Designing and implementing a system for automating the Java project analysis process

Zheng Yan
Abstract

A process for the analysis and collection of information of software systems has been defined. It extracts relevant information of project source files from an online repository and stores that meta-information in a database for further processing. Then, according to the meta-information in the database, it downloads the source files and writes the feedback information back to the database as well. Now the data can be used as input for various analysis tools, in our case a tool called VizzAnalyzer, which reads the project source code and performs a series of software quality analyses.

But actually, the process, which is mentioned above needs, a lot of manual work, makes the work inefficient and the analysis of large numbers of projects impossible.

Thus, a series of thesis projects has been devised to automate the whole process. This thesis aims at automating the information extraction and source file download work, which will make the latter preparation of the analysis task much easier and more efficient.

Key words: Version Control System, CVS, Subversion, SVNKit, checkout, revision, automated-download subsystem
# Table of contents

1 Introduction .......................................................................................................................... 1  
  1.1 Motivation .................................................................................................................. 1  
  1.2 Thesis goal ............................................................................................................... 2  
  1.3 Disposition ............................................................................................................... 3  
  1.4 Acknowledgements ................................................................................................. 3  

2 Theory ................................................................................................................................... 4  
  2.1 Background of Version Control Systems .................................................................. 4  
      2.1.1 CVS ............................................................................................................ 4  
      2.1.2 Subversion ............................................................................................... 5  
  2.2 SourceForge.net ......................................................................................................... 8  
      2.2.1 What is SourceForge.net? ........................................................................ 8  
      2.2.2 How SourceForge.net arrange projects? .............................................. 8  
      2.2.3 SourceForge.net with Version Control System ....................................... 10  
      2.2.4 Project Repository Structure in the SourceForge.net ......................... 11  

3 Automated-download system ............................................................................................. 12  
  3.1 Architecture of the java project extraction system ...................................................... 12  
  3.2 Process of the java project extraction system .............................................................. 13  
  3.3 Architecture of the automated-download subsystem .................................................. 13  
  3.4 Automated-download subsystem analysis .................................................................... 14  
      3.4.1 Abstract Repository API .......................................................................... 15  
      3.4.2 Abstract Storage API ............................................................................... 17  
      3.4.3 Implement of the Abstract Repos API ...................................................... 18  
      3.4.4 Implement of the Abstract Storage API .................................................... 20  
  3.5 SVNKit .......................................................................................................................... 21  
      3.5.1 Why SVNKit? .......................................................................................... 21  
      3.5.2 Some Features of SVNKit ...................................................................... 22  
      3.5.3 Purpose of SVNKit ................................................................................ 22  
      3.5.4 Architecture of SVNKit ......................................................................... 23  
  3.6 Test of the subsystem ................................................................................................... 23  
  3.7 System Performance and Evaluation ......................................................................... 27  
      3.7.1 System Performance .............................................................................. 27  
      3.7.2 System Evaluation .................................................................................. 28  

4 Conclusion and Future work ............................................................................................... 30  
  4.1 Did we reach our goal? ............................................................................................. 30  
  4.2 Future work ............................................................................................................. 31  

5 Reference ............................................................................................................................. 32  

Appendix A - SourceForge Connection Information ......................................................... 33
1 Introduction
In this chapter we will give an introduction to this thesis. First we explain the motivation of writing this thesis. Further on we will discuss the goals and criteria of our thesis work. Also the disposition of the report and the acknowledgements to the people are included in this chapter.

1.1 Motivation
We are currently conducting a study collecting and analyzing metric information from a huge amount of open source projects written in Java. The source code of these projects is downloaded together with meta-information about the projects. Then the projects are prepared for analysis, and analyzed (using a tool called VizzAnalyzer [1]). Then the gathered meta-data, metrics information, and statistical data are stored in a database for further process using, e.g. MS Excel. Figure1.1 shows the scenario of the above project analysis procedure. But in fact, this whole process of the java project analysis involves a lot of manual work (e.g. we should go to the project website and download those projects one by one), which slows it down.

For automating the above process as much as possible we devised some approaches to improve the efficiency. One thesis is to parse the source forge homepage and store the meta-information about the project together with their download location (the project repository URL) in a database, another project deals with the manual preparation of the projects for analysis and their analysis. To even further automate the current process we would like to automatically download the projects listed in a database to the local project workspace for further processing.

![Figure 1.1: The Scenario of the Project Analysis](image-url)
1.2 Thesis goal
The main task of this thesis is to implement a tool, which automates the download task of the projects. It should reach the goals as showed in the following:

- **It shall first access a database for reading the repository locations (the information which is stored before)**
  
  Criteria: We need to get connection to the database to read those project information in which includes the project name, the project UNIX name and the project repository URL, etc.

- **It should automatically download these projects into the local disk.**
  
  Criteria: If download succeeded, the source files of the projects will be stored in our local disk and cataloged into different folders according to the project name. If the download process failed due to some reasons like problems with the website or the network connection troubles, it will display the error message.

Additionally various configuration options should be available:

- **Download a selection or all of the projects stored in the database**
  
  Criteria: We can choose whether to download all the content of the project source files from the specified remote repository or just a part of them. (Usually most of the users have much more interest in the so-called ‘trunk’ part of the repository, which contains most of the java source files.)

- **Awareness if a project has already been downloaded**
  
  Criteria: When a certain project is actually and successfully downloaded into the local disk, some information should be written back to the database which includes a tag value telling whether the download operation is successful or not. So when the program is run again as the specified project is already stored in the disk, it should tell the existence of the project to the user.

- **Download the latest version of the project from repository**
  
  Criteria: If the user does not specify the vision of the projects to be downloaded, the program should download the latest version of them by default.

- **Download multiple earlier versions of the project from repository**
  
  Criteria: Users are able to choose the version (ranges from the first version to the latest version) of the projects to be downloaded.

- **Write some feedback information back to the database when the download process finished.**
Criteria: Some feedback information is needed for the integrality of the whole system. This information includes the author, check-in date, reversion and the directory path where the user stored the project source files.

