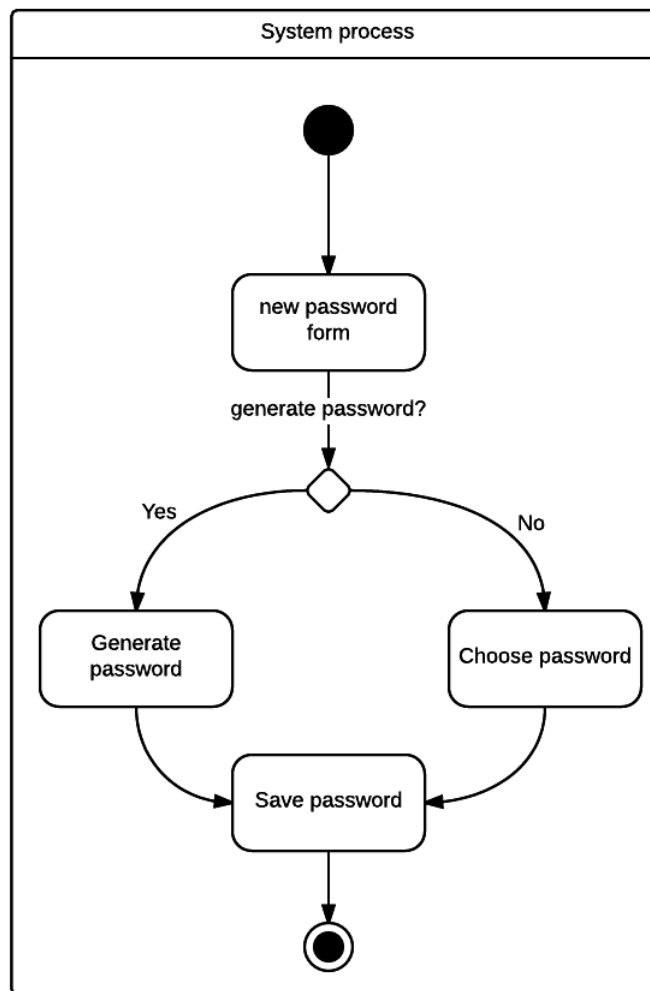
Figure 2: Activity diagram for creating a new password

