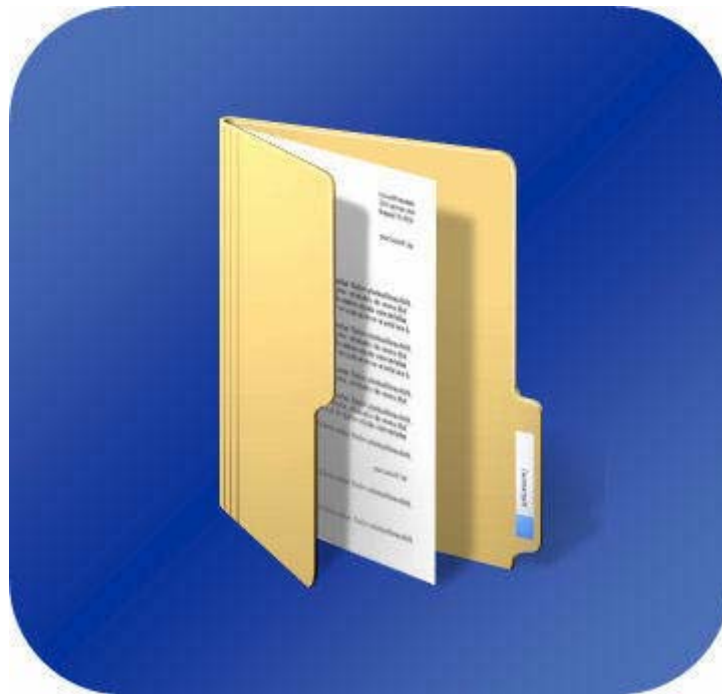


Improving a Software Corpus

November 2007 – February 2008



Ted Pei-Hsuan Han
Software Engineering
University of Auckland
Auckland City
10 February 2008

Summary

During the summer between November 2007 and February 2007 I worked at the University of Auckland.

The Faculty of Science Summer Studentship research project I was involved with deals with the improvement of an existing Java software corpus. The project duration is approximately 9 weeks and consisted of 400 hours of Sub Professional work.

In this report I shall introduce the Qualitas Corpus and the way it is organized to support empirical studies. This is followed by a detailed description of work that was done throughout the project. The final section is a discussion on future work.

Skills I have gained during summer include; experience using Perl programming language, an increased understanding about distributed file systems, and most importantly designing of software solutions while the requirements constantly change.

The work that was done during this project increased the level of confidence that can be had in the quality of Qualitas Corpus. A distribution of Qualitas Corpus was released in mid January.

Acknowledgements

I would like to thank The Faculty of Science of the University of Auckland for the funding of this project.

I would also like to thank my project supervisor, Associate Professor Ewan Tempero from the University of Auckland for his constant assistance and in providing the overall direction of the project.

Table of contents

1. Introduction	1
2. Background	1
2.1 Andrew File System (AFS)	1
2.2 Perl	1
2.3 Qualitas Corpus	2
2.3.1 Data Storage	2
2.3.2 Logical Structure	3
3. Work Place	5
3.1 Nature of Organization	5
3.2 Staff Organization	5
3.3 Building Layout and Facilities	5
4. Work Done on Qualitas Corpus	7
4.1 Overview	7
4.2 Working with Applications	9
4.2.1 Organization	9
4.2.2 Process	10
4.2.3 Techniques	11
4.2.3.1 Using the documentation provided	11
4.2.3.2 Check against the provided source distribution	12
4.2.3.3 The file names as clues	12
4.2.3.4 Compare jar fiels for duplicate classes	12
4.2.4 Tools	12
4.2.4.1 determineSrcPkgs.pl	12
4.2.4.2 showDuplicates.pl	13
4.2.4.3 showNestedJar.pl	14
4.2.5 Progress	14
4.3 The properties file	14
4.3.1 Automated Checking	16
4.3.1.1 Format Checking	16
4.3.1.2 Completeness Checking	16
4.3.1.3 Correctness Checking	17
4.3.1.4 Problem Reporting	17
4.3.2 Generating Reports	18
4.4 Distributing the Corpus	19
4.4.1 How it works	19
4.4.1.1 .install file	20
4.4.1.2 distribution.pl	20
4.4.1.3 install.pl	21
4.4.2 Distribution criteria	22
4.5 Testing Corpus Distribution	22
4.5.1 .test_install.pl	22

4.5.2 distribution_test.pl	22
4.5.3 Automation	23
4.5.3.1 checkInstall.pl	23
4.6 Adding Applications to the Corpus	23
4.6.1 Tools Created	24
4.6.1.1 setupApplications.pl	24
4.6.1.2 checkFolders.pl	24
4.6.1.2.1 Structure Checking	24
4.6.1.2.2 Completeness Checking	25
4.6.1.2.3 Correctness Checking	25
4.6.1.2.4 Problem Reporting	25
4.6.1.3 generateInstall.pl	26
4.6.1.3.1 Procedure	26
4.6.1.3.2 Supported types	27
4.6.1.4 depositApplication.pl	27
4.6.1.4.1 Usage	27
4.6.1.4.2 Procedure	27
4.6.2 How it works	27
4.7 Tools for Corpus Support	29
4.7.1 backupProperties.pl	29
4.7.2 backupInstall.pl	29
4.7.3 listLatestVersion.pl	29
4.7.4 listNonOpenSource.pl	29
4.8 Java Source Code	29
4.8.1 Source Criteria	30
4.8.2 .src	30
4.8.3 compareSrc.pl	31
5. Future Work	32
5.1 Work towards checking all source code	32
5.2 Applying Patches and Updates to the corpus distribution	32
5.3 Query the corpus for applications	32
5.4 Create a classification for applications	32
5.5 Adding more applications	32
5.6 New versions of existing applications	32
5.7 Accommodate applications other than Java	33
6. Conclusions	33
7. References	34
Appendix I: List of Applications in Qualitas Corpus	35
Appendix II: Organization Structure of Workplace	37

List of Figures

Figure 1: Physical Structure of Qualitas Corpus	2
Figure 2: Logical Structure of Qualitas Corpus	3
Figure 3: Construction of Logical Structure	4
Figure 4: City Campus	6
Figure 5: Floor Layout	6
Figure 6: Journal on Qualitas Research Group Wiki	7
Figure 7: Project Timeline	8
Figure 8: Application Organization and Preparation	10
Figure 9: Hertrix-1.8 – Standard Directory Structure	11
Figure 10: determineSrcPkgs.pl – Output	13
Figure 11: showDuplicates.pl – Output	13
Figure 12: showNestedJar.pl – Output	14
Figure 13: The properties file	16
Figure 14: Generated Report	18
Figure 15: Creating a Corpus Distribution	19
Figure 16: Corpus Reconstruction	20
Figure 17: Distribution Structure	21
Figure 18: checkInstall.pl – Output	23
Figure 19: setupApplications.pl	24
Figure 20: checkFolders.pl – Expected Structure	25
Figure 21: Generated .install File	26
Figure 22: Setup Application Structure	28
Figure 23: Preparing new Applications	28
Figure 24: Deposit Applications into the Corpus	29
Figure 25: .src File	30
Figure 26: compareSrc.pl – Output	31

1. Introduction

This report describes work done over summer improving a software corpus to support repeated studies. It was done primarily for the Faculty of Science Summer Studentship research project but it doubles as practical work experience for my Engineering degree at the University of Auckland.

Replication of studies involves repeating studies in different settings and with different samples to investigate if similar findings are obtained. As well as giving credibility to the research, it also leads to findings that can be generalized. A standard corpus of source code is useful for doing replicated studies. In order for the studies to be meaningful there must be agreement as to exactly what is in the corpus, that is, what the "standard" actually is.

There are several issues deciding on what is contained in a software corpus. Just saying it contains "applications" merely begs the question as to what an "application" is. Does it include all the standard library or third-party code? Does it include the test code? Does it include installers or other infrastructure support? If there is both source code and compiled code, what happens if the two don't appear to agree (e.g., compiled classes not in the source or vice versa)?

The corpus contains metadata for each application. These metadata are stored in text files. There is a need for tools that will manage and check these files.

The aim of this project was to address these issues through improving the quality of an existing Java Software Corpus (Qualitas Corpus), develop a means for this corpus to be distributed to interested parties and provide a set of support tools.

2. Background

This section of the report provides information on the Qualitas Corpus, the existing software corpus this project aimed to improve, also to provide some background information about Andrew File System and Perl.

2.1 Andrew File System (AFS)

The Andrew File System (AFS) is a distributed networked file system developed by Carnegie Mellon University as part of the Andrew Project. Its primary use is in distributed computing [1]. The University of Auckland uses AFS for file storage. Files are stored on what is known as AFS volumes. These volumes can be mounted and access on any machine connected to the University Network.

2.2 Perl

Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks.

Its major features are that it's easy to use, has powerful built-in support for text processing [2] and calling system facilities. These features make Perl a suitable language for work on this project and hence all tools outlined in this report are written in Perl.

2.3 Qualitas Corpus

As mentioned in the introduction, the Qualitas Corpus is an organized collection of software systems intended to be used for empirical studies in software engineering. The primary goal is to provide a resource that supports reproducible studies of software [3].

It was initially conceived and developed by Hayden Melton for his PhD research during 2005 [4]. As of the November 2007, at the start of this project, it contained approximately 100 Java applications. The Corpus described in this section of the report reflects its state at the beginning of the project however its main structure remains unaltered throughout this project.

2.3.1 Data Storage

The Qualitas Corpus is physically stored on the University of Auckland network. The file backup system used by the university imposes a limitation on the size of each volume making it unpractical to have any AFS volume over the size of 8 GB [5]. As a result of this, several AFS volumes have to be used to house the corpus.

Each AFS volume is mounted at directories named 'corpusN'. Where $N = 1, 2, \dots, n$. Data for applications are then stored in directories below this level. The naming convention used on these directories is 'appname-version_id'. Where 'appname' is used to identify the application and 'version_id' is used to uniquely identify an application to be of a particular version. A visual representation this is shown in figure 1.