- **Update the already downloaded project into a certain version**

Criteria: As is mentioned above, if some projects are already downloaded before, the user is able to update the project source files into a certain version.

In our case, the program should use the SVNKit API for accessing the SVN repository. Also the program should allow replacing the SVN API with any other API for CVS repository or some other kind of project repository. Thus flexibility and reusability are important design constraints.

### 1.3 Disposition

The remainder of the thesis is structured as following. In Chapter 2, we have a brief introduction about version control systems (especially used in software engineering and development) and list some major version control tools (CVS and Subversion). Chapter 3 starts by expatiating the architecture of our java project extraction system; also the automated-download subsystem and the system test result are included. Finally, in Chapter 4, we discuss the thesis conclusion and some future work.

### 1.4 Acknowledgements

I would like to thank all the people who helped me finish this thesis with their enthusiastic support. I would like to thank my supervisor Rüdiger Lincke, for his support and comments to my thesis work. Also I would like to thank my professor Mathias Hedenborg who instructed me the methodology of writing a better bachelor thesis. Finally, I want to express my thanks to my family for their support and interest during all this time.
2 Theory
This thesis work is basically related to the automated-download task of the projects from project repositories, which make use of some version control system to manage large amounts of projects. Therefore, this chapter gives a brief introduction to the version control system and I will give a view of one of the most famous project repositories, SourceForge.Net (www.sourceforge.net).

2.1 Background of Version Control Systems
Revision control (some one calls it version control system, source control or code management) is a tool to manage a large amount of different revisions of the information, which is usually shared by a group of people [2].

As a matter of fact, a revision control system is usually used in software engineering aiming to manage the development of documents such as the source code of projects and some information shared by a group of people. Usually people give each change on the source code a certain number or some code, which are used as the “number of revision”. A common approach in the revision control like this: first people set the revision number to be “1”, and when some changes is made on the source code, people set the number to be “2”. In a word, when each change is made by someone on the source code of the project, the revision number will be increased by “1”.

A feature of the version control system (especially used in the field of software engineering) is that the system is able to trace back to any version of the project which means people can have access to every state of the project from the first version to the latest version. This feature is very important to those developers who need to develop the software projects iteratively.

As with the source code of projects, sometimes people also use the version control system to deal with other information as well like the configuration information or some technical documents.

Nowadays, more and more people have supported such a conclusion that the version control system not only makes it easier for a group of people to work together, but also helps the developers to form some good working practices.

For instance, when people begin to use the version control system, they found it better to have a review of their modifications on the project before doing the check in operation since the version control system is able to make the review task much easier. As a result, when finishing the review work, no doubt the source code is much more reliable and readable than people do not do anything on their work. Maybe other people in the same group could pay more attention to the new ideas for the projects instead of spending that unnecessary time in the debugging work.

As is mentioned above, we can see the importance of the version control system in developing software products especially when people work together with each other in a group. The version control system not only increases the efficiency but also makes some improvements to the quality of software projects. All of these reasons lead to the popularity of using such kind of tools.

2.1.1 CVS
What is CVS?
In the field of software development, the Concurrent Versions System (CVS), also known as the Concurrent Versioning System, provides a version control system based on open-source code [3].

Version control system software keeps track of all work and changes in a set of files,
and allows several developers (potentially widely separated in space and/or time) to collaborate.

**Some features of CVS**

CVS makes use of the client-server architecture which works like this pattern: A server stores the current version of a project and all of the history revisions, and many clients connect to the server in order to download a complete copy of the project (this kind of operation in the field of version control system is called **check out**), work on this copy later and then submit their changes to the server (this kind of operation is called **check in**).

Usually, the client and server are connected through the LAN or over the WAN. Another case is that the clients and server may both run on the same computer if CVS needs to keep track of the version history of a project with local developers.

CVS features that it makes it possible that several developers could work on the same project concurrently, everyone doing modifications on the project files within their own "working copy", and then submitting(also called check in) their modifications to the server. In order to avoid the possibility of people stepping on each other's toes, there exists a mechanism that the server would only accept those changes made to the latest version of a project file. All the developers are expected to keep their working copy up-to-date by importing other people's changes on a regular basis. The CVS client mostly accomplishes the task automatically. It requires manual intervention only when a conflict appears between a checked-in modification and the unchecked local version of a project file.

**Some CVS Terminology**

CVS marks a set of project files as a **module**. A CVS server stores the modules it manages in its project repository.

As is mentioned above, a **check out** operation refers to downloading a complete copy of the projects from the project repository. Similarly, submitting the local project files to the repository server is called the **check in** operation.

Developers acquire copies of modules by checking them out from the repository. The checked-out files in the local disk are called the **working copy**. Any modifications made on the working copy will be reflected in the repository by submitting them. The **update** operation is to merge the modifications in the repository with the local working copy.

**2.1.2 Subversion**

**What is Subversion?**

Another famous version control system, which is currently gaining in popularity, is Subversion.

The goal of the Subversion (SVN) is to build a version control system that is a perfect replacement for CVS in the open source community.

Subversion is well-known in the open source community and is used on many open source projects such as: Apache Software Foundation, KDE, GNOME, Free Pascal, GCC, Python, Ruby, Samba and Mono. SourceForge.net and Tigris.org also provide Subversion hosting for their open source projects. Google Code and BountySource systems use it exclusively [4].
Features of Subversion
Since the subversion version control system has gained the popularity among many big IT companies, we should have a look at those features it holds.

- **Subversion shares most of the features of CVS**

Subversion can be viewed as a better version of CVS, so it shares most of CVS's features.

- **Support Directories Rename**

One of the most common drawbacks about CVS is that CVS does not support the directory rename. To get over this problem, the Subversion version control system makes the rename operation possible to the users.

- **Offer the Atomic Submitting Mechanism**

The submitting of a local working version will take effect only when the entire submitting is successfully done. Else the server will reject the check in operation. Generally, the revision numbers are per-commit while the log messages are attached to the revision, not stored redundantly as in CVS [5].