Figure 2 displays the activity diagram of the create new password scenario. It is a simple process where the user enters a form and the system can generate a safe password or the user can enter a password themselves. When a password is determined, the password entry will be stored in the database.
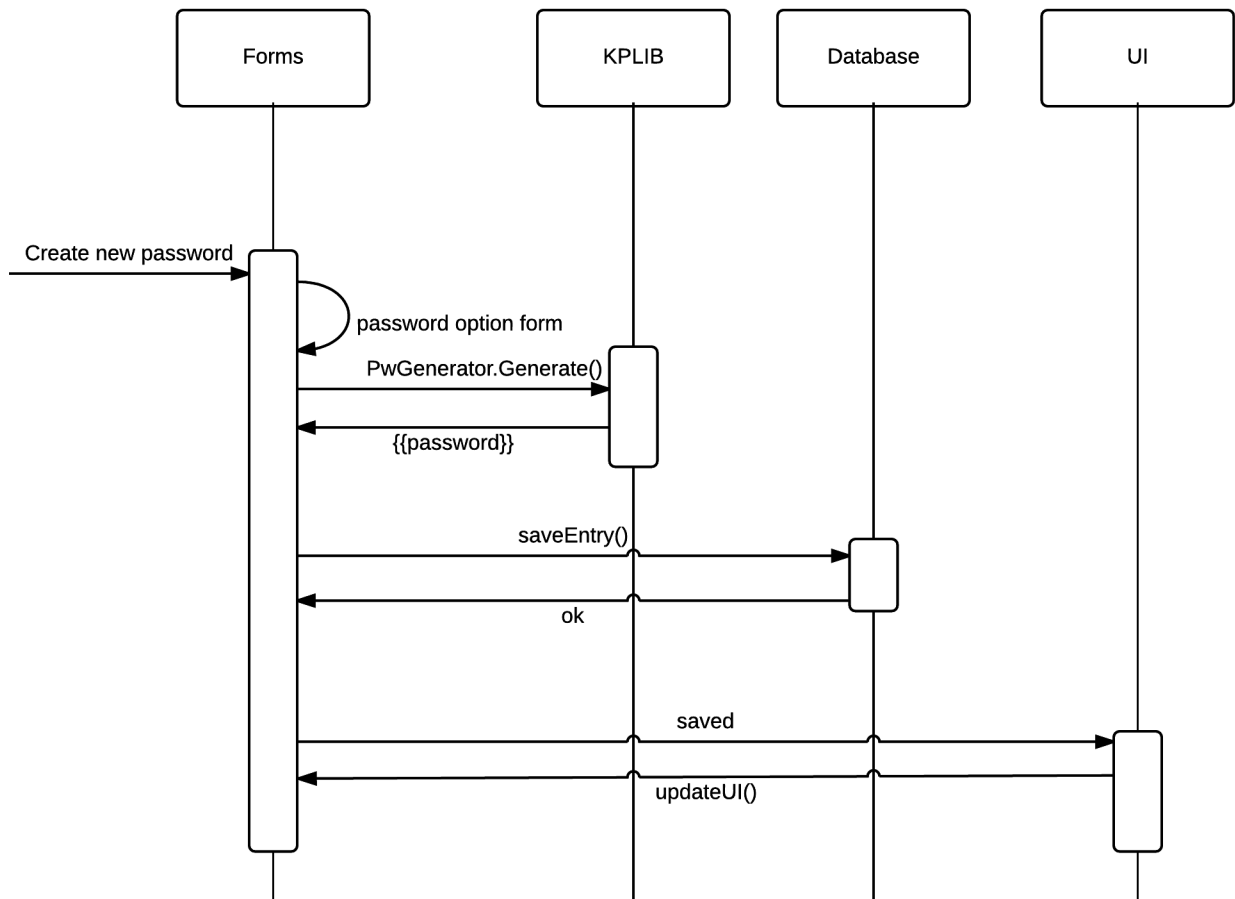
Figure 3: Sequence diagram for creating a new password

In figure 3 the sequence of the create new password scenario is described. This figure shows that when a user chooses to create a new password, a new password option form will be opened. Then a password can be generated by the password generator located in KeePassLib. After this the password entry is saved in the database and the form will be updated by the UI.

### 3.2.2   Finding passwords

Finding a password is the action that the user takes to get a password that is stored in the database. When the user is logged in and in the main view of the program, a list of all the entries is shown. The user can double click on an entry to copy the password to the clipboard. This copy will be removed after a fixed amount of seconds. In this way no one can see the password directly and it is only stored on the clipboard for a short amount of time.
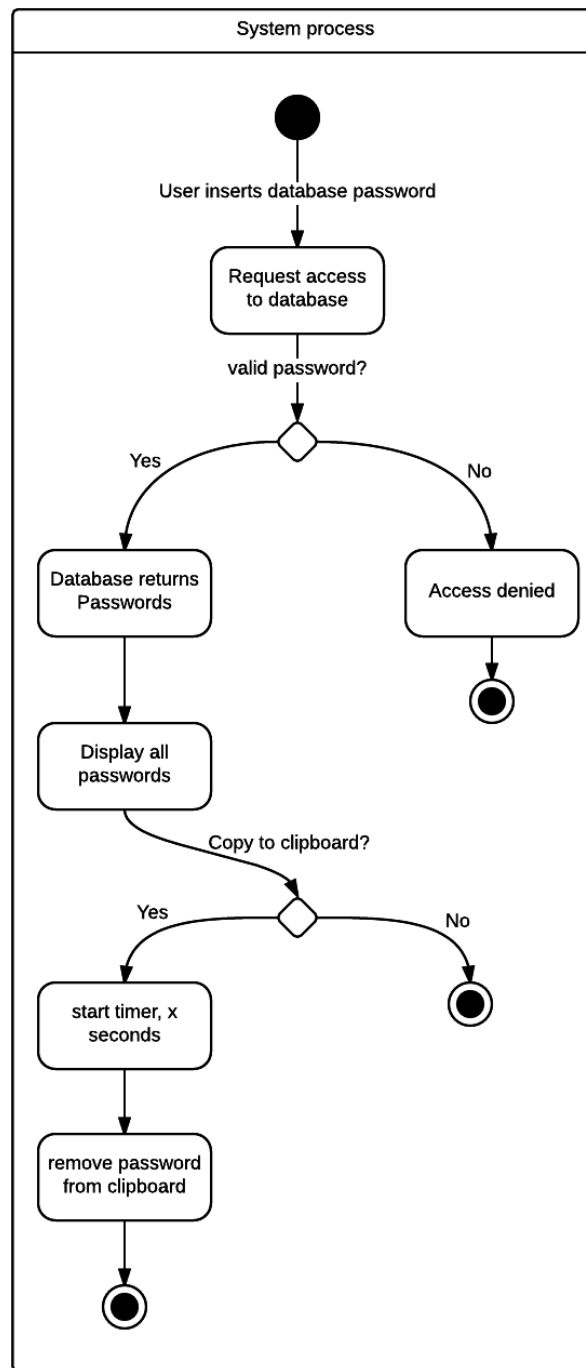
Figure 4: Activity diagram for finding passwords

