Projektdokumentation „Steganographie“

Titelblatt MEGASUPERDESIGN!

Inhaltsverzeichnis

[Einführung 2](#_Toc325401958)

[Motivation 2](#_Toc325401959)

[Ziel 2](#_Toc325401960)

[Abgrenzung 2](#_Toc325401961)

[Organisation 2](#_Toc325401962)

[Theoretische Ausgangslage 2](#_Toc325401963)

[Einleitung 2](#_Toc325401964)

[Sicherheit 2](#_Toc325401965)

[Technische Steganographie 3](#_Toc325401966)

[Praktische Umsetzung 3](#_Toc325401967)

[Grundlage 3](#_Toc325401968)

# Einführung

## Motivation

Für dieses Projekt wollten wir uns mit einem Thema aus der Kryptografie auseinandersetzen. Beim durchforschen des Fachgebiets sind wir mit dem Begriff Steganographie in Berührung gekommen. Bei der Steganographie handelt es sich im Wesentlichen um das Verstecken von Informationen in einem Trägermedium. Dieser Gedanke, dass man Informationen versteckt versenden kann, und der Empfänger die verborgene Information wieder herausfiltern kann, war für uns der Anstoss, dass wir uns im Rahmen dieses Projektes mit der Umsetzung einer solchen Lösung auseinandersetzen wollten.

## Ziel

Wir wollen eine Applikation entwickeln, welche Bildern und Audiodateien als Trägermedium verwenden kann um andere Daten darin zu verstecken. Unser Ziel ist es, dass wir unabhängig von den verwendeten Algorithmen, keine Einschränkungen bei den Daten haben, welche wir verstecken wollen. Diese soll uns ermöglichen, beliebige Textdokumente, Bilder, Audiodateien oder sogar EXE-Dateien in den genannten Trägermedien zu verstecken.   
Nebst dem Verstecken von Informationen ist natürlich das Herausfiltern oder Suchen dieser ebenso wichtig und stellt den zweiten wichtigen Bestandteil unserer Applikation dar.  
Mit unserer Applikation soll als Beispiel das folgende Szenario ermöglicht werden. Eine Person (Sender) soll Daten mit unserer Applikation in einer Trägerdatei verstecken könne. Anschliessend soll das veränderte Trägermedium über einen beliebigen Kanal an eine andere Person (Empfänger) übermittelt werden, welche wiederum mit unserem Programm die versteckten Informationen herausfiltern kann, ohne jemals das originale, unveränderte Trägermedium gesehen zu haben.

## Abgrenzung

Um die gesteckten Ziele in der zur Verfügung stehenden Zeit erfüllen zu können, mussten wir einige Abgrenzungen bezüglich der Funktionalitäten und des Umfanges festlegen.

1. In unserer Applikation werden wir Algorithmen implementieren, welche es ermöglichen sollen, Dateien oder Informationen in Bild- oder Audio-Trägerdateien zu verstecken. Es werden keine weiteren Algorithmen, für zum Beispiel Text- oder Video-Trägerdateien, implementiert.
2. Da es bei Bild- und Audiodateien diverse Dateiformate gibt, die sich insbesondere in den Komprimierungsverfahren, als auch in der inneren Dateistruktur unterscheiden, haben wir uns auf die verlustfreien (Komprimierungs-)Verfahren beschränkt.  
   Namentlich unterstützen wir bei den Bilddateien die Formate BMP und PNG und bei den Audiodateien WAV.

# Projektmanagement

## Organisation

### Scrum Master

Jens-Christian Fischer  
Dozent an der ZHAW

### Kunde

Lars Kruse  
Dozent an der ZHAW

### Projektmitglieder

Renato Estermann  
Informatik-Student an der ZHAW, 4. Semester

Niki Hausammann  
Informatik-Student an der ZHAW, 4. Semester

## Iterationsplan

Der Iterationsplan besteht aus vier Iterationen, mit den jeweiligen User-Stories. Die User Stories wurden im Tool *The Bug Genie* erfasst, sowie den verwendeten Aufwand.  
<http://esternet.dyndns.org/thebuggenie-3.2rc8/thebuggenie/swp2stegano>

### Iteration

|  |  |  |
| --- | --- | --- |
| User Story # | User Story Titel | Beschreibung |
| 1 | Einlesen von Trägerdaten (Bild) | Der User kann ein Bild, welches als Trägermedium für die versteckten Informationen/Daten dient, in das System einlesen. |
| 2 | Einlesen von Trägerdaten (Bild) mit versteckten Informationen | Der Benutzer kann ein Bild mit den versteckten Informationen ins System laden. |
| 3 | Textinformationen ins System laden | Der User kann einen Text, der den versteckten Inhalt enthält, ins System laden. |