- **Branching and tagging are cheap operations**

Usually developers prefer to make these operations (branching and tagging) easier to be carried out. So the subversion version control system just takes them as an underlying kind of copy operation, which occupies rather small space.

- **Client/server Architecture**

Subversion also uses the client/server design architecture, which can effectively avoid some maintenance problems. Unlike CVS, there is no such a concept like module in subversion, all the source files are arranged with some well-defined interfaces which can be called by other applications.

- **Proportional costs to change size**

Unlike other version control system, the time spent on the subversion operation is proportional to the size of the changes resulting from that operation instead of the whole size of the project in which the changes are taking place.

- **Offers different types of repository**

Repositories can be created with either an embedded database back-end (BerkeleyDB) or with normal flat-file back-end (FSFS).

- **Compatibility of binary files**

Subversion is able to handle binary files as the common text files with its special binary algorithm [5].
As showed in the Table2.1, we can have a clear view of the feature comparison of the SVN and CVS.

<table>
<thead>
<tr>
<th>Features</th>
<th>SVN</th>
<th>CVS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directories Rename</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atomic Submitting Mechanism</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cheap branching and tagging operation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture to use</td>
<td>Client/server</td>
<td>Client/server</td>
</tr>
<tr>
<td>Proportional costs to change size</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table2.1 SVN and CVS features comparison*

Structure of File System in Subversion

As showed in the Figure2.1, the Subversion file system is described as a "three dimensional" file system. While most representations of the directory tree use two dimensional structure, in Subversion the added dimension is to represent the revisions.

The Subversion file system uses transactions to keep changes. A transaction begins from a specified revision of the file system, not necessarily the latest one. The transaction makes changes on its own root. It is then either committed and becomes the latest revision, or is aborted [4].
Drawbacks to Subversion
Although we can see many benefits of the Subversion version control system over the traditional CVS tool, also some problems will happen when we choose to use SVN. As we said before, the Subversion developing team decided to simplify and speed up the branching and tagging operation. Actually, for the reason that a branch can be viewed as another folder within the repository, it is appropriate to make it as simple as possible. But for the tags, this is not a good idea since Subversion does not restrict editing tags. Actually, in SVN the concept ‘tag’ does not work like it does in CVS. So there is no need for us to make this operation simple and fast.

2.2 SourceForge.net
In this section we will give a view of one of the most famous project repositories, SourceForge.Net. Also we will have a look at how this website relates different version control system with management of its open source projects.

2.2.1 What is SourceForge.net?
SourceForge.net is a source code repository which acts as a medium for the software developers to control and manage the open source software products.

One of the most famous features of SourceForge.net website is that it owns a large amount of open source projects. (Nearly 169,281 projects are stored in the repository of the www.sourceforge.net website)[6].

Figure 2.2 shows the homepage of the SourceForge.net website.

![SourceForge.net](image)

SourceForge.net is the world’s largest Open Source software development web site. SourceForge.net provides free hosting to Open Source software development projects with a centralized resource for managing projects, issues, communications, and code.

Figure 2.2 Homepage of SourceForge.net

2.2.2 How SourceForge.net arrange projects?
SourceForge.net provides plenteous of space for the source code versions managed with the CVS or the Subversion version control system which are mentioned above. In the Figure2.3, we can see that several projects information is listed to the users.
In the SourceForge.net website, we can see a short description of each project as showed in the Figure2.4. By reading this information, users will know what this software product is used for immediately.

**Figure2.4 Short description of Project**

As showed in the Figure2.5, we can see some details information of a project. Those information includes Project Unix name, User Interface, Registered date, Operating System, etc. By viewing those information, users could understand the software project much more deeply.
2.2.3 SourceForge.net with Version Control System

Almost all the projects in the SourceForge.net website can be managed by the CVS or Subversion version control system. In the Figure 2.6, we can find that there are three items for each repository menu which are the repository type (CVS/SVN), the Repository Browse and the Repository Statistics (see Figure 2.7).

**G-java**

Welcome to G-java! G-java is an easy-to-use game software development kit based on Java. Non-programmers and programmers alike can use it. It can be used on Windows, Mac OS and Linux.

Also we can get the project repository URL which is useful to get access to the project source files and other configuration files. Furthermore, the SourceForge.net website offers the history repository Activity Statistics which displays a graph showing the frequency of different operations (read and write) on the repository.
2.2.4 Project Repository Structure in the SourceForge.net

In the SourceForge.net website, anyone who is interested in a certain project can scan its source files repository. When you input the repository URL and get into the repository webpage (see Figure 2.8), everything is just as simple as you are faced with your local work space. So it is easy for us to find any project files as quickly as possible.

![Subversion Activity For G-java](image)

*Figure 2.7 Activity Statistics of Project Repository*

<table>
<thead>
<tr>
<th>File</th>
<th>Rev.</th>
<th>Age</th>
<th>Author</th>
<th>Last log entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>1188</td>
<td>9 minutes</td>
<td>kiasubal</td>
<td>Voted and WapTo actions now use the property editor. Small changes to the base.</td>
</tr>
<tr>
<td>Ebean</td>
<td>882</td>
<td>2 months</td>
<td>kiasubal</td>
<td>Great improvements to lookiyiof and plugin system. Started project Ebean - bring.</td>
</tr>
<tr>
<td>G-C++-al</td>
<td>1048</td>
<td>4 weeks</td>
<td>slejja</td>
<td>Added some feature events</td>
</tr>
<tr>
<td>G-C-Sharp</td>
<td>1183</td>
<td>25 minutes</td>
<td>kiasubal</td>
<td>Updated libGCS: integer, hyped and vaped support</td>
</tr>
<tr>
<td>G-Create Installer</td>
<td>697</td>
<td>5 months</td>
<td>tping</td>
<td>Fixed svn</td>
</tr>
<tr>
<td>G-Create Install Script</td>
<td>823</td>
<td>4 months</td>
<td>kiasubal</td>
<td>Improved installer</td>
</tr>
<tr>
<td>G-Java</td>
<td>1180</td>
<td>27 hours</td>
<td>kiasubal</td>
<td>Changes to the plugin system.</td>
</tr>
<tr>
<td>G-JavaLibrary</td>
<td>698</td>
<td>5 months</td>
<td>tping</td>
<td>Added Java 8b</td>
</tr>
<tr>
<td>G-Plus Extension Compiler</td>
<td>697</td>
<td>5 months</td>
<td>tping</td>
<td>Fixed svn</td>
</tr>
<tr>
<td>GM Importer</td>
<td>1182</td>
<td>26 hours</td>
<td>kiasubal</td>
<td>Fixed GM Importer. Added update system to G-C-Sharp.</td>
</tr>
<tr>
<td>build.xml</td>
<td>1</td>
<td>13 months</td>
<td>porfaio</td>
<td>Importing</td>
</tr>
</tbody>
</table>