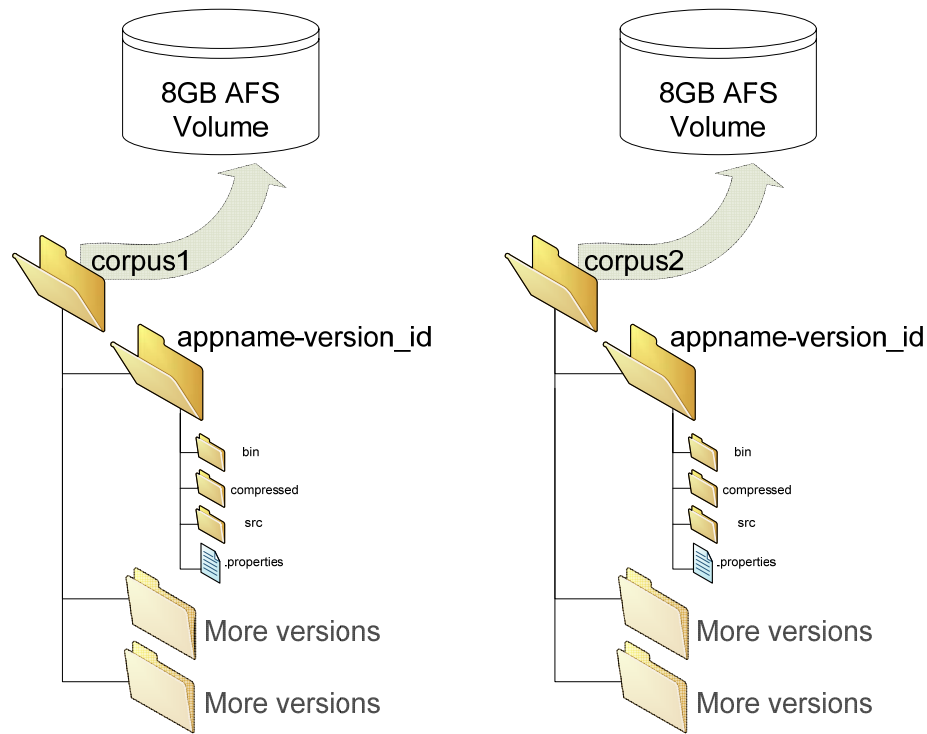


Figure 1: Physical Structure of Qualitas Corpus

2.3.2 Logical Structure

Having data spanning across several different volumes could make organizational tasks difficult. Therefore a separate logical structure is used to bind data together. This allows the Qualitas Corpus to be logically organized in a hierarchical manner. Multiple "versions" or "releases" of the same application in the corpus these are grouped together under the common application name. Each version is then distinguished by the application name and the version identification [6].

The logical structure of the Qualitas Corpus is shown in figure 2.

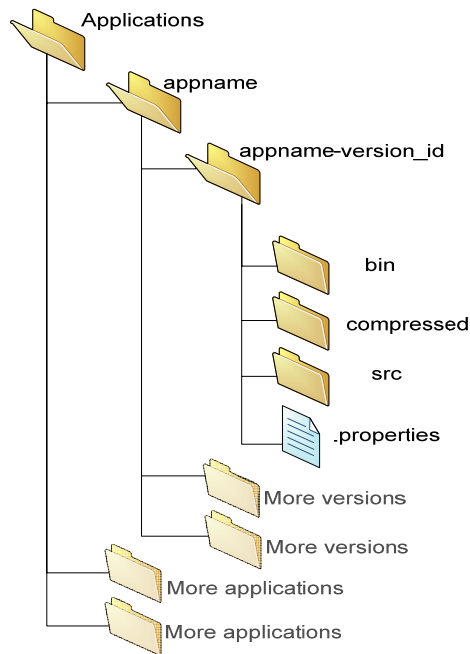


Figure 2: Logical Structure of Qualitas Corpus

Applications

The top-level directory.

appname

This directory contains everything for the specified application. It typically will contain only subdirectories corresponding to each version of the application in the corpus.

appname-version_id

This directory contains everything for a specific version of the application.

compressed

This directory contains what was retrieved from the application download site with no modification.

bin

The compiled form of the application version as provided in the deployment form.

src

This contains the source code to a particular version of an application.

.properties

File containing all metadata relevant to a particular version of an application.

The logical structure of the corpus is constructed by the creation of a hierarchy of directories. Many symbolic links were used to point to the physical data (Figure 3). This effectively combines all data across different volumes together in a single structure.

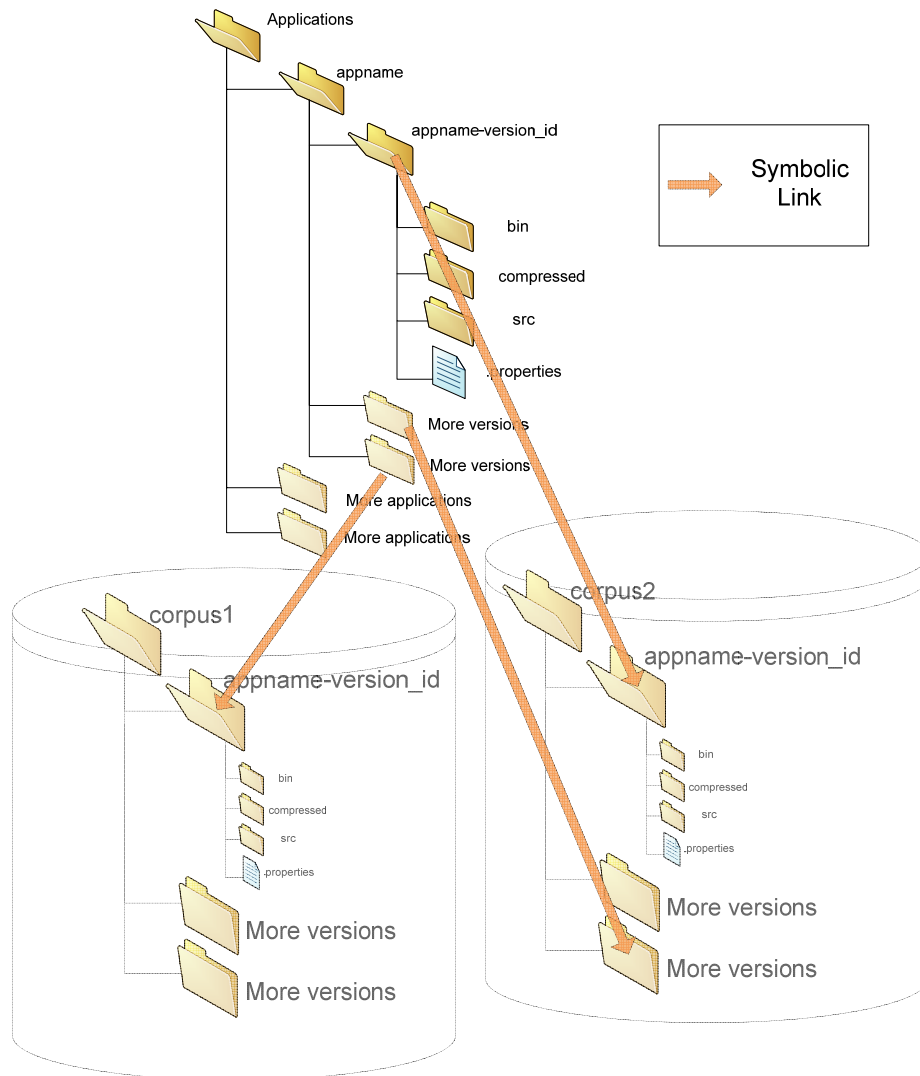


Figure 3: Construction of Logical Structure

3. Work Place

This project doubles as practical work experience. Because of this, a description of the work place is warranted.

3.1 Nature of Organization

University of Auckland is a research-led University, first established in 1883. Its mission is to be internationally recognized for the excellence of its teaching and research. It is also New Zealand's largest university, teaching and research being conducted across eight faculties [7].

Today, The University of Auckland hosts approximately 40,000 students on five Auckland campuses with around 5000 students enrolled for postgraduate studies, 1200 of whom are undertaking doctorates [7].

3.2 Staff Organization

According to the University of Auckland website, it currently employs 2,796 academic staff and 2,818 general staff, a total of 5,614 full time employees [7].

The head of the University is the Vice-Chancellor, Professor Stuart McCutcheon, the chief academic and administrative officer and the employer of all staff. Each of the eight faculties is headed by a Dean. The Dean is responsible for the management of teaching, research and administrative tasks of a faculty. Most faculties comprises of several departments, run by the Head of Department.

I report directly to my project supervisor, Associate Professor Ewan Tempero. He reports to the Head of Department of Computer Science, Associate Professor Robert Amor, who reports to the Dean of Science, Professor Dick Bellamy.

A complete organization structure of the University is included in Appendix II.

3.3 Building Layout and Facilities

My place of work is located on the University City Campus. The City Campus is located in the heart of Auckland City, covering 16 hectares of land. It provides a full range of amenities, including cafés, health services, libraries, childcare facilities, and a recreation centre.

This building I was working in is building 303, The Department of Computer Science. (Figure 4)

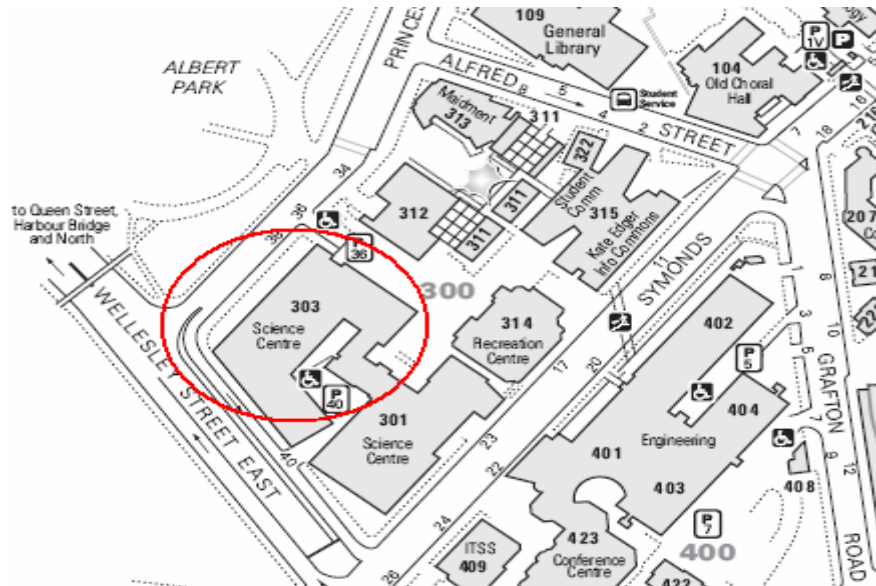


Figure 4: City Campus

I work on level 4 of this building. The layout of this floor is shown in figure 5. This floor contains several offices of lectures and professors. Other people working in this area include postgraduate students and other students doing research projects.