Figure 4 shows the activity diagram of the find password action. It shows a process where a user must enter the master key to get a list of all the password entries. After that the user can select the password to be
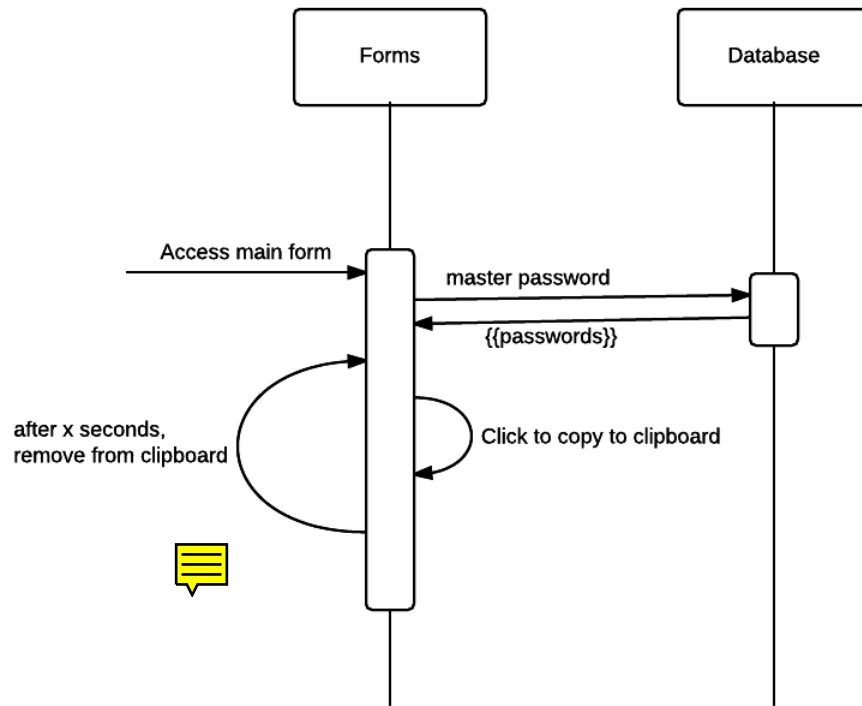
copied to the clipboard.



Figure 5: Sequence diagram for finding passwords

Figure 5 shows the sequence diagram of the find password process. It shows that there is only a connection between the forms and the database packages. The database is needed to retrieve the passwords and the forms package shows the password entries to the user.

# 4 Pattern Documentation

This section the patterns, which have been identified, are outlined. We used the IDAPO[6] method to identify the patterns that were used in the architecture.

For each pattern its traceability is explained, the documentation of the pattern is listed, the reason why it was implemented, how it was implemented and why the pattern was used. Furthermore the implications of each pattern is listed.

## 4.1 Shared Repository

**Traceability**

This pattern can be deduced from the listed features of KeePass[2]. The 'PwDatabase' is the class that serves as the shared repository. It can be found in 'KeePass/KeePassLib/PwDatabase.cs'.

**Source**

This pattern is documented in [4].

**Issue**

Passwords need to be stored in a secure place. Furthermore the passwords need to be used by multiple components.