*Figure 2.8 Structure of Project Repository*

Through what is mentioned above, we have an intuitionistic impression of the structure of SourceForge.net. Also we learned how this project repository deal with large amounts of projects and manage them by those version control system listed in section 2.1.1 and section 2.1.2. Everything we will do in the latter part of the thesis is closely related to the SourceForge.net.
3 Automated-download system

In this section, I will introduce the general architecture of the project extraction system; also we will have a look on the main part of the whole system which is called the automated-download subsystem.

3.1 Architecture of the java project extraction system

In Figure 3.1, we can see the basic structure of such the java project extraction system which consists of the remote projects repository (SourceForge.net), website information extraction subsystem, local database, Java projects automated-download subsystem, and the local disk to store the downloaded java projects.

Through the Figure 3.1 we can have a rough view of how the whole java project extraction system works. First, we get the access to a project source file website (in our
case, it is SourceForge.net). Then we use our web site information extraction subsystem to get the java project information (like the project name, project UNIX name, repository URL, author, etc) and store this information into a local database. When all the work mentioned above is ready, it comes to the main part of the whole system, which however, is the main goal of this thesis, the java projects automated-download subsystem. To make the download process as automatic as possible, we use the project information (especially the repository URL), which is stored before in the database as the input data to the automated-download subsystem, and then the subsystem analyzes the project information and gets access to the specified remote repository to download the java projects. When the subsystem finishes downloading the source files, it also writes some information (like download tag, download revision, last author, storage path, etc) back to the local database.

We see it clearly that the whole structure of the java project extraction system makes it possible to automate the project information extraction and download work, also makes it efficient to do the preparation work for the project analysis task.

3.2 Process of the java project extraction system

1) Get access to the specified remote website
We get access to a specified remote website which consists of large amounts of project information.

2) Extract and store the project information into the local database
After connecting to the specified remote website, we extract some useful information of the projects. Usually, we get the html files of the project web page and analyze these html files, extract some key information or data, which is to be used later. Then we store these extracted information into our local database.

3) Download the projects source files automatically
When the project information extraction work is completed and the information is stored into the database, we continue to download the project source files. Users can get the information of the java projects; also the repository information of the projects (like the content information or the history changes information of each revision in the repository, etc) can be viewed. Then the user should set some download parameters such as the path directory to save the source files and the revision of which he/she wants to download. After all these works is finished, the download work begins and after several seconds (usually it takes 5~20 seconds to download a project) the download operation is over.

4) Write information back to the local data base
Since the local database keeps not only the project information but also some repository operation information related to the java projects, when the download work is over, our automated-download subsystem writes some feedback information back to the database.

3.3 Architecture of the automated-download subsystem
As described in the beginning of this chapter, the main task of this thesis is to develop a method to download (check out) the project sources files automatically and implement it as a program, the following part shows some details about the automated-download subsystem.
Basically, we read the information which is stored in the database in advance and according to this information, we connect to the specified remote repository to download the source files.

Through Figure3.2 we can have a rough view of the architecture of this subsystem. Altogether two main abstract API are designed to cooperate with the automated-download subsystem. One is the Abstract Repository API, which is used to deal with the entire affair related to the remote repository operation (for each certain kind of repository, we have a special API to implement the abstract interface). The other is the Abstract Storage API, which is designed to manage the project information storage affairs. In the other hand, we offer different special API to meet the needs that users may choose different storage measures like local Database, XML files or something else.

### 3.4 Automated-download subsystem analysis

To implement the automated-download subsystem, I used the Java programming language in the practical work.

For the intention that the automated-download subsystem can be integrated into the Eclipse IDE in future and be flexible to different kind of application (e.g. the SVN repository or the CVS repository), also based on the consideration that the user may
choose different types of storage measures, we have defined two main kinds of API which are mentioned above to be implemented and can be adapted to several applications.

Through the Figure3.3, we can have an overview of the big picture about the whole system and figure out the relationship between the interfaces and their implements.

**Figure3.3 UML Structure Diagram of the Automated-download Subsystem**

### 3.4.1 Abstract Repository API

One API offered to the user is called the Abstract Repository API, which is responsible for all the operation related to the remote repository. There are four main interfaces, which are to be implemented by different kinds of specified repository API.

**Common Package:** This package includes all abstract interface related to the repository operation which can be implemented by different kinds of specified repository API (in our case we implement them by the SVNKit which is an API for the subversion repository management)
LocalCheckout.java: This interface is used to deal with the download operation of the remote project repository. The function **doCheckout()** is responsible for accomplishing the checkout task.

![Figure3.6 LocalCheckout Interface](image)

LocalUpdate.java: This interface is used to deal with the update operation of the remote project repository. The function **doUpdate()** is responsible for accomplishing the update task.

![Figure3.7 LocalUpdate Interface](image)

ShowReposContent.java: This interface is designed to display information of all the contents in the remote project repository. The function **showContent()** is responsible for showing the repository content information.