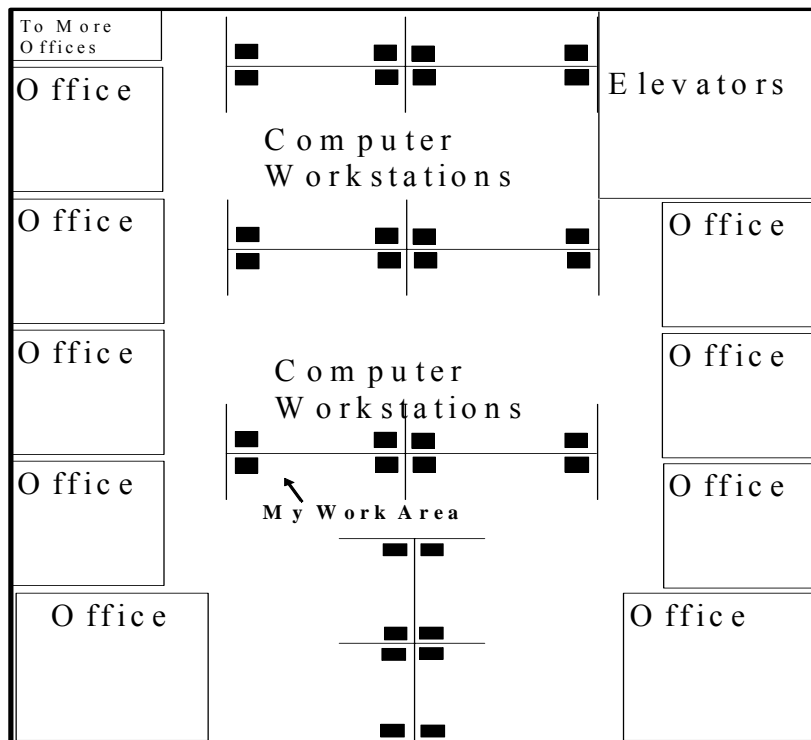


Figure 5: Floor Layout

4. Work Done on Qualitas Corpus

This section of the report describes the work that was done for the duration of the project.

4.1 Overview

The majority of work involves improving the quality of the Qualitas Corpus. While the existing corpus contained approximately 100 software applications, it lacked consistency. These software applications were deposited into the corpus by several different people. There was no standard process in doing this. More importantly there were no criteria to specify what pieces of data we must have before an application can be added in our corpus. In example, some applications in the corpus are missing source code, some application is missing compiled binary code, and there might be code that is duplicated. Because of this, one cannot be totally confident with results obtained from studies performed on the corpus.

Every effort is made to ensure existing corpus content is as accurate and complete as possible because a high degree of confidence is needed before distribution of the corpus to other parties can commence.

Furthermore, a quality control measure is developed for adding new content to the corpus. This greatly helps with maintainability as the corpus expands.

A journal was kept and updated daily during the project. It is stored on the Qualitas Research Group Wiki. It contains a record of all work done, any decisions that were made and tracks the progress. (Figure 6)

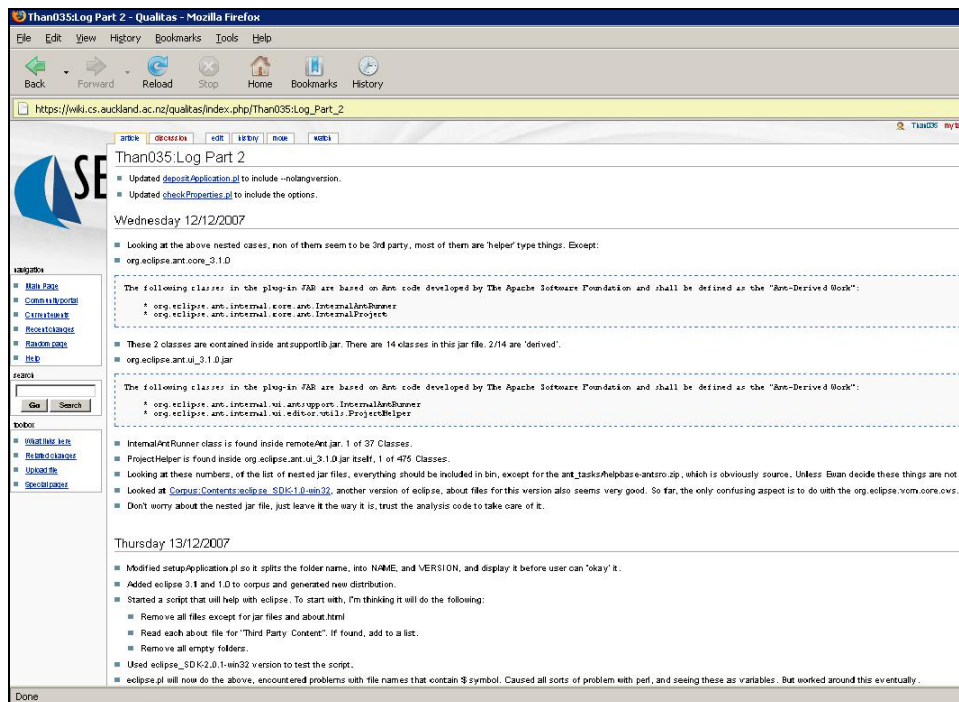


Figure 6: Journal on Qualitas Research Group Wiki

Work on the project spans over a 9 week period, commencing mid November 2007. A timeline of this period is shown in figure 7.

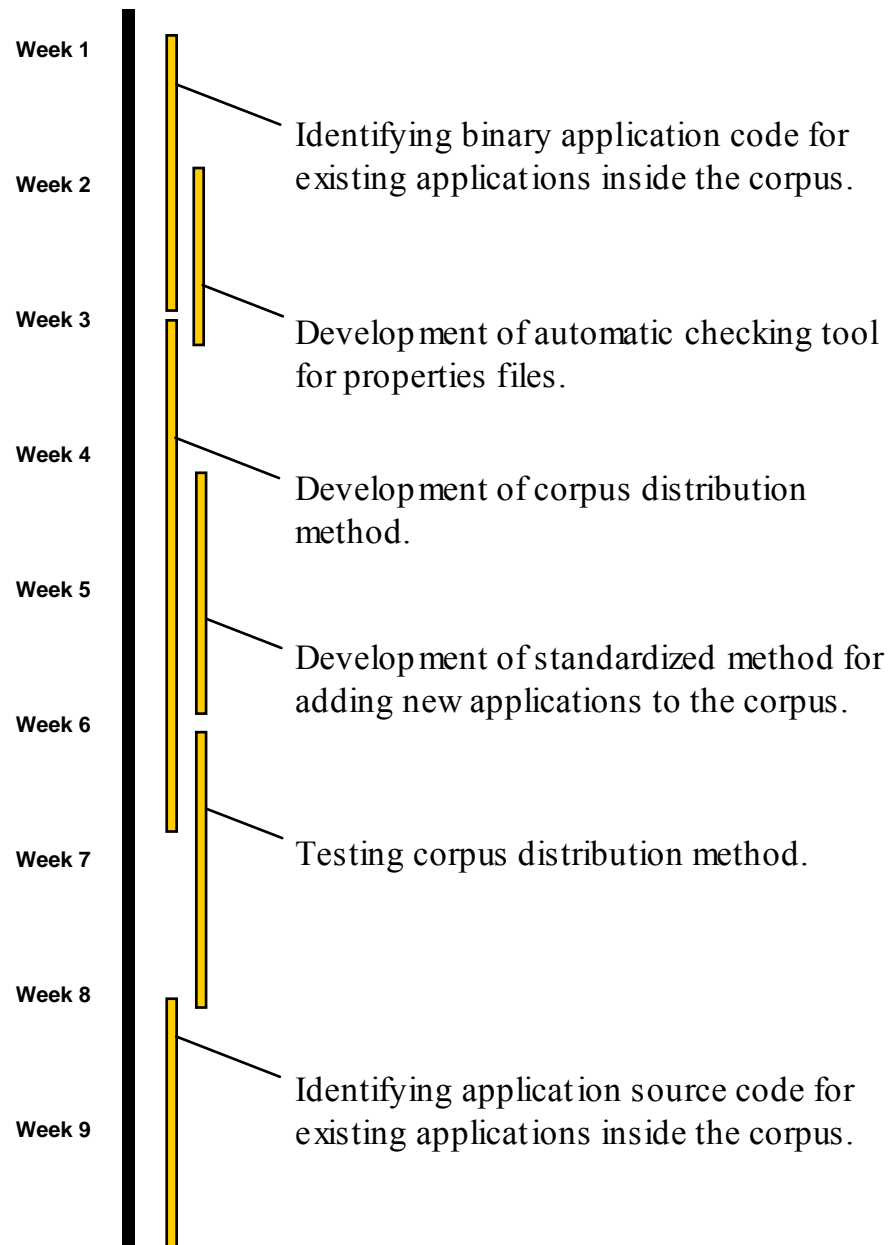


Figure 7: Project Timeline

4.2 Working with Applications

Many of the applications in the Qualitas Corpus require third-party software to run or deploy or build. Sometimes the required third-party software are distributed along with the application release other times not.

Given that such software is usually not under the control of the developers of the application, including it in the analysis would be misleading in terms of understanding what decisions have been made by developers for that application [8].

There is a need to determine what is and what is not, part of the application code. This turned out to be a non trivial task, some of the issues encountered were:

- There is no common format for organizing distributions across different applications.
- Some applications are distributed as a single jar file with no indication as to what classes are third party software codes.
- Third party software may have been modified. There is often little indication to tell us if the changes were significant or merely a different package declaration.
- Large applications are distributed with hundreds of jar files and tens of thousands of classes.

4.2.1 Organization

Applications are typically distributed in the form of one or more compressed archive files. In example, the application “hertrix-1.8” contains two zip archives, a binary release and a source release. This can be seen in figure 8.

The type of content found inside these archives varies significantly from application to application. At bare minimum there should be jar files containing byte code. Other files we can expect to see are; shell scripts, configuration files, documentation, license agreements and images to name a few.

In order to perform empirical studies on the corpus, applications first have to be prepared. This is done by organizing them into a standard directory structure that supports experimentation; this structure was described in section 2.3 of this report.