### Iteration

|  |  |  |
| --- | --- | --- |
| User Story # | User Story Titel | Beschreibung |
| 4 | Bildexport | Der Benutzer kann das modifizierte Bild aus der Applikation exportieren, damit es weiter versendet oder verarbeitet werden kann. |
| 5 | „Steganographisieren“: Informationsdaten in Trägerdaten (Bild) verstecken | Der Benutzer kann die zuvor eingelesenen Informationsdaten in einem Bild, welches als Trägerdatei dient, verstecken. |
| 6 | Gegenüberstellung des Original und des modifizierten Bildes | Die originale Trägerdatei und das neu modifizierte Bild werden dem Benutzer in der Applikation, zum Vergleich, angezeigt. |

### Iteration

|  |  |  |
| --- | --- | --- |
| User Story # | User Story Titel | Beschreibung |
| 7 | Verborgene Daten aus einer Bilddatei wiederherstellen | Der Benutzer kann das modifizierte Bild mit den verborgenen Informationen einlesen (siehe User Story #2) und die verborgenen Daten werden mit einem Algorithmus wieder herausgefiltert. |
| 8 | „Steganographisieren“: Informationsdaten in Trägerdaten (Audio) verstecken | Der Benutzer kann die zuvor eingelesenen Informationsdaten in eine Audiodatei, welche als Trägerdatei dient, verstecken. |

### Iteration

|  |  |  |
| --- | --- | --- |
| User Story # | User Story Titel | Beschreibung |
| 9 | Verborgene Daten aus einer Audiodatei wiederherstellen | Der Benutzer kann die modifizierte Audiodatei mit den verborgenen Informationen einlesen (siehe User Story #2) und die verborgenen Daten werden mit einem Algorithmus wieder herausgefiltert. |
| 10 | Veränderungsgrad der Trägerdatei | Der Benutzer, welcher die Information in der Trägerdatei versteckt, kann den Veränderungsgrad (Stärke der Veränderung od. Verunreinigung), der angewendet werden soll, angeben. |

# Theoretische Grundlagen

Man spricht von Steganographie, wenn eine Information in anderen Daten versteckt wird. Das Wort kommt aus der griechischen Sprache und heisst „verdeckt schreiben“. Bei der Steganographie versucht man also, die Existenz einer Nachricht zu verbergen. Historisch ist diese Methode schon lange bekannt und wurde schon abseits der digitalen Welt erfolgreich eingesetzt. Man kann nachlesen, dass zum Beispiel Nachrichten auf die Kopfhaut von Sklaven tätowiert wurden und der Empfänger diese dann geschoren hat um die Nachricht zu lesen. Aber auch uns allen bekannte Hilfsmittel wie „Geheimtinte“ fallen unter Steganographie.

## Sicherheit

Gemäss dem allgemein bekannten Prinzip von Kerckhoff ist uns bekannt, dass die Sicherheit eines Systems nicht von der Geheimhaltung des Algorithmus abhängen darf, sondern einzig und allein von der Geheimhaltung eines Schlüssels. Wenn wir nun die Steganographie im Allgemeinen nach diesem Prinzip betrachten fällt uns auf, dass nur das Verstecken der Nachricht wohl keine Datensicherheit garantieren kann. Bei der Umsetzung haben wir uns auf ein einfaches Verstecken und Herausfinden beschränkt. Diese Methode ist also nicht sicher.

## Technische Steganographie

Wir beschränken uns bei dieser Umsetzung auf die technische computergestützte Steganographie. Möglichkeiten dazu bieten vor allem Dateiformate, welche nicht von einem Computer weiterverarbeitet werden sondern von einem Menschen interpretiert werden. Dies sind zum Beispiel Bildformate, Audio- und Videodateien. Ein Mensch kann bei solchen Dateien geringe Abweichungen nicht wahrnehmen und schöpft deshalb auch keinen Verdacht, dass da noch mehr sein könnte.

# Praktische Umsetzung

In diesem Kapitel möchten wir aufzeigen, wie wir Informationen in anderen Dateien verstecken und wieder herausholen.

## Grundlage

Wir modifizieren Dateien auf Byte-Ebene. Das heisst, dass wir die Dateien, in welchen wir Daten verstecken wollen, zuerst einmal Byte für Byte einlesen, die Information Bit für Bit „einpflanzen“ und danach die modifizierte Datei wieder auf dem Dateisystem ablegen. Die meisten Audio-, Video- und Bildformate sind schon mehr oder weniger als Bytes organisiert. Wenn wir als Beispiel einen Bildpunkt eines BMP-Files anschauen sehen wir, dass dieser durch 3 Bytes beschrieben wird: je ein Byte für den Blau-, Rot- und Grünkanal. Ein Weisser Bildpunkt präsentiert sich als 1111 1111 1111 1111 1111 1111 in binär oder 255 255 255 in hex. Damit können 16‘777‘216 unterschiedliche Farben repräsentiert werden.   
Man geht heute davon aus, dass das menschliche Auge etwa 400‘000 Farbwerte voneinander unterscheiden kann[[1]](#footnote-1), was offensichtlich viel weniger ist, als wir mit einer BMP-Datei darstellen können. Wenn wir nun hingehen, und bei jedem Byte das „least significant byte“ (LSB) modifizieren, haben wir immer noch 221 = 2‘097‘152 unterschiedliche Farbwerte, was immer noch deutlich mehr ist als der durchschnittliche Betrachter erkennen kann. Wir können also sagen, dass bei einem Verfahren, welches jeweils das LSB jedes Bytes modifiziert, optisch keine Manipulation der Bilddaten auffällt.