![Figure3.8 ShowReposContent Interface](image)

ShowReposHistory.java: This interface is designed to display information of all the history changes (for each revision) in the remote project repository. The function **showHistory()** is responsible for showing the repository history information.
ReposContentInfo.java: This class is defined to store all the information of the content in the remote repository, which can be used later.

```
import declarations

public class ReposContentInfo {
    String repository;
    String UID;
    ArrayList<RepositoryElement> elements;
    long latestRevision;

    public ReposContentInfo() {
        // constructor
    }

    public String getRepository() {
        // get repository
    }

    public void setRepository(String value) {
        // set repository
    }

    public String getUID() {
        // get UID
    }

    public void setUID(String value) {
        // set UID
    }

    public ArrayList<RepositoryElement> getElements() {
        // get elements
    }

    public void addElement(RepositoryElement element) {
        // add element
    }

    public int size() {
        // get size
    }

    public RepositoryElement getElementAt(int index) {
        // get element at
    }

    public void setLatestRevision(long value) {
        // set latest revision
    }

    public long getLatestRevision() {
        // get latest revision
    }
}
```

ReposHistoryInfo.java: This class is defined to store all the information of the history changes (for each revision) in the remote repository, which can be used later.

```
import declarations

public class ReposHistoryInfo {
    ArrayList<HistoryElement> elements;

    public ReposHistoryInfo() {
        // constructor
    }

    public void addElement(HistoryElement element) {
        // add element
    }

    public void setAllElements(ArrayList<HistoryElement> elements) {
        // set all elements
    }

    public HistoryElement getElementAt(int index) {
        // get element at
    }

    public int size() {
        // get size
    }
}
```

3.4.2 Abstract Storage API

Another API offered to the user is called the Abstract Storage API, which is responsible for all the project information storage affairs. There exists a main interface which is to be implemented by different kinds of specified storage API like Access Database, XML files or something others. (In our case we implement the interface by the Access Database, ODBC and JDBC)

RepositoryStorage.java: This interface is used to deal with all the project information storage work. The function `writeCheckoutInfo()` is used after the download operation is over (to store the check out information) and the function `writeUpdateInfo()` is used after the update task is finished (to store the update information).
3.4.3 Implement of the Abstract Repos API

Since these abstract interfaces mentioned above just define the frame of different operations and we need to implement or extend all these interfaces by using our own API then the program can be applied to some specified application environment.

In our case, we use an API called SVNKit [7], which is developed to operate on the subversion repository in order to implement the Abstract Repository API. For the project information storage, we choose to use the Microsoft Access Database [8] as the medium to store the information. (We use JDBC connecting to ODBC, and then finally have the access to the Access Database)

**SVN package:** This package includes all the SVN-specified classes which are related to the SVN repository operation. All the classes in this package implements the Abstract Repository API mentioned above.

**SVNLocalCheckout.java:** This class implements the LocalCheckout interface which deal with the download operation (check out) of the remote project repository. The major function *doCheckout()* receives six parameters.

The first parameter is used to get a remote repository location URL, user’s account name and password which are used to authenticate him to the server. The second one specifies the remote repository URL. The third one specifies the path where to store the project into the local disk. The fourth one indicate whether to download all the project files or just a part of them (usually users only concentrate on the ‘trunk’ part of the repository). The last parameter points out the revision of the project to be downloaded.

```java
public void doCheckout(String[] args, String src, String wcpath, String inst, long rev)
```
SVNLocalCheckout.java: This class implements the LocalUpdate interface which deals with the update operation of the remote project repository. The major function 

```java
public void doUpdate(String[] args, String src, String wcpath, long upNum)
```  

SVNLocalUpdate.java: This class implements the LocalUpdate interface which deal with the update operation of the remote project repository. The major function

doUpdate() receives five parameters.

The first three parameters of the function play the same role as they do in the doCheckout() function which is touched above. The fourth parameter specifies the revision of which the user intends to update their local project files.

```java
public void doUpdate(String[] args, String src, String wcpath, long upNum)
```  

SVNShowReposContent.java: This class implements the ShowReposContent interface which is designed to display information of all the contents in the remote project repository.

The major function showContent() receives just two parameters. The first one works just like what is mentioned above to get a remote repository location URL, user's account name and password which are used to authenticate him to the server. The second parameter takes the URL of the repository whose content information is to be displayed. This function returns a data structure of ReposContentInfo type which is defined in the common package to store all the information of the content information which can be used in future (if anyone else who wants to make use of the content information, he or she can just take a parameter of the ReposContentInfo type).

```java
public ReposContentInfo showContent(String[] args, String src)
```
SVNShowReposHistory.java: This class implements the ShowReposHistory interface which is designed to show information of all the history changes (for each revision) in the remote project repository.

The major function showHistory() receives two parameters. The first one works just like what is mentioned above to get a remote repository location URL, user’s account name and password which are used to authenticate him to the server. The second parameter takes the URL of the repository whose history changes information is to be showed. Like the showContent() does, this function also returns a data structure of ReposHistoryInfo type which is defined in the common package to store all the information of the history changes in the remote repository that can be used later (so if anyone else who want to make use of the repository history changes information, he or she can just take a parameter of the ReposHistoryInfo type).

```
public ReposHistoryInfo showHistory(String[] args, String src)
```

3.4.4 Implement of the Abstract Storage API

The project information storage is an important component in our automated-download subsystem. That information makes the entire repository related operations clear to the users. Through the information, we can know some useful info like the revision of the projects downloaded or updated, where the projects files are stored and the author who check in the specified revision.

But it comes out a problem that different users would like to choose different measures that seem to be the most appropriate one to keep this project information. For instance, user A may choose the Access Database for the use of storage while user B would prefer to use the XML files to save that information. So the Abstract Storage API mentioned above just offers a basic frame for the extending work by use of different storage techniques.

In our case, to implement the Abstract Storage API for the project information storage, we choose to use the Microsoft Access Database as the medium to save the information. (We use JDBC connecting to ODBC, and then finally have the access to the Access Database).