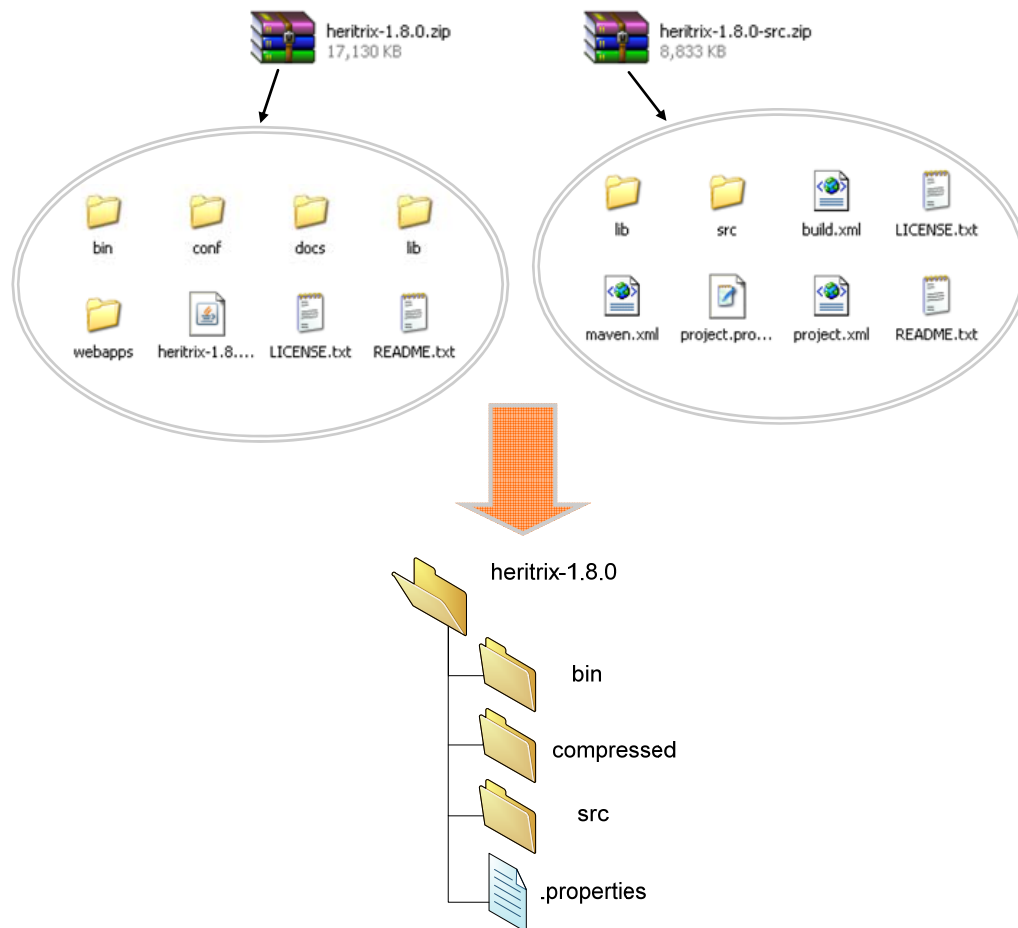


Figure 8: Application Organization and Preparation

4.2.2 Process

The process of organizing a newly downloaded application is shown in figure 8.

- Application byte code is to be extracted into the `<bin>` directory.
- The compressed archives are to be moved into the `<compressed>` directory with no modification.
- All java source code and other miscellaneous items are to be extracted into the `<src>` directory.
- Metadata is to be recorded in the `.properties` file. Refer to section 4.3 of this report for more details.

The most complex part this process is in determining the contents of the `<bin>` directory. Studies currently being done on the corpus are performed on jar files containing compiled java byte code. Therefore only jar files are to be kept inside this directory.

A decision was made not to include something in a given application in this directory if it could also appear in some other application in the corpus. This will avoid double-counting of code measurements that are done over the entire corpus [8].

The end result for “hertrix-1.8” is shown in figure 9.

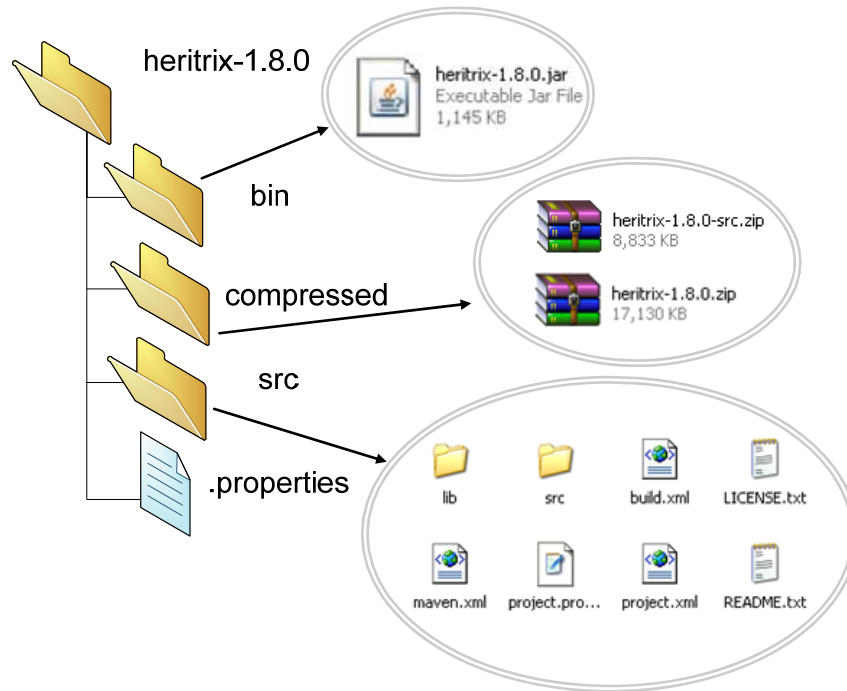


Figure 9: Hertrix-1.8 – Standard Directory Structure

4.2.3 Techniques

The process of determining the contents of <bin> outlined above is an expensive one. It is a very time consuming task and potentially error prone.

A number of techniques can be used to do this, although the applicability of each technique depends on the application being prepared. A summary of all techniques used are outlined below:

4.2.3.1 Using the documentation provided.

A well documented application would clearly identify what third-party systems are included in their distribution and or what packages application code belong to. Using this information we can filter out all unwanted extraneous material from <bin> directory.

4.2.3.2 Check against the provided source distribution.

It is rare for an application to distribute source code of third-party systems. Therefore compiled byte code for which no source is provided could be identified as third-party systems and discarded from <bin> directory. There are exceptions, but most of time this is found to be an effective technique.

4.2.3.3 The file names as clues.

There are common third-party jar files that appear in many applications throughout the corpus. The names of jar files can be used as an indication of its contents. Identification of third-party content can then be made through the comparison of class file names and file sizes against known jar files belonging to other applications.

4.2.3.4 Compare jar files for duplicate classes.

Some applications provide example code, different jar files to use on different operating systems or split their application into many optional modules. This can lead to identical pieces of code appearing more than once. To prevent double-counting of code measurements, the duplicate code should be removed from the <bin> directory.

4.2.4 Tools

Much of the process outlined in section 4.22 is manual work that has to be performed by a human. To increase efficiency and reduce chances of error, tools were developed to aid the process. These tools are aimed to produce information that will enable the human operator make appropriate decisions faster.

4.2.4.1 determineSrcPkgs.pl

Given the path to a directory, it will seek out all jar files contained within it. It lists each jar file along with all packages contained within it. Having package names are helpful, they can provide an indication to the nature of code inside the package, they can be compared with packages found within the <src> directory. The output of this tool given the <bin> directory of “ant-1.1” is shown in figure 10.

```

$ ./determineSrcPkgs.pl  ant/ant-1.1/bin/

ant.jar ->  com/oreilly/servlet
ant.jar ->  org/apache/tools/ant
ant.jar ->  org/apache/tools/ant/taskdefs
ant.jar ->  org/apache/tools/tar
jaxp.jar ->  javax/xml/parsers
parser.jar -> com/sun/xml/parser
parser.jar -> com/sun/xml/tree
parser.jar -> com/sun/xml/util
parser.jar -> org/w3c/dom
parser.jar -> org/xml/sax
parser.jar -> org/xml/sax/helpers

```

Figure 10: determineSrcPkgs.pl - Output

4.2.4.2 showDuplicates.pl

Given the path to a directory, it will seek out all jar files contained within it. For each jar file, it will compare the classes inside with all other jar files in order to locate any duplication of code. The number of redundant files in each jar file (If any) is listed. This tool can be used to remove jar files that are redundant and it can also be used to identify third-party content by comparing the jar files in question with other jar files known to be third-party software. The output of this tool given the <bin> directory of “spring-framework-1.1.5” is shown in figure 11.

```

$ ./showDuplicates.pl  spring-framework/spring-framework-1.1.5/bin/

[spring-framework/spring-framework-1.1.5/bin/spring-framework-1.1.5/dist/spring-aop.jar]
[100%] 136 of 136 class files are redundant.

[spring-framework/spring-framework-1.1.5/bin/spring-framework-1.1.5/dist/spring-context.jar]
[100%] 231 of 231 class files are redundant.

[spring-framework/spring-framework-1.1.5/bin/spring-framework-1.1.5/dist/spring-core.jar]
[100%] 195 of 195 class files are redundant.

[spring-framework/spring-framework-1.1.5/bin/spring-framework-1.1.5/dist/spring-dao.jar]
[100%] 211 of 211 class files are redundant.

[spring-framework/spring-framework-1.1.5/bin/spring-framework-1.1.5/dist/spring-orm.jar]
[100%] 179 of 179 class files are redundant.

[spring-framework/spring-framework-1.1.5/bin/spring-framework-1.1.5/dist/spring-web.jar]
[100%] 87 of 87 class files are redundant.

[spring-framework/spring-framework-1.1.5/bin/spring-framework-1.1.5/dist/spring-webmvc.jar]
[100%] 125 of 125 class files are redundant.

[spring-framework/spring-framework-1.1.5/bin/spring-framework-1.1.5/dist/spring.jar]
[100%] 1164 of 1164 class files are redundant.

```

Figure 11: showDuplicates.pl - Output

4.2.4.3 showNestedJar.pl

Given the path to a directory, it will seek out all jar files contained within it. For each jar file, it will check its inner content for any nested files of the format; jar, zip, tar.gz and war. It will then list any files that have been found. These nested files could also contain application code that would be missed by the above tools. It is essential that the user is made aware of the existence of these files. The output of this tool given the <bin> directory of “aglets-2.0.2” is shown in figure 12.

```
$ ./showNestedJar.pl aglets/aglets-2.0.2/bin/

[aglets/aglets-2.0.2/bin/aglets-2.0.2.jar]
1. lib/jaxp.jar
2. lib/log4j.jar
3. lib/parser.jar
4. lib/aglets-2.0.2.jar
5. lib/ant.jar
6. lib/crimson.jar
7. lib/optional.jar
```

Figure 12: showNestedJar.pl - Output

4.2.5 Progress

The process outlined in this section of the report was applied to 88 applications in the Qualitas Corpus, totaling 214 individual versions. A reasonable level of confidence can now be had in the contents of the <bin> directory for these applications.

4.3 The properties file

The properties file contains metadata about an application. There is a properties file associated with every version of applications in the Qualitas Corpus.

The properties file consists of fields of information. An example properties field is shown in figure 13. The description of each field is outlined below:

fullname

This is the full name of which the application is known by.

domain

Classifies the application.

notes

Any notes about the application.

acquisitiondate

Date when this application is first added to the Corpus.

acquisitionperson

The person that deposited the application into the Corpus.

language

The programming language the application is written in.

languageversion

Version of the particular language used.

origin

Name of the source where the application is obtained.

url

A link to where the application was downloaded from.

releasedate

Release Date of the particular version of the application.

opensource

If the application is open source, true or false.

obfuscated

Can source be obtained by reverse engineering of binary bytecode, true or false.

versionnotes

Any additional comments specific to the version.

sourcepackages

Packages containing only classes that have been written for this application.