**Assumptions/constraints**

The password database consists of only one file and is easily transferable to other machines.

**Solution**

The password database is implemented mainly in 'PwDatabase.cs'. It contains the name, description and security attributes of the database. The password entries themselves are contained in a 'PwGroup', which can also contain other 'PwGroup's to allow nested password groups.

**Rationale**

The complexity is significantly decreased by using this pattern. There is one point of access of the data. This also makes it easies to secure the data, because the data is stored in one place.

**Implications**

Data consistency needs to be ensured.

## 4.2 Pipes and Filters

**Traceability**

The pattern has been deduced from the source code of KeePass and the listed features of KeePass[2].

**Source**

This pattern is documented in [4].

**Issue**

One of the main capabilities of KeePass is the ability to export the password list to various different formats like TXT, HTML, XML and CSV as well as the ability import various export format used

by other password safes. For example Password Agent and Password Keeper are two of the over 35 supported formats.

**<mark>Assumptions/constraints</mark>**

**Solution**

The Pipes and Filters pattern is used to structure the import and export password lists. The process of importing and exporting is split into various sub-tasks like the reading of the file, parsing the file and error handling.

**Rationale**

By splitting the process of importing and exporting a password file the complexity of these operations is significantly decreased. Furthermore the Pipes and Filters patterns improves the extensibility of KeePass. For example new formats of password lists can easily be added and existing ones can be modified when needed.

**Implications**

This pattern increases the extensibility of the system as well as decreasing the complexity of KeePass. <mark>However this pattern might cause some overhead.</mark>

## 4.3   Plug-in pattern

**Traceability**

The features of KeePass[2] list the support for plug-ins. The <mark>map</mark> 'KeePass/Plugins' contains all the classes to facilitate the support for plugins.

**Source**

This pattern is documented in [4].

**Issue**

One of the main capabilities of KeePass is the Plugin architecture. This allows other people to extend the functionality of KeePass, like providing additional import and export methods for different formats.

**<mark>Assumptions/constraints</mark>**

**Solution**

The Plugin pattern is used to facilitate this functionality. It introduces <mark>an interface</mark> that is a single, central point so that configuration is easily managed. This is realised by the 'Plugin' class.

**Rationale**

This pattern is relatively easily to implement and provides a transparent way for other people to implement plug-ins.

**Implications**

The extensibility of KeePass is increased as the application can be extended to include new features. Furthermore parallel development is possible, because plug-ins can be developed by multiple teams.

## 4.4 Composite pattern

**Traceability**

This pattern is used in the 'PwGroup' class. This class is used to group several password entries. A password entry can be a single entry or a 'PwGroup'.

**Source**

This pattern in documented in [5].

**Issue**

Passwords entries should be able to consist of entries and/or groups of password entries.

**Assumptions/constraints**

**Solution**

The 'PwGroup' class contains two lists where one of them stores 'PwEntry's and the other stores 'PwGroup's.

**Rationale**

This pattern provides an easy and efficient way to let a 'PwGroup' class both contain 'PwGroup's and 'PwEntry's. Furthermore by using this pattern it is possible to treat individual object and the composition of such objects uniformly.

**Implications**

Usage of this pattern makes the program simpler for the client as they can treat simple structures and composite structures uniformly.


## 4.5 Template method

**Traceability**

**Source**

**Issue**

**Assumptions/constraints**

**Solution**

**Rationale**

**Implications**


## 4.6 Chain of responsibility

**Traceability**

**Source**

**Issue**

**Assumptions/constraints**

**Solution**

**Rationale**

**Implications**

## 4.7   Memento

**Traceability**

There are multiple classes that facilitate serialization of data/objects. For example the 'XmlSerializerEx' class that serializes the user's preferences to an XML file so it can be reused.

**Source**

This pattern is documented in [5].

**Issue**

Object and data need to be stored so that they can be reused. The preferences of user need to be saved so that they can be used every time when KeePass is used.

**Assumptions/constraints**

**Solution**

Serialization is used to save the user's preferences. These are saved in the form of a XML file.

**Rationale**

It a well-known technique to save states of objects. Furthermore it is widely supported and relatively easy to implement.

**Implications**

This pattern could cause some overhead if a lot of serialization and deseralization have to be done.