Database package: This package includes all the Database-specified classes which are related to the Access Database information storage and all the classes in this package are the implements of the Abstract Storage API mentioned above.
**DatabaseInfo.java**: This class implements the Repository Storage interface which deals with all the project information storage work. There are two major functions in this class. The `writeCheckoutInfo()` function works in the way that when the download work is over, it immediately write some related information back into the database. Similarly, the `writeUpdateInfo()` function plays the same role as the `writeCheckoutInfo()` does, the difference is that it writes some related information back to the database when the update work is finished.

```java
public void writeUpdateInfo(String dbname, ReposHistoryInfo content, long revision)

public void writeCheckoutInfo(String dbname, ReposHistoryInfo content, long revision, String directory)
```

**3.5 SVNKit**

As is mentioned above, in our case, we selected to use an API called SVNKit to operate on the SVN repository (Subversion). So in this section, we will have a view of this special API, together with its architecture.

**3.5.1 Why SVNKit?**

Briefly, Subversion is a leading Open Source version control system. It brings Subversion closer to the Java world. SVNKit is a pure Java toolkit which implements all Subversion features and provides APIs to work with Subversion working copies, access and manipulate Subversion repositories [7]. As we can see in the Figure3.20, the SVNKit just works as a bridge between the remote/local repository and the local application/IDE.
3.5.2 Some Features of SVNKit

SVNKit features that the API is written in Java and does not require any additional binaries or native applications, which makes it compatible to most of the developing environment.

3.5.3 Purpose of SVNKit

Since SVNKit is such a flexible API for developers to use, of course, it can be applied in several occasions. In this section, we will have a look at how SVNKit integrates with different user applications.

- **Used for standard operation between Subversion server and local clients**
  The SVNKit API is adapted to most of the operations on the Subversion version control system whether it is a local working copy or a remote repository object.

- **Compatible with any object**
  The SVNKit API is fully compatible with any types of the file object and offers the version management in a flexible manner.
Adapted for web application
If some developers want to make a web application which can get access to the local/remote repository and operate on it, the SVNKit just offers such an API to deal with those connection and data transmission affairs.

3.5.4 Architecture of SVNKit
There are altogether three abstract levels of API in the SVNKit for different use in developing applications.

High Level API
In this level of API, a class called SVNClientManager is available that we can get access to a lot of interfaces, which allow us to perform almost every function that a Subversion user may need. Those functions include but not limited to checking out, updating, committing, getting history, differences and browsing repository.

Low Level API
In this level of API, the class called SVNRepository is designed to implement something unique in terms of performance. We can use it to connect to and manipulate the subversion repository directly.

JavaHL API
Native Subversion includes JNI bindings, which are available through JavaHL interface - SVNClientInterface. SVNKit implements it with SVNClientImpl class [9], so that you may just switch between JavaHL and SVNKit in the runtime. Or just replace standard JavaHL jar and its native binaries with SVNKit jar file.

3.6 Test of the subsystem
To test our automated-download subsystem, we store ten records of project information into our local Access Database in advance (see Table 3.1).
Table 3.1 Initialization Status of Access Database

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Username</th>
<th>Repository URL</th>
<th>downloaded</th>
<th>checkin date</th>
<th>author</th>
<th>revision</th>
<th>local address</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>OpenedScape</td>
<td>openscape</td>
<td><a href="https://openedscape.svn.sourceforge.net/svnroot/openscape">https://openedscape.svn.sourceforge.net/svnroot/openscape</a></td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>G Java</td>
<td>gjava</td>
<td><a href="https://gjava.svn.sourceforge.net/svnroot/gjava">https://gjava.svn.sourceforge.net/svnroot/gjava</a></td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>DerquiNsej</td>
<td>derquinsej</td>
<td><a href="https://derquinsej.svn.sourceforge.net/svnroot/derquinsej">https://derquinsej.svn.sourceforge.net/svnroot/derquinsej</a></td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Jeeves</td>
<td>jeeves</td>
<td><a href="https://jeeves.svn.sourceforge.net/svnroot/jeeves">https://jeeves.svn.sourceforge.net/svnroot/jeeves</a></td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>JavaCareWissEvolver</td>
<td></td>
<td><a href="https://jwmp.svn.sourceforge.net/svnroot/jwmp">https://jwmp.svn.sourceforge.net/svnroot/jwmp</a></td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Dozer</td>
<td>dozer</td>
<td><a href="https://dozer.svn.sourceforge.net/svnroot/dozer">https://dozer.svn.sourceforge.net/svnroot/dozer</a></td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>VisualBamnator</td>
<td>vexanimator</td>
<td><a href="https://vexanimator.svn.sourceforge.net/svnroot/vexanimator">https://vexanimator.svn.sourceforge.net/svnroot/vexanimator</a></td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then we begin to run our program.

1. The user chooses whether to have a look of the information of all the content in the specified remote repository or not.

The software is: SVNUtils

Do you want to have a look at the content in the repository of SVNUtils (Y/N)?

If he/she selected ‘yes’, then the system will display the content information.

Content in the Repository

The Repository URL is:
https://svnutils.svn.sourceforge.net/svnroot/svnutils

The UUID of this repository is: 6e911b3c-c233-0410-905c-9b4530117ae4

/branches/safranp/2/Sat Jun 30 07:40:17 CEST 2007
tags/0.01/safranp/3/Sat Jun 30 07:44:31 CEST 2007
The user chooses whether to have a look of the information of the history changes in the specified remote repository or not.

Do you want to have a look at the history changes in the repository of SVN Utils (Y/N)?

If he/she selected ‘yes’, then the system will display the history changes information for each revision.