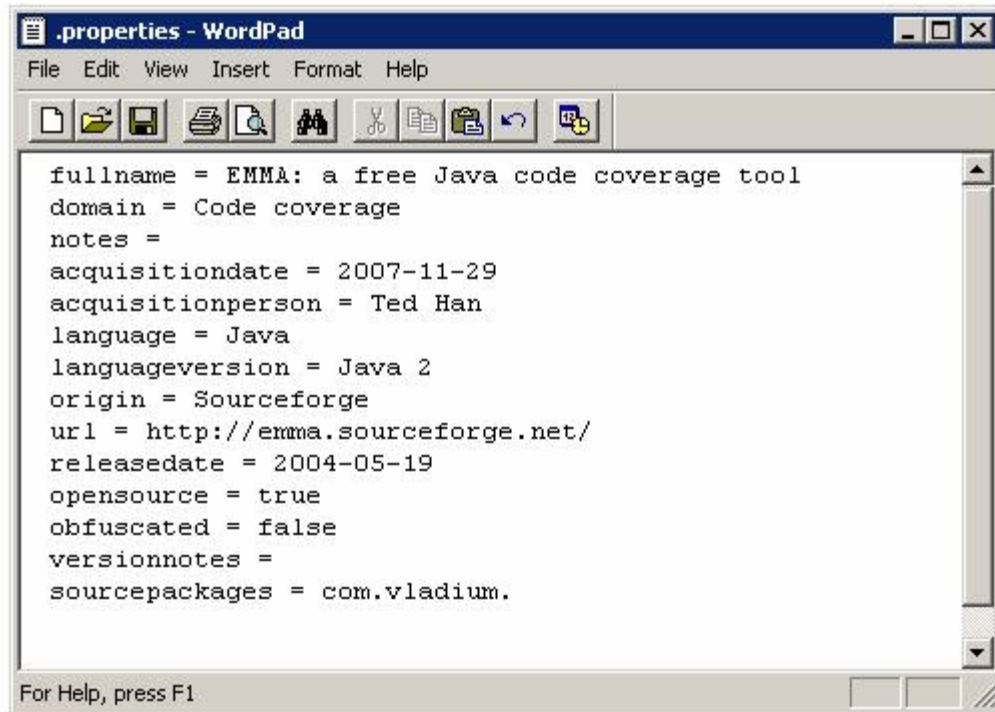


Figure 13: The properties file

4.3.1 Automated Checking

Metadata is gathered and recorded into the properties file manually. As a quality assurance measure, all properties files must be checked. Checking should be automated where possible to increase efficiency.

To support semi-automated checking of properties file, checkProperties.pl is created. This tool checks a given properties file for correctness, completeness and formatting problems. It will then report on any problems discovered.

4.3.1.1 Format Checking

1. The tool knows how many fields and hence lines should be present in a properties file. It will discard any blank lines encountered and remove them. The number of lines is compared to the expected number of fields.
2. Checks for the existence of an '=' symbol. This special character acts as a separator between field name and field value.
3. It checks the order fields occur in.

4.3.1.2 Completeness Checking

The following fields must not be empty:

- acquisitiondate
- acquisitionperson
- language
- languageversion
- origin
- url
- releasedate
- opensource
- obfuscated
- sourcepackages

4.3.1.3 Correctness Checking

1. Check that the date values of acquisitiondate and release date are:
 - In the standard form of yyyy-mm-dd.
 - The values of year, month, and date are valid.
 - releasedate is always earlier than acquisitiondate.
2. Check the URL field value has the correct syntax.
3. Check that opensource and obfuscated field values are either 'true' or 'false'.
4. Check that the package specified for sourcepackages is found inside the jar files of <bin> folder.
5. The entire properties file is printed on the screen for user to review and confirm correctness.

4.3.1.4 Problem Reporting

1. Notify that properties file cannot be opened.
2. Notify that properties file have incorrect number of lines.
3. Identify the line number of lines not having an '=' symbol.
4. Any unexpected field name encountered is printed to the screen along with the expected field name for that line.
5. Notify which field value failed the check and suggest possible solution or produce reason.

4.3.2 Generating Reports

There are over 200 versions of applications in the corpus, hence over 200 properties files. It is useful to be able to check multiple properties files at a time and generate a report of all encountered problems.

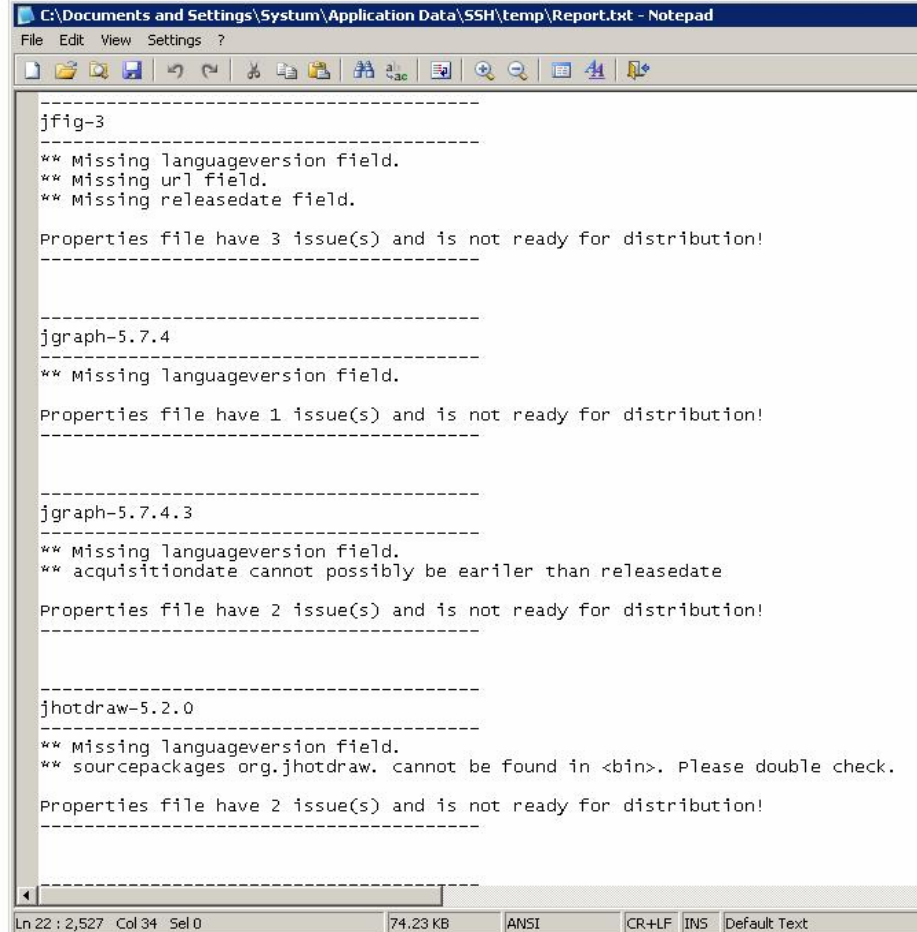
To support this, checkProperties.pl takes a quiet mode option. This means users will not be prompted for input. This allows checking of multiple files rapidly.

Another tool, checkAllProperties.pl, is used to walk over the corpus specifying properties files to check. (Figure 14)

```

$ ./checkAllProperties.pl >> Report.txt

```



```

-----
jfig-3
** Missing languageversion field.
** Missing url field.
** Missing releasedate field.

Properties file have 3 issue(s) and is not ready for distribution!
-----

jgraph-5.7.4
** Missing languageversion field.

Properties file have 1 issue(s) and is not ready for distribution!
-----

jgraph-5.7.4.3
** Missing languageversion field.
** acquisitiondate cannot possibly be earlier than releasedate

Properties file have 2 issue(s) and is not ready for distribution!
-----

jhotdraw-5.2.0
** Missing languageversion field.
** sourcepackages org.jhotdraw. cannot be found in <bin>. Please double check.

Properties file have 2 issue(s) and is not ready for distribution!
-----

```

Ln 22 : 2,527 Col 34 Sel 0 74.23 KB ANSI CR+LF INS Default Text

Figure 14: Generated Report

4.4 Distributing the Corpus

One of the main focus of this project is to ready the corpus for distribution. Several methods of distribution were considered. The two main possibilities are: direct download through the internet or on a DVD medium.

But large amounts of internet bandwidth can be costly and splitting the data across many separate disks can prove to be problematic. Therefore, for either method, the amount of data to be distributed should be kept to the minimum.

The contents of each application in the Qualitas Corpus are extracted from the contents of the <compressed> directory. The process is outlined in section 4.2 of this report. To minimize the size of the corpus distribution, only the contents of <compressed> and the properties files are included in the distributed form.

In order for this to work, a method of installing (unpacking) must be developed so the corpus can be reconstructed automatically from the distribution.

4.4.1 How it works

The first step is to walk over the corpus extracting all files that are needed for the distribution. These are the compressed archives, the properties files and a file containing instructions for reconstruction. To supplement this, the distribution must contain an installer and documentation; these are fetched from a subversion repository. (Figure 15)

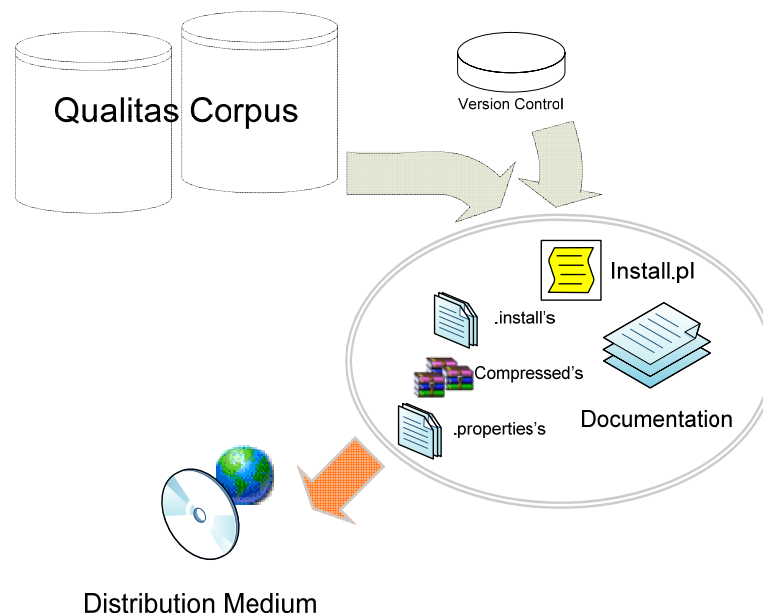


Figure 15: Creating a Corpus Distribution

This can then be distribution through either the internet or using DVDs. The corpus can be then reconstructed at a foreign file system by using the included installer. (Figure 16)