Die Grundlage unserer Methode, Daten zu verstecken, ist also Daten, welche von einem Menschen mit seinen Sinnen interpretiert werden so zu verändern, dass dieser die Veränderung nicht wahrnehmen kann.

## Header – Aufbau der versteckten Daten in der Trägerdatei

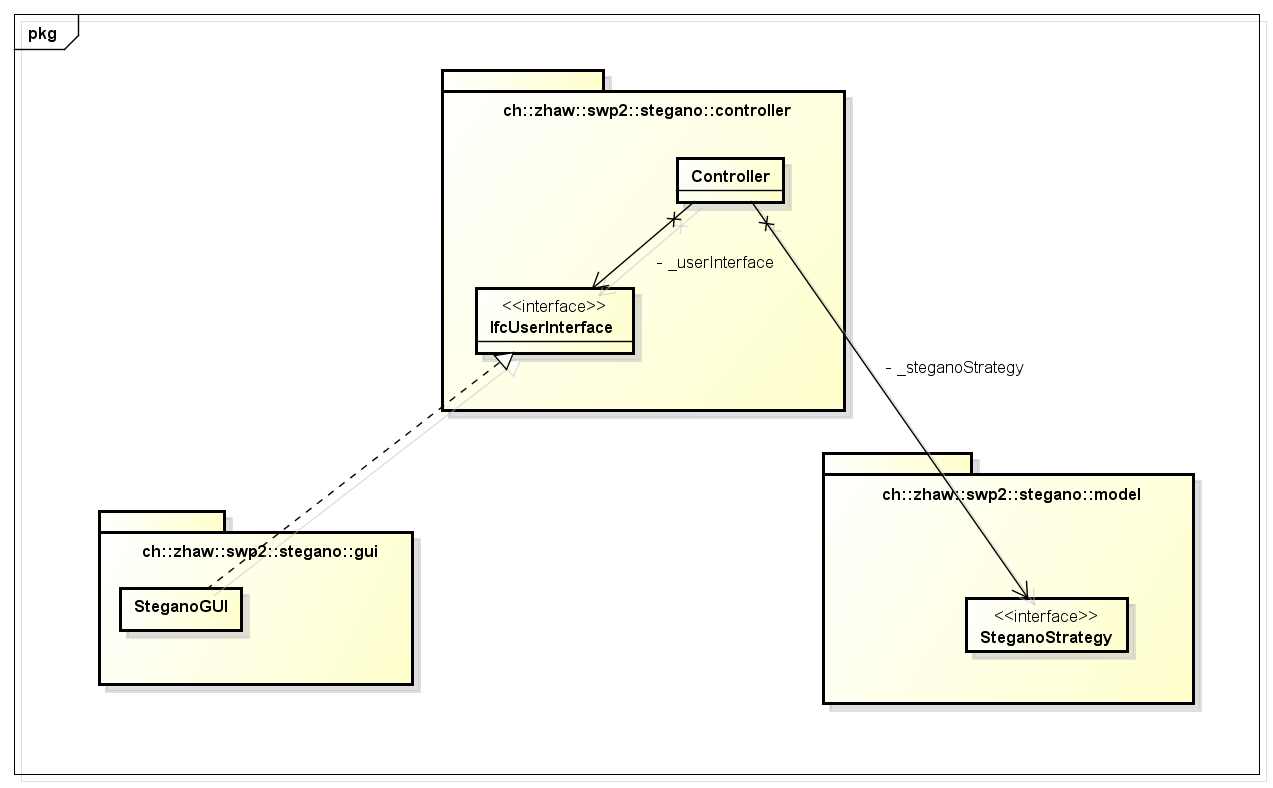
Aufgrund der Anforderung, dass man beliebige Daten in der verwendeten Trägerdatei verstecken können soll, mussten wir zusätzliche Informationen in der Trägerdatei hinterlegen. Damit wird sichergestellt, dass beim Herausfiltern die versteckte Datei auch wieder komplett rekonstruiert werden kann.  
Wir verfolgten für die Lösung dieses Problems den Ansatz eines Protokolls, wie man es aus der Netzwerk- und Nachrichtentechnologie kennt. Unser Protokoll setzt sich aus den folgenden Feldern zusammen:

* *File-Extension (3 Bytes)*: Dieses Feld repräsentiert die Dateierweiterung, wobei jedes der drei Byte einen Buchstaben (US-ASCII codiert) der Erweiterung beinhaltet. Die Bytes werden sequentiell gelesen.
* *Hiddenfile Length (4 Bytes)*: In diesem Feld wird die Länge der versteckten Datei (in Bytes) angegeben. Es handelt sich hierbei um einen signed Integer Wert, der positiv sein muss. Dadurch ergibt sich auch die maximale Anzahl Bytes von 231 Bytes im Feld *Data*.
* *Pollution (1 Byte)*: Dieses Feld gibt an, wie gross die Verunreinigung des ausgelesenen Wertes aus der modifizierten Trägerdatei ist. Die Verunreinigung kann je nach Steganographie-Algorithmus unterschiedlich interpretiert werden. (z.B. Anzahl verunreinigte Bits in einem Byte oder Anzahl verunreinigte Bytes in einem Wort)
* *Data (n Bytes)*: In diesem Feld werden die eigentlichen Nutzdaten, respektive die eigentlichen Bytes, die die zu versteckende Datei repräsentieren, abgefüllt. Die maximale Anzahl an Bytes wurde bereits im Feld *Hiddenfile Lenght* thematisiert.
* *Checksum (CRC32) (8Bytes)*: Abschliessend wird eine Prüfsumme (basierend auf java.util.zip.CRC32) über den kompletten Header und die Daten erzeugt und am Ende angefügt. Dadurch wird sichergestellt, dass nachträgliche (manuelle) Manipulationen in der modifizierten Trägerdatei erkannt werden.

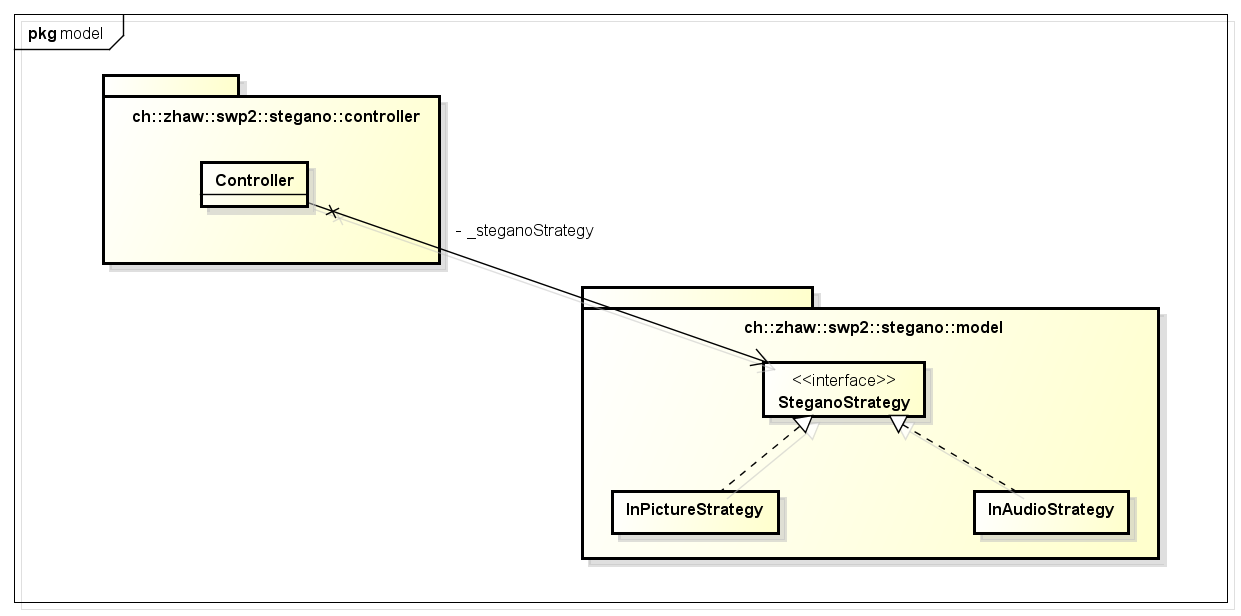
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 32 bit | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 15 | 16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 31 |
| File-Extension (3 Bytes) | | | | | | | | | | | | | | | | | | | | | | | | Hiddenfile Length (4 Bytes) | | | | | | | |
| Hiddenfile Length (4 Bytes) | | | | | | | | | | | | | | | | | | | | | | | | Pollution (1 Byte) | | | | | | | |
| Data (max. 231 Bytes) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Checksum (CRC32) (8 Bytes) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Checksum (CRC32) (8 Bytes) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