Revision: 0
Author: null
Date: Tue Jun 26 00:07:12 CEST 2007
Log Message: null
The changed paths are: [] 'A' means added, 'D' means deleted 'M' means modified

Revision: 1
Author: safranp
Date: Sat Jun 30 07:01:26 CEST 2007
Log Message: Imported first version of the tools
The changed paths are: [] 'A' means added, 'D' means deleted 'M' means modified
Type: A Path: /src/net/sf/fileutils/WriterUtils.java
Type: A Path: /TODO
Type: A Path: /build/unix/embeddedlog
Type: A Path: /build
Type: A Path: /build/build.xml

Revision: 2
Author: safranp
Date: Sat Jun 30 07:40:17 CEST 2007
Log Message: Created proper SVN directory structure
The changed paths are: [] 'A' means added, 'D' means deleted 'M' means modified
Type: A Path: /trunk/lib
Type: D Path: /TODO
Type: A Path: /trunk/src
Type: A Path: /tags

Revision: 3
Author: safranp
Date: Sat Jun 30 07:44:31 CEST 2007
Log Message: Created tag for version 0.01
The changed paths are: [] 'A' means added, 'D' means deleted 'M' means modified
Type: A Path: /tags/0.01
3. The user specifies the path to store the project files

Where you want to store the working copy from the repository?

```
e:/work space/Svnutil
```

4. The user chooses whether to download all the project files or just the ‘trunk’ part of it

Do you want to download all the source files (A) or just trunk part (T) of them?

A

```
You choose to download all the source file from the repository!
```

5. The user specifies the revision of the project files to be downloaded

```
Which reversion of the repository do you want to check out?

Revision 1
```

6. The program begins to download the specified revision of the project files to the specified path.

Now checking out a working copy from 'https://svnutils.svn.sourceforge.net/svnroot/svnutils

```
A e:\work space\svnutils\trunk
A e:\work space\svnutils\trunk\build
```

At revision 3

After the download work is over, we check the specified path to see whether the project files are downloaded successfully.

![Downloaded Projects in the local work space](image)

**Figure3.24** Downloaded Projects in the local work space

Also we check whether the related information is wrote back to the Access Database

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>Unix name</th>
<th>svnrepository</th>
<th>downloaded</th>
<th>checkin date</th>
<th>author</th>
<th>revision</th>
<th>local_address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SVNUtil</td>
<td>svnutil</td>
<td><a href="https://svnutils.svn.sourceforge.net/svnroot/svnutils">https://svnutils.svn.sourceforge.net/svnroot/svnutils</a></td>
<td>yes</td>
<td>Sat Jun 30 07:01:26 CEST 2007</td>
<td>safranp</td>
<td>1</td>
<td>e:/work space/svnutil</td>
</tr>
</tbody>
</table>

**Table3.2** Updated Access Database

Then repeat the procedure mentioned above and the program downloaded all the
projects into local work space (see Figure 3.25).

Figure 3.25 All the Downloaded Projects

Also when a project has already been downloaded, we will let the user to select a revision to update the local work space.

You have already checked out the software: SVN Utils in revision 1

Do you want to update the local working copy (Y/N)?

Y

Which reversion of the repository do you want to update?

3

Updating 'e:\work space\svnutils'...

At revision 3

Also we check whether the update information is wrote back to the Access Database

Table 3.3 Updated Access Database

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>svnrepository</th>
<th>downloaded</th>
<th>checkin_date</th>
<th>author</th>
<th>revision</th>
<th>local_address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SVNUtils</td>
<td><a href="https://svnutils.svn.sourceforge.net/svnroot/svnutils">https://svnutils.svn.sourceforge.net/svnroot/svnutils</a></td>
<td>yes</td>
<td>Sat Jun  30 07: 44: 31 CEST 2007</td>
<td>safranp</td>
<td>3</td>
<td>e:/work space/svnutils</td>
</tr>
</tbody>
</table>

3.7 System Performance and Evaluation

In this section, we will see the system performance of our project according to the test result mentioned above. Also we will make some performance evaluation of our system.

3.7.1 System Performance

In our test program, we make it download 10 projects of the latest revision from the SourceForge.net website.

According to our test result, we spent altogether 167 seconds downloading 8 projects from the svn project repository (2 of the 10 projects are moved out of the project repository). As showed in the Table 3.4, we can have a look at how much time every download cost together with their file sizes.
<table>
<thead>
<tr>
<th>Projects</th>
<th>Spent Times (seconds)</th>
<th>File Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVN Utils</td>
<td>12</td>
<td>356KB</td>
</tr>
<tr>
<td>OpenedScape</td>
<td>18</td>
<td>9.95MB</td>
</tr>
<tr>
<td>G-java</td>
<td>20</td>
<td>49.7MB</td>
</tr>
<tr>
<td>Derquinse modules for java</td>
<td>11</td>
<td>239KB</td>
</tr>
<tr>
<td>Jeeves</td>
<td>3</td>
<td>27.6MB</td>
</tr>
<tr>
<td>Java CoreWars Evolver</td>
<td>20</td>
<td>1.14MB</td>
</tr>
<tr>
<td>Dozer</td>
<td>31</td>
<td>47.3MB</td>
</tr>
<tr>
<td>Smith</td>
<td>52</td>
<td>21.5MB</td>
</tr>
</tbody>
</table>

Table 3.4 Time consuming and Project File size

Test Environment:
- Computer Type: HP nx6120
- OS: Microsoft Windows XP SP2
- Processor: Intel®Pentium®M 1.60GHz
- Memory: 512MB
- Network Adaptor: Broadcom NetXtreme Gigabit Ethernet
- Network Bandwidth: 10MB

As we can see in the Table 3.3, the test result obeys such a rule that the larger the project file size is, the much more time we spent on the download operation. (Although some projects take rather little time to download despite of their relatively big file size, e.g. the project “Jeeves” is 27.6MB but our system just took 3 seconds to download it). Also some other factors such as the network environment and the response speed of the SourceForge.net website affect the download rate.

3.7.2 System Evaluation
As is mentioned in the beginning of the thesis, our aim is to make the preparation work for the project analysis task as efficient as possible, which greatly decreases the time spent on downloading phases.

Assuming that we want to analyze all the java projects in the SourceForge.net website. Referring to the survey (see Figure 3.26), there are altogether 16333 java projects stored in the sourceforge.net website.