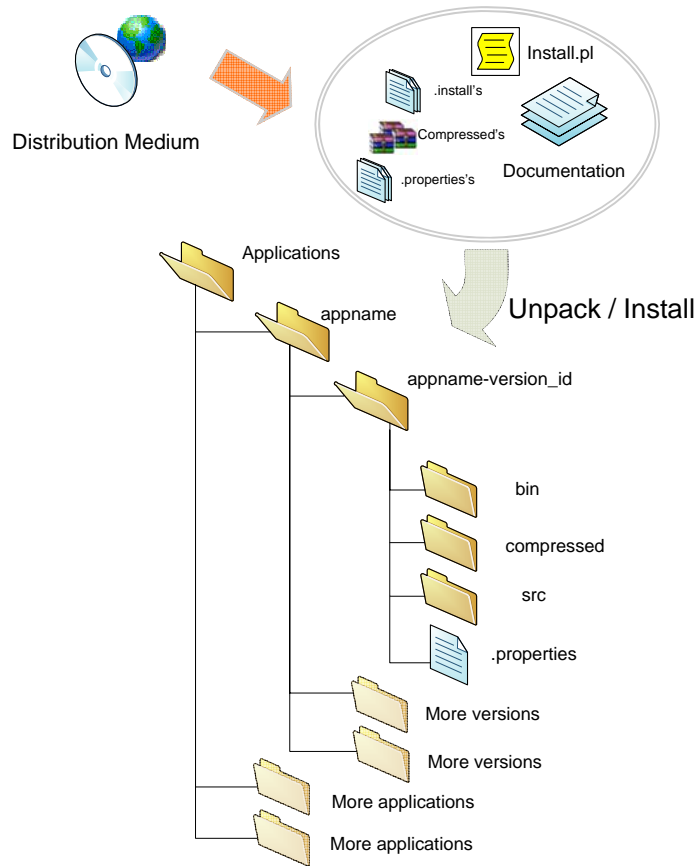


Figure 16: Corpus Reconstruction

4.4.1.1 .install file

Every application in the corpus will be distributed with an .install file. This file is a Perl script that serves two purposes:

- It specifies which compressed archives are required in order to reconstruct the application it is associated with.
- It also contains instructions to reconstruct the application by automatically going through the process outlined in Section 4.2.2 of this report.

4.4.1.2 distribution.pl

This tool is used to create a distribution of the corpus at a specified location automatically.

It walks through the corpus and does the following for each application:

- Setup directory structure for the application at the specified location.
- Consults the .install file to obtain a list of required files. These files are then copied over to the specified location.
- The properties file is copied to the specified location.

Then documentation about the Qualitas Corpus and the installer is copied over to the specified location from a subversion repository.

The result of running this tool:

- The creation of a directory called “QualitasCorpus-Date” at the specified location. Where “Date” is the date of running distribution.pl.
- All data associated with applications are placed in a directory called <Applications>. The organization of this directory is identical to the logical structure of the corpus.
- Installer is placed in a directory called <bin>.
- Documentation is placed in a directory called <docs>.

The created structure (Figure 17) is ready to be written to a DVD media or pressed into a single archive and distributed.

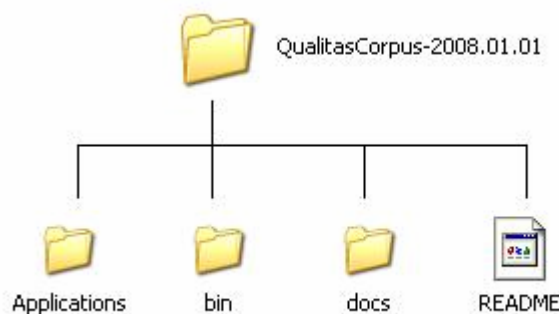


Figure 17: Distribution Structure

4.4.1.3 install.pl

This is the installer used to reconstruct the corpus from its distributed form. It works by calling individual .install files of each application.

4.4.2 Distribution criteria

For an application to be eligible for distribution, it has to:

- Be open source.
- Its properties file must be checked to be complete.
- Been through the process outlined in section 4.2.2 of this report.
- Have .install file.

4.5 Testing Corpus Distribution

Before the corpus distribution can be distributed to other parties, it must be tested to make sure it can be correctly reconstructed.

The only way to make sure everything unpacks correctly is to run the installer and check the result. This takes a significant amount of time, but it cannot be avoided without sacrificing quality.

However having to do this every time a new application is added or when an existing .install file is modified is not practical, there is a need to isolate and test these modifications and additions individually. A method to do this has been established.

4.5.1 .test_install.pl

This file is used to name an .install of an application that has never been tested, or was modified since it was last tested. For example, a newly added application will not have an .install instead it will be named .test_install.pl. Similarly before making modifications to existing .install files, they are to be first renamed to .test_install.pl.

4.5.2 distribution_test.pl

This tool performs the same task as distribution.pl. They differ in the way that instead of creating a complete distribution, it only selects applications containing .test_install.pl file. The result is a distribution of the corpus that contains a smaller number of applications and can be tested by running the installer.

4.5.3 Automation

To perform a test, we need provide a temporary space to create a distribution with `distribution_test.pl`, run the installer to reconstruct the corpus, and if this is successful, the `.test_install.pl` files need to be renamed to `.install`.

Majority of this can be automated to aid the tester and prevent errors from being introduced through the testing process itself.

4.5.3.1 checkInstall.pl

This tool is created to automate the testing method outlined above.

It decides on a temporary location and automatically creates a distribution by calling `distribution_test.pl` and then runs the installer. The tester is then prompted for confirmation of correctness, if confirmation given, it will rename `.test_install.pl` to `.install`.

Files created as a result of testing at the temporary location are removed upon the termination of the tool.

An example output of running `checkInstall.pl` is shown in figure 18.

```
=== STARTING TEST ===
Looking for candidates from corpus1...
Looking for candidates from corpus2...
  Copying ant-1.1
    Creating sub directories for ant-1.1
Complete!

Starting installation (This could take many minutes. Please Wait...)

NOW EXTRACTING APPLICATIONS
  ant-1.1...
    unpacking jakarta-ant-1.1-src.zip
    unpacking jakarta-ant-1.1-bin.zip
Complete! Corpus Successfully Distributed.

Please manually check QualitasCorpus/Applications/ant/ant-1.1. Has it been correctly unpacked?(yes/no): yes
renamed [ant-1.1]'s .test_install.pl to .install
Cleaning up temp files...DONE

TEST PASSED
```

Figure 18: checkInstall.pl - Output

4.6 Adding Applications to the Corpus

One way to ensure the content of the corpus is of high quality is to have a standard method of adding applications to the corpus. Checking is to be done during this process to ensure applications with incomplete information do not to enter the corpus. Applications are only moved into the corpus after a set standard have been met. A method for doing this was developed.

4.6.1 Tools Created

Several tools were created to support the process.

4.6.1.1 setupApplication.pl

This tool is used when adding a new application to the corpus. It will create a blank folder structure and a properties file template at a given destination.

This ensures the initial directory structure and properties file format for new applications are correct and always consistent. Figure 19 shows this being executed.

```
./setupApplication.pl /qualitas/TEST

Please type the name of this application ( Format: Application_Name-Version ): ant-1.0.0
Will create folder named [ant-1.0.0], is this correct?(yes/no): yes

Created -> /qualitas/TEST/ant-1.0.0
Created -> /qualitas/TEST/ant-1.0.0/bin
Created -> /qualitas/TEST/ant-1.0.0/compressed
Created -> /qualitas/TEST/ant-1.0.0/src
Created -> /qualitas/TEST/ant-1.0.0/.properties
```

Figure 19: setupApplications.pl

4.6.1.2 checkFolders.pl

This tool checks an application directory. It checks <bin> directory, <src> directory and <compressed> directory for completeness and correctness.

4.6.1.2.1 Structure Checking

1. Checks that the application directory has the correct structure. (Figure 20):

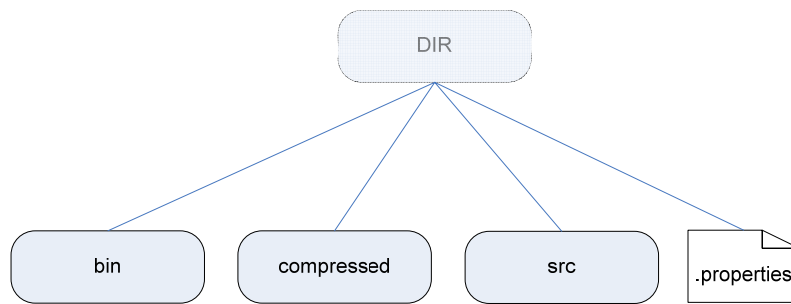


Figure 20: checkFolders.pl – Expected Structure

2. Check that there are no extra files or directories.

4.6.1.2.2 Completeness Checking

1. <bin> directory must exist.
2. <bin> must contain at least one jar file.
3. <compressed> directory must exist.
4. <compressed> directory must not be empty.
5. <src> directory must exist.
6. <src> directory must contain at least one java source file.

4.6.1.2.3 Correctness Checking

1. Check that <compressed> does not contain any sub-directories.
2. Check that the files inside <compressed> is of one of these formats; jar, tar.gz and zip.

4.6.1.2.4 Problem Reporting

1. Notify any directories that cannot be opened.
2. Notify existence of files that are not .properties or .install.
3. Notify existence of directories that are not <bin>, <compressed> or <src>.

4. Notify which sub directories (<bin>, <compressed> and <src>) failed the check and suggest possible solution or produce reason.

4.6.1.3 generateInstall.pl

This tool checks will automatically generate an .install file for a given application. If an install file already exists, it will give the user the option to use the existing one.

The generated .install file is a Perl script that contains two sub routines. The functions of which are outlined in section 4.4.1.1 of this report. Figure 21 is an example of an .install file generated using generatedInstall.pl.

```
#!/usr/bin/perl -w
use strict;

if ($ARGV[0] eq "-list") {
    my @list = list_names($ARGV[1]);
    printf("@list");
} elsif ($ARGV[0] eq "-reduced") {
    my $destination = $ARGV[1];
    install_reduced($destination);
} else {
    my $destination = $ARGV[0];
    install_reduced($destination);
}

sub install_reduced {
    my ($corpusdir) = @_;
    my $src = "emma-2.0.4019-src.zip";
    my $bin = "emma-2.0.4019.zip";
    my @files = ("emma-2.0.4019/lib/emma.jar");
    printf("\t\t unpacking $src\n");
    system("unzip -qo $corpusdir/compressed/$src -d $corpusdir/src\n");
    printf("\t\t unpacking $bin\n");
    system("unzip -qo $corpusdir/compressed/$bin @files -d $corpusdir/");
}

sub list_names {
    my ($appverdir) = @_;
    opendir(COMPRESSED, "$appverdir/compressed") || die "can't open $a";
    my @archives = grep { ! /^\.$/ and ! /^\.\/$/ } readdir(COMPRESSED);
    close(COMPRESSED);
    return @archives;
}
```

Figure 21: Generated .install File

4.6.1.3.1 Procedure

1. If the script finds that an install file already exists, it will ask the user if this one should be used. If the user answers 'yes', the script will exit here.
2. The contents of <compressed> directory are printed to the screen and the user is asked to identify the source archive and binary archive.
3. The names of all the jar files inside <bin> are extracted and then it searches through the binary archive for each jar file. Any jar files that cannot be located will be reported to the screen. The locations of jar files inside the binary archive are stored.