## 4.8   Observer

**Traceability**

**Source**

**Issue**

**Assumptions/constraints**

**Solution**

**Rationale**

**Implications**

## 4.9   Visitor

**Traceability**

This pattern is deduced from the source code. It is found in the 'PwGroup' class and the delegate definitions in 'KeePassLib/Delegates/Handlers.cs'.

**Source**

This pattern is documented in [5].

**Issue**

The tree structure of the password database makes simple iteration difficult. It is undesirable to have each component define recursive methods to traverse the tree when they need to perform an operation on each password entry.

**Assumptions/constraints**

**Solution**

The 'PwGroup' class contains the 'TraverseTree' method, which takes a traversal-method parameter (pre-order, in-order or post-order) and a delegate to call for each group and entry encountered.

**Rationale**

**Implications**

# 5 Architecture Evaluation

# References

[1] Comparison of KeePass versions. `http://keepass.info/compare.html`.

[2] Features of KeePass. `http://keepass.info/features.html`.

[3] KeePass community. `http://sourceforge.net/p/keepass/discussion/`.

[4] Paris Avgeriou and Uwe Zdun. Architectural Patterns Revisited – A Pattern Language.

[5] Ralph Johnson John Vlissides Erich Gamma, Richard Helm. Design Patterns - Elements of Reusable Object-Oriented Software.

[6] Klaas Jan Stol and Paris Avgeriou. IDAPO: A Process for Identifying Architectural Patterns in Open Source Software.

## Week 1

| Name | Task | Hours |
|------|------|-------|
| Brandsma | Meeting (10-12-15) | 1 Hour |
| | Introduction | 1 Hour |
| | | **2 Hours** |
| Gunnink | Meeting (10-12-2015) | 1 Hour |
| | | **1 Hours** |
| Mauritsz | Meeting (10-12-15) | 1 Hour |
| | Setting up document | 1 Hour |
| | Key drivers | 0.5 Hour |
| | | **2.5 Hours** |
| Tijsma | Meeting (10-12-15) | 1 Hour |
| | Chapter 2 | 1.5 Hours |
| | | **2.5 Hours** |
| **Group Total:** | | 8 Hours |

## Week 2

| Person | Task | Hours |
|--------|------|-------|
| Brandsma | Coaching session | 1 Hour |
| | Reading documentation/website/fora | 2 Hours |
| | Improving chapter 1 | 1 Hour |
| | | **4 Hours** |
| Gunnink | Coaching session | 1 Hour |
| | Studying documentation | 1 Hour |
| | Setting and reading up source code | 2 Hours |
| | | **4 Hours** |
| Mauritsz | Coaching session | 1 Hour |
| | Updating requirements, processing feedback | 2.5 Hours |
| | Reading documentation/website/fora keepass | 2 Hours |
| | | **5.5 Hours** |
| Tijsma | Coaching session | 1 Hour |
| | Reading source code | 3 Hour |
| | | **4 Hours** |
| **Group Total:** | | 17.5 Hours |

# Week 3

| Person | Task | Hours |
|---|---|---|
| Brandsma | Coaching session | 1 Hour |
| | Meeting and work | 5 Hours |
| | Code research + Chapter 3 | 8 Hours |
| | | **14 Hours** |
| Gunnink | Coaching session | 1 Hour |
| | Meeting and finding patterns | 6 Hours |
| | Pattern searching & documenting | 6 Hours |
| | Review | 1 Hour |
| | | **14 Hours** |
| Mauritsz | Coaching session | 1 Hour |
| | Meeting and work | 6 Hours |
| | Chapter 3 software architecture + reading KeePass code | 8 hours |
| | | **15 Hours** |
| Tijsma | Coaching session | 1 Hour |
| | Meeting and finding patterns | 6 Hours |
| | Chapter 4 | 8 Hours |
| | | **15 Hours** |
| **Group Total:** | | 58 Hours |

Table 4: Personal totals

| Name | Hours |
|---|---|
| Jeroen Brandsma | 20 Hours |
| Marco Gunnink | 19 Hours |
| Daan Mauritsz | 23 Hours |
| Ynte Tijsma | 21.5 Hours |
| **Grand total:** | 83.5 Hours |