As is listed in section 3.7.1, we spent 167 seconds on downloading 8 projects of about 157.1MB. So based on our calculation, the download rate from the SourceForge.net website under the test environment mentioned above should be 0.94MB/s which is a rather fast speed.

In the other hand, the 8 projects we tested hold 157.1MB. So on average the size of each project can be imaged as 19.625MB. (It is relatively objective since the file size of most projects in the SourceForge.net ranges from 10MB to 30MB). As a result, if we
want to get all the 16333 java projects to be analyzed, we should spent about 340984 seconds \( \approx 3.94 \text{ days} \) downloading them. If the bandwidth of the computer and the network transport rate can be even more satisfying, the time will continue to decrease into a rather low level.
4 Conclusion and Future work
This section concludes the thesis by checking the thesis goals. We also focus on some future developments which will make our work adaptable to different environments and convenient to use.

4.1 Did we reach our goal?
In section 1.2, we presented our thesis goal, and for each goal we attach some criteria respectively. Now we will check whether we have reached all these goals.

Our Goal:

● **Access a database for reading the SVN repository locations (the information which is stored before)**

Now we can read the project information from the Access Database and get the SVN repository URL.

● **Automatically download these projects into the local disk**

As is displayed in section 3.6, our system can checkout all the projects in the database and store them in our local disk.

● **Download a selection or all of the projects stored in the database**

Our system makes it possible for the user to select how much he/she wants to download.

● **Awareness if a project has already been downloaded**

If a project has already been downloaded into our local disk, we set the tag information “downloaded” to be true. So when the system checked the tag information, we are aware of its existence.

● **Download the latest version of the project from SVN**

If users do not specify any special revision, our system just checkout the latest revision by default.

● **Download multiple earlier versions of the project from SVN**

Users can specify any revision (ranges from the first revision to the latest revision) of the projects to be checked out.

● **Write some feedback information back to the Access Database when the download process finished.**

As showed in the Table3.2, our system successfully wrote the feedback information into the Access Database.
● **Update the already downloaded project into a certain version**

If our system found that the project is already downloaded, it will let the user select the revision of the project to update.

On the other hand, we developed the automated-download subsystem in a flexible manner with emphasis on the reusability of the system and adaptability to different application environment. We solved these problems by using several abstract interfaces which can be implemented in different manners by different users.

To summarize, we have implemented a system which makes the latter preparation work of project analysis task much easier and more efficient.

4.2 **Future work**

Since our system is well adapted to the SVN repository, some users also want to have the possibility to use it in some other kinds of projects repository like the CVS repository. As we can see in the SourceForge.net website, some projects afford both the CVS repository and the SVN repository while other projects only work well with the CVS repository which makes our subsystem useless.

It is also a better idea to make this subsystem united with Eclipse, which seems to make it as an eclipse plug-in. Then the automated-download subsystem can be used in a rather convenient way.
5 Reference

Appendix A - SourceForge Connection Information

The following configuration settings are used to access a SourceForge.Net-hosted SVN repository [10]:

- **Hostname**: PROJECTNAME.svn.sourceforge.net (PROJECTNAME is the project's UNIX name)
- **Port**: 443
- **Protocol**: HTTPS
- **Repository Path**: /svnroot/PROJECTNAME (PROJECTNAME is the project's UNIX name)
- **Username**: Your SourceForge.net username for SVN write operations, none will be requested otherwise.
- **Password**: Your SourceForge.net user password for write operations, none will be requested otherwise.
Appendix B – SourceForge.net User Configuration

OS Platforms
The following OS are supported for access to the SourceForge.net website [10]:
- Microsoft Windows 98, 2000, and XP
- Apple Mac OS X
- Recent Linux distributions
- Recent FreeBSD releases

Web Browser Software
The following browsers are known to work reliably with SourceForge.net, without compatibility problems [10]:
- Mozilla 1.7.11 and above
- Mozilla Firefox 1.0.6 and above
- Mozilla Camino 0.8.4 and above (Mac OS X)
- Microsoft Internet Explorer 6 and above
- Netscape 8 and above
- Safari (Mac OS X)

Web Browser Requirements [10]
- JavaScript, which allows to do certain types of dynamic user interface elements.
- Coolies, which allow storing authentication tokens and other persistent session information on the workstation.
- SSL (Secure Sockets Layer), which may encrypt data sent between the workstation and the SourceForge.net site for added security. SSL support is required for login to SourceForge.net.
Appendix C – SourceForge.net SVN Service

SourceForge.net SVN Environment
SourceForge.net currently runs the 1.3.x series of SVN software. Regular version upgrades occur based on testing of new releases and security needs.

SourceForge.net SVN Terminology
The following is a list of common terms used throughout the SourceForge.net SVN documentation [10]:

- **SVN client**: Software runs by a SVN user to access the SVN server.
- **SVN repository**: The SVN server stores a copy of the software and data that the project has uploaded. The server retains both the most recent version and every historical version (past changes). This copy of the software and data uploaded by the project is a SVN repository. Each project hosted on SourceForge.net has its own SVN repository.
- **Project UNIX name**: The unique name the project founder selected when registering a project for hosting on SourceForge.net. This value can be located on the project summary page (e.g. https://www.sf.net/projects/PROJECTNAME/) to the right of the phrase, “Project UNIX name:”
- **Working copy**: Though the SVN repository stores every version of every file that has been uploaded to the repository, when retrieving data from the SVN repository using the SVN client, only one version of each file is saved to the hard drive. The copy of the data got from the SVN server is called a "working copy", obtained using the "checkout" command.
- **Module**: Unlike CVS, SVN has no concept of a module.
- **Trunk**: Development with SVN progresses similar to that of a tree. The main development occurs against the trunk. Conceptually, it is identical to that of the trunk in CVS.
- **Branch**: A split of development off of the trunk or another branch. This allows for development to diverge either permanently or temporarily for a variety of possible reasons.