4. Using the information gathered from 2 and 3, the install file is generated inside the application directory. The generated file is named ‘.test_install.pl’.

4.6.1.3.2 Supported types

This tool currently handles applications distributed in the form of:

1. Non-nested zip archives.
2. Non-nested tar.gz archives.
3. Up to two-level jar files.

4.6.1.4 depositApplication.pl

This tool is used when a new application is ready to be deposited into the corpus. It will check a given application for its folder structure, the properties file, create an install file and move it into the corpus.

Running this tool is the final step in adding new applications to the corpus. This tool works by calling several other tools.

4.6.1.4.1 Usage

depositApplication.pl DIR

(Where DIR is a directory containing the application.)

4.6.1.4.2 Procedure

1. Checks the folder structure by running checkFolders.pl (Section 4.6.1.2)
2. Checks the properties file by running checkProperties.pl (Section 4.3.1)
3. Create an install file by running generateInstall.pl (Section 4.6.1.3)
4. Move DIR into the Qualitas Corpus.

4.6.2 How it works

The first step in adding a new application is to setup a new application structure at a temporary location outside of the corpus. (Figure 22)

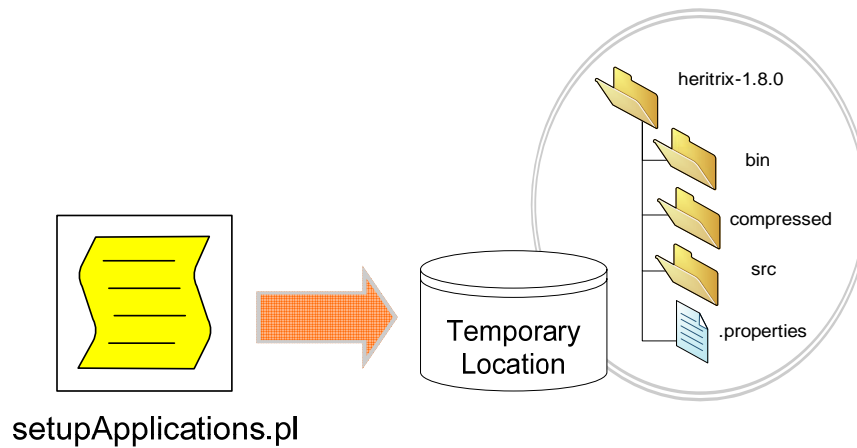


Figure 22: Setup Application Structure

The next step involves preparing of the new application using the process outlined in section 4.2.2 of this report at the temporary location. (Figure 23)

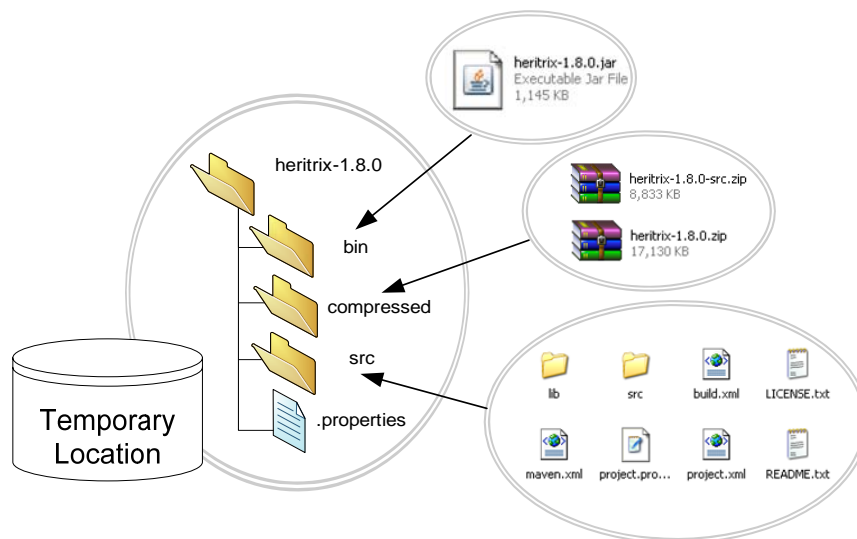


Figure 23: Preparing new Application

The final step is running depositApplication.pl. This invokes a series of checks on the application and an .install file is generated. The application is then moved into the corpus. (Figure 24)

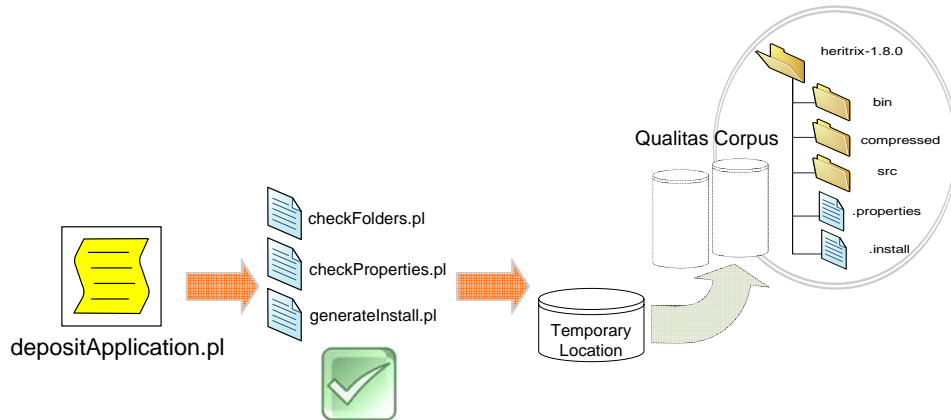


Figure 24: Deposit Application into the Corpus

4.7 Tools for Corpus Support

Several additional tools were also developed. These are used to produce information about the corpus contents and to help with the maintenance of the corpus.

4.7.1 backupProperties.pl

This tool creates a backup of all properties files in the Qualitas Corpus. The backup can be then be placed in a separate volume under version control to prevent any loss of information.

4.7.2 backupInstall.pl

This tool creates a backup of all .install files in the Qualitas Corpus. The backup can be then be placed in a separate volume under version control to prevent any loss of information.

4.7.3 listLatestVersion.pl

This tool produces a list of the most recent version of every application contained in the Qualitas Corpus.

4.7.4 listNonOpenSource.pl

This tool produces a list of application contained in the Qualitas Corpus that are not open source.

4.8 Java Source Code

Studies done using the Qualitas Corpus so far have been based on compiled byte code. However, research on java source code in the near future is a distinct possibility. The corpus needs to be further extended to support studies of this sort.

A similar process to one outlined in section 4.2.2 of this report has to be done on the source code of every application. Java source code considered to be part of an application has to be identified.

Because this is an on going process, a new field is added to the properties file, the “source” field. This field is used to identify whether an application have completed this process of identifying source code. This can be either true or false.

4.8.1 Source Criteria

Currently, in order for the source field to be true, the application must meet the following criteria:

- The <src> directory must contain java source code for every compiled java byte code contained in the <bin> directory.
- Any java source code in <src> directory not also found in compiled form in the <bin> directory must be either test code or example code.
- Have a list containing the path to every java source code that has a compiled counterpart in <bin> directory.

4.8.2 .src

This file contains a list of paths to java source files. This file is to be found under the application directory on the same level as .properties and .install files. (Figure 25)

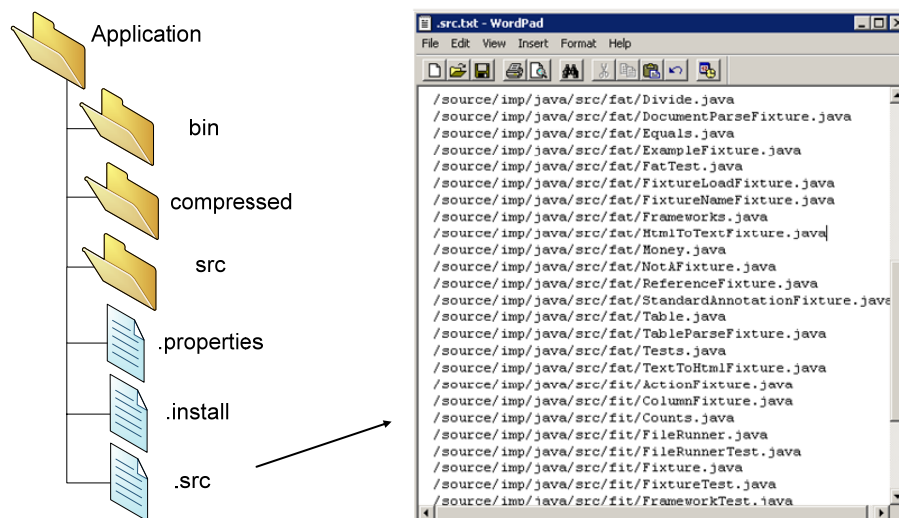


Figure 25: .src File

4.8.3 compareSrc.pl

This tool is created to make identify source code easier.

It works by obtaining a list of all compiled classes from the <bin> directory and a list of all java source code from the <src> directory, then:

- It presents a summary of number of classes found in <bin> and <src>, the union, intersection and differences between these directories.
- It lists all the top level packages found.
- It lists class files that are only found in the <bin> directory.
- It lists class files that are only found in the <src> directory.
- It can then write the intersection to use in the .src file of the application.