## Angewandte Pattern

### Model-View-Controller Pattern (MVC)



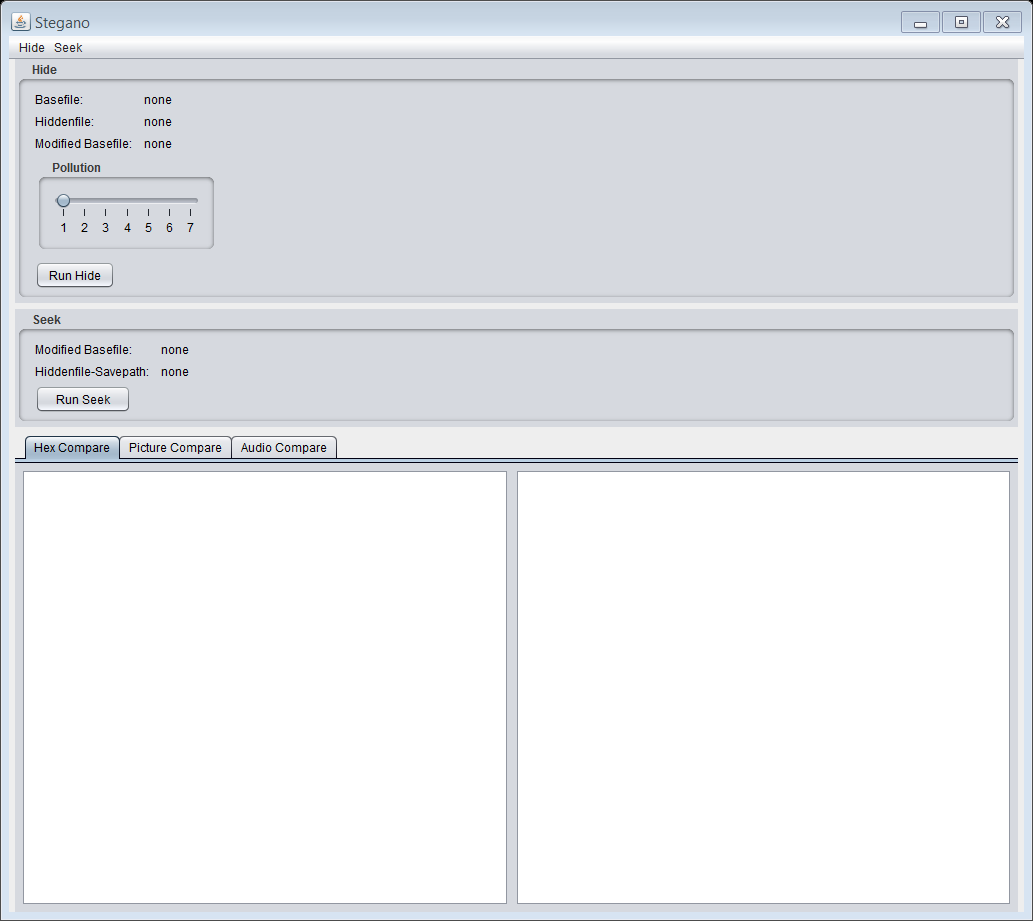
### Strategy-Pattern



## Verwendete Algorithmen?!?

## Benutzeranleitung

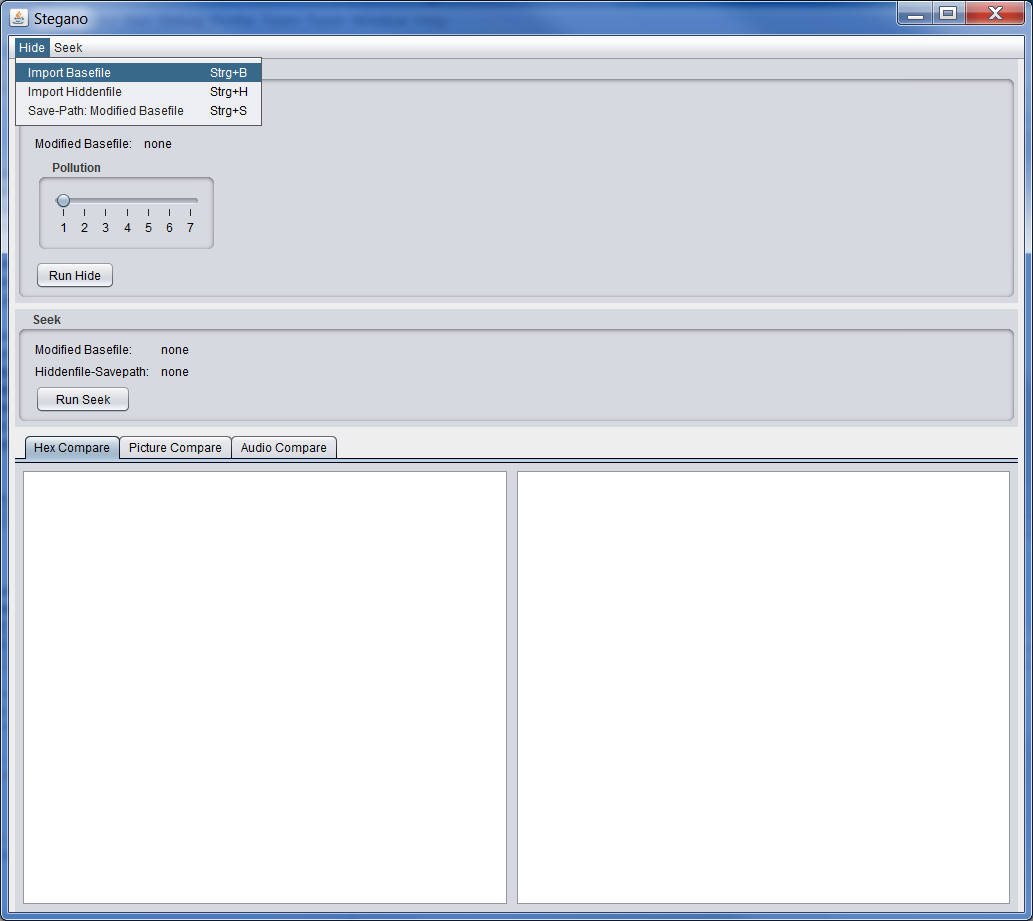
In diesem Kapiteln wird unsere Applikation, und die notwendigen Schritte, erklärt, um eine Datei in einer Bild- oder in einer Audiodatei zu verstecken und die verborgene Datei wieder heraus zu filtern.

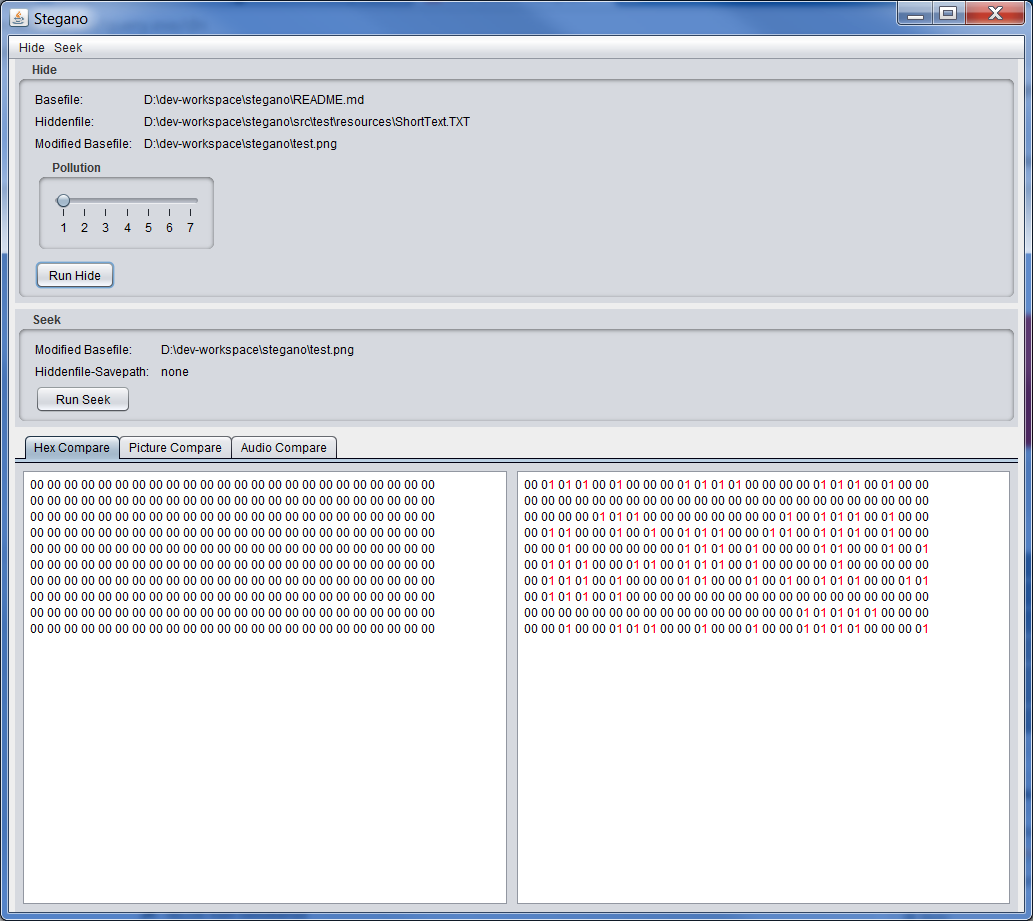


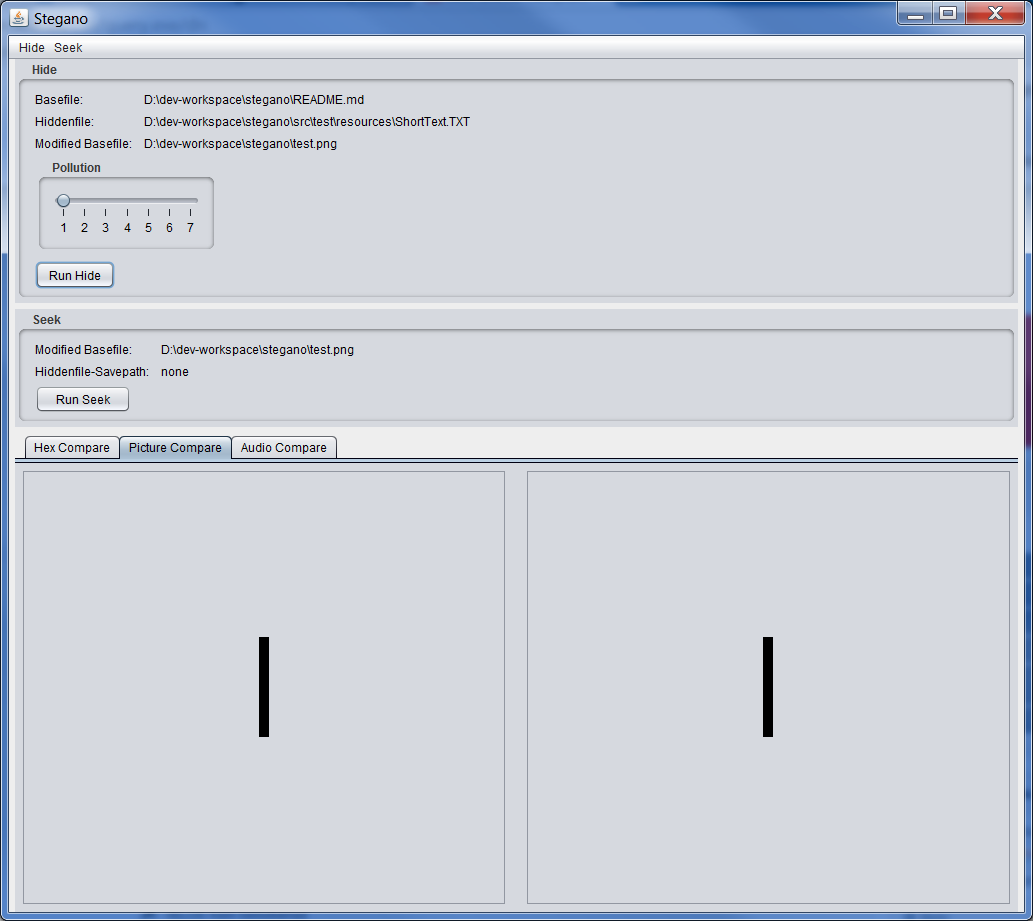
### Verstecken einer Datei

Um eine Datei in einer Trägerdatei zu verstecken muss folgendermassen vorgegangen werden:

1. Über den Menüpunkt *Hide – Import Basefile* wird die Trägerdatei angeben. Erlaubte Formate sind BMP, PNG und WAV
2. Danach muss die Datei angegeben werden, die in der Trägerdatei versteckt werden soll. Dies erfolgt über das Menü *Hide – Import Hiddenfile*. Einschränkungen bei dieser Datei sind, dass die Dateiendung aus drei Zeichen bestehen muss (z.B.: txt, exe, u.s.w.) und in der Trägerdatei genügend Speicherplatz vorhanden sein muss um diese Datei zu verstecken. Die Applikation meldet allerdings, wenn zu wenig Speicherplatz vorhanden sein sollte.
3. Anschliessend muss der Speicherort/Dateipfad, über den Menüpunkt *Save-Path: Modified Basefile*, für die modifizierte Trägerdatei angegeben werden (inklusive Dateinamen und Erweiterung).
4. Nun kann man auf den Button *Run Hide* klicken, und Applikation erstellt die modifizierte Trägerdatei.
5. Anschliessend werden, die Änderungen die durch den entsprechenden Algorithmus bewirkt wurden in einer Hexadecimal Vergleichsansicht (*Hex Compare*) dargestellt. Auf der linken Seite befinden sich jeweils die Originaldaten und Rechts die Veränderungen in rot eingefärbt.
   1. Handelt es sich bei der Trägerdatei um ein Bild, so wird zusätzlich der Bildvergleich (*Picture Compare*) entsprechend aufbereitet.
   2. Handelt es sich bei der Trägerdatei um eine Audiodatei,







### Herausfiltern einer verborgenen Datei

Um eine verbogene Datei aus einer modifizierten Trägerdatei wieder heraus zu filtern muss folgendermassen vorgegangen werden:

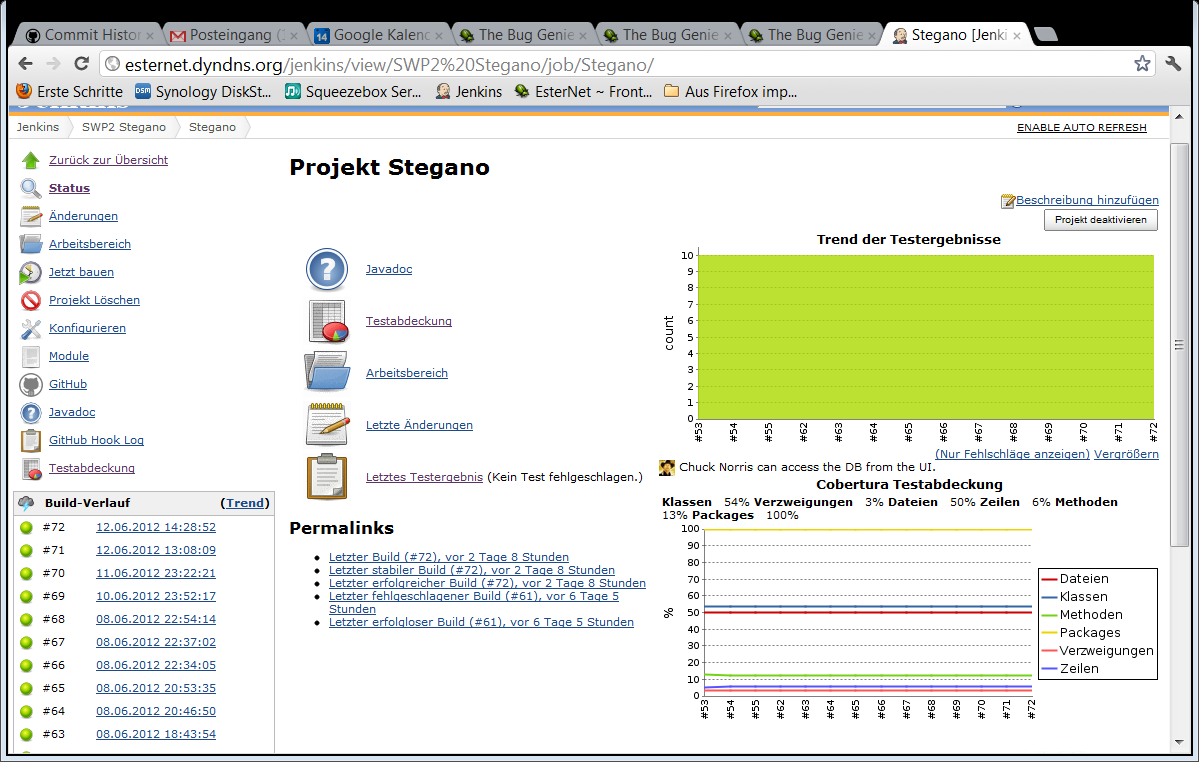
## ! Nicht erreichte Ziele: z.B. Stärke der Verunreinigung (Pollution)

## Feedback

# Anhang

## Verwendete Programme und Hilfsmittel

Die Applikation wurde in Java programmiert und es wurden die folgenden Programme und Hilfsmittel verwendet.

* Es wurden die folgenden zwei Entwicklungsumgebungen verwendet:   
  *Eclipse [*[*http://eclipse.org/*](http://eclipse.org/)*] und NetBeans [*[*http://netbeans.org/*](http://netbeans.org/)*]*
* Für das Erstellen von automatisierten Tests wurde das folgende Framework verwendet:   
  *JUnit [*[*http://www.junit.org/*](http://www.junit.org/)*]*
* Für die Verwaltung des Codes wurde das folgende Version Control System eingesetzt:  
  *Git & Github [*[*https://github.com*](https://github.com)*]*
  + Repository: <https://github.com/esterren/stegano>
* Für die Planung des Projektes wurde das folgende Tool verwendet:
  + *The Bug Genie [*[*http://www.thebuggenie.com/*](http://www.thebuggenie.com/)*]*<http://esternet.dyndns.org/thebuggenie-3.2rc8/thebuggenie/swp2stegano>
* Für die Sicherstellung eines konsistenten Programm-Codes, sowie die automatisierte Ausführung von JUnit-Test und Generierung von JavaDoc wurde das folgende Continuous Integration Tool eingesetzt:   
  *Jenkins CI [*[*http://jenkins-ci.org/*](http://jenkins-ci.org/)*]*
  + <http://esternet.dyndns.org/jenkins/view/SWP2%20Stegano/>  
    
* Für die Manipulationen an den WAV Audiodateien wurden externe Java-Klassen verwendet, die von Herrn A. Greensted bereitgestellt wurden
  + <http://www.labbookpages.co.uk/audio/javaWavFiles.html>

1. <http://de.wikibooks.org/wiki/Einführung_in_die_Fotografie/_Grundlagen_der_Bildgestaltung> [↑](#footnote-ref-1)