The output of this tool running with “ant-1.1” directory is shown in figure 26.

```
NOTES
-----
./src/jakarta-ant/lib/jaxp.jar in <src> could contain more classes.
./src/jakarta-ant/lib/parser.jar in <src> could contain more classes.

Classes in <bin>      : 70
Total classes in <src> : 87
Unique classes in <src>: 87
Union                : 88
Intersection         : 69
Difference           : 19

Top level packages in <bin>
-----
com
org

Only in <bin>
-----
com/oreilly/servlet/MailPrintStream

Only in <src>
-----
/jakarta-ant/src/testcases/org/apache/tools/ant/AllJUnitTests
/jakarta-ant/src/testcases/org/apache/tools/ant/EnumeratedAttributeTest
/jakarta-ant/src/testcases/org/apache/tools/ant/IntrospectionHelperTest
/jakarta-ant/src/testcases/org/apache/tools/ant/PathTest
/jakarta-ant/src/main/org/apache/tools/ant/taskdefs/optional/NetRexxC
/jakarta-ant/src/main/org/apache/tools/ant/taskdefs/optional/RenameExtensions
/jakarta-ant/src/main/org/apache/tools/ant/taskdefs/optional/XalanLiaison
/jakarta-ant/src/main/org/apache/tools/ant/taskdefs/optional/XslpLiaison
/jakarta-ant/src/main/org/apache/tools/ant/taskdefs/optional/ejb/DDCreator
/jakarta-ant/src/main/org/apache/tools/ant/taskdefs/optional/vss/MSVSS

Write intersection to file?(yes/no): no
```

Figure 26: compareSrc.pl - Output

5. Future Work

The task of improving and maintaining the Qualitas Corpus is an on going process. This section describes some of these tasks.

5.1 Work towards checking all source code.

Only 28 versions of applications have had their java source code identified. (Section 4.8) To support studies on the source code, more applications are required.

5.2 Applying Patches and Updates to the corpus distribution.

New corpus distributions will be released as new applications are added and or changes are made to the existing corpus. A way of patching or updating older releases of the corpus need to be implemented.

5.3 Query the corpus for applications

There should be a way to search for applications in the corpus. For example, the ability to produce a list of all applications released prior to 2001. The possibility of implementing this using a mySQL database was explored during this project, it was a low priority task and it was decided at the time, it wasn't worth the overhead time required to set up the infrastructure.

5.4 Create a classification for applications.

Come up with an appropriate classification for software applications and then classify all application contains in the corpus.

5.5 Adding more applications.

This is an on going process.

5.6 New versions of existing applications.

A way to automatically check for newer version of existing application contained in the corpus as they become available. Then automatically download and deposit the new versions into the corpus.

5.7 Accommodate applications other than Java.

The Qualitas Corpus is currently a Java only corpus. All applications contained in the corpus are written in Java. In the future the corpus should be able to support multiple programming languages.

6. Conclusions

The aim of this project is to improve the overall quality of the Qualitas Corpus to better support empirical studies and so that it can be distributed. Various different tools were developed to achieve this.

After a 9 week period the status of the corpus is as follows:

- High degree of confidence in the contents of the <bin> directory for 214 versions of applications.
- High degree of confidence in the contents of the <src> directory for 28 versions of applications.
- Automated methods of creating and testing distributions of the corpus have been developed.
- A standard process of adding new applications has been established.

The corpus is continually growing in size. New applications are now constantly being added using the process developed in this project.

The first version of the Qualitas Corpus distribution was released on the 18th of January, 2008. It contains 88 applications, 21 applications with multiple versions, with 214 versions total.

7. References

1. Andrew File System. Retrieved January 28, 2008 from http://en.wikipedia.org/wiki/Andrew_file_system
2. Perl 5.10.0 documentation. Retrieved January 28, 2008 from <http://perldoc.perl.org/perlintro.html>
3. Tempero, E. *Qualitas Corpus*. Retrieved January 28, 2008 from <http://www.cs.auckland.ac.nz/~ewan/corpus/>
4. Tempero, E. *History of the Qualitas Corpus*. Retrieved January 28, 2008 from <http://www.cs.auckland.ac.nz/~ewan/corpus/history.html>
5. Documentation: Volumes. Retrieved January 29, 2008 from <https://wiki.cs.auckland.ac.nz/qualitas/index.php/Documentation:Volumes>
6. Tempero, E. *Qualitas Corpus Content Structure*. Retrieved January 29, 2008 from <http://www.cs.auckland.ac.nz/~ewan/corpus/structure.html>
7. About The University. Retrieved February 7, 2008 from <http://www.auckland.ac.nz/uoa/about/uoa/>
8. Tempero, E. *What is an Application*. Retrieved January 30, 2008 from <http://www.cs.auckland.ac.nz/~ewan/corpus/application.html>

Appendix I: List of Applications in Qualitas Corpus

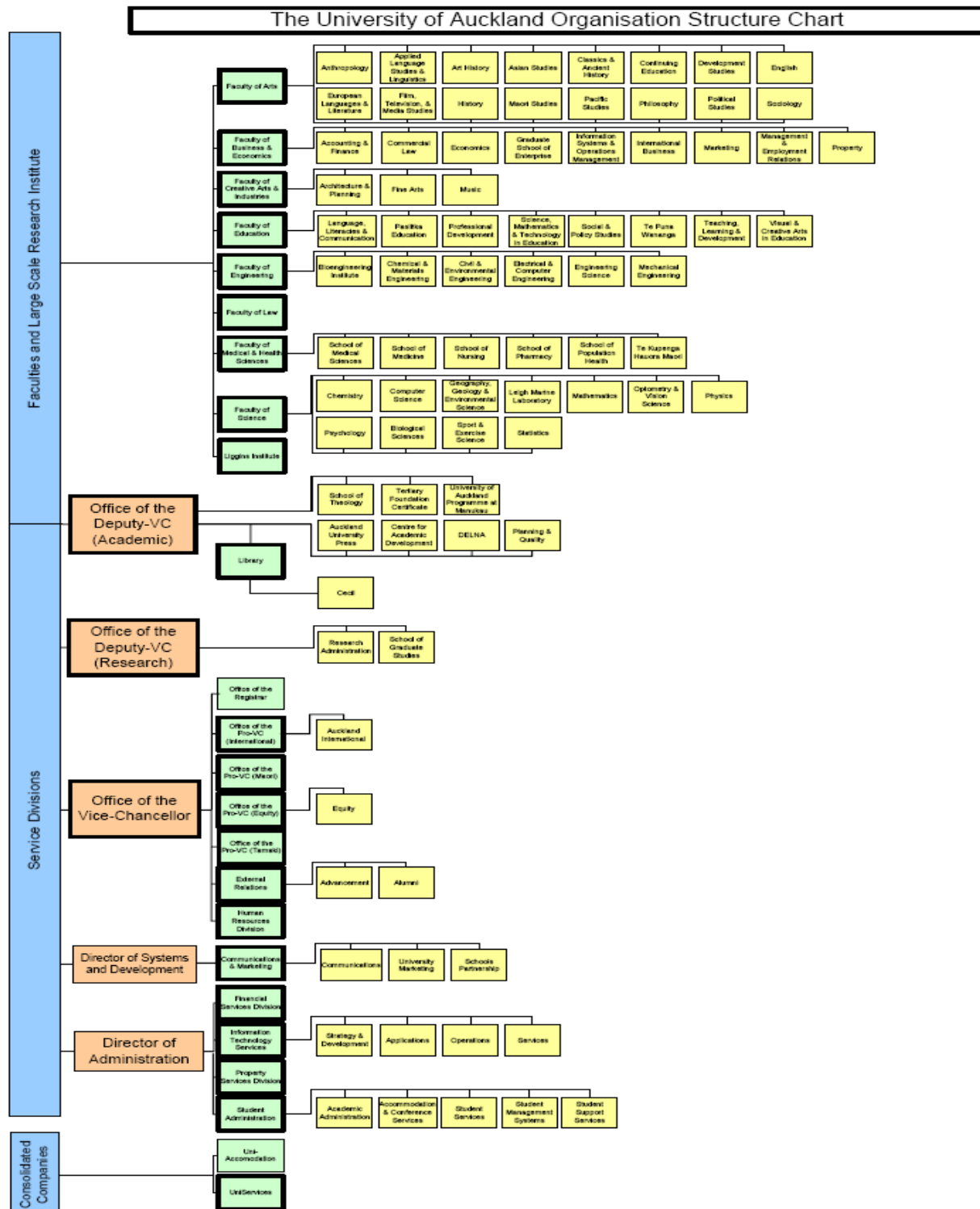
aglets-2.0.2	bluej-2.1.0	freecs-1.2.20060130
ant-1.1	c_jdbc-2.0.2	galleon-1.8.0
ant-1.2	checkstyle-4.2	ganttproject-1.11.1
ant-1.3	checkstyle-4.3	geronimo-1.0-M5
ant-1.4	cobertura-1.9	glassfish-9.0-b15
ant-1.4.1	colt-1.0.1	gt2-2.2-rc3
ant-1.5	colt-1.0.2	heritrix-1.8.0
ant-1.5.1	colt-1.0.3	hibernate-3.1-rc2
ant-1.5.2	colt-1.1.0	hsqldb-1.8.0.2
ant-1.5.3.1	colt-1.2.0	hsqldb-1.8.0.4
ant-1.5.4	columba-1.0	htmlunit-1.8
ant-1.6.0	compiere-250d	infoglue-2.3Final
ant-1.6.2	compiere-251e	informa-0.6.5
ant-1.6.5	derby-10.1.1.0	intellij-5.0.2
ant-1.7.0	displaytag-1.1	ireport-0.5.2
antlr-2.4.0	drawn-v1	itext-1.4
antlr-2.5.0	drawn-v2	ivatagroupware-0.11.3
antlr-2.6.0	drawn-v3	izpack-3.8.1
antlr-2.7.0	drawn-v4	j_ftp-1.48
antlr-2.7.1	drawn-v5	jag-5.0.1
antlr-2.7.2	drawn-v6	jaga-1.0.b
antlr-2.7.3	drawn-v7	james-2.2.0
antlr-2.7.4	drawn-v8	jasml-0.10
antlr-2.7.5	drawn-vA	jasperreports-1.1.0
antlr-2.7.6	drawn-vB	javacc-3.2
aoi-2.2	drawn-vC	jboss-4.0.3-SP1
aoi-2.5.1	drawswf-1.2.9	jbuilderfoundation-2005
argouml-0.16.1	drjava-20050814	jchempaint-2.0.12
argouml-0.18.1	eclipse_SDK-1.0-win32	jdeveloper-10.1.2.1b1913
argouml-0.20	eclipse_SDK-2.0.1-win32	jedit-4.2
aspectj-1.0.6	eclipse_SDK-2.0.2-win32	jeppers-20050607
axion-1.0-M2	eclipse_SDK-2.0-win32	jetty-5.1.8
azureus-2.0.1.0	eclipse_SDK-2.1.1-win32	jext-5.0
azureus-2.0.3.0	eclipse_SDK-2.1.2-win32	jfig-3
azureus-2.0.3.2	eclipse_SDK-2.1.3-win32	jFin_DateMath-R1.0.0
azureus-2.0.4.0	eclipse_SDK-2.1-win32	jfreechart-1.0.0-rc1
azureus-2.0.4.2	eclipse_SDK-3.0.1-win32	jfreechart-1.0.1
azureus-2.0.7.0	eclipse_SDK-3.0.2-win32	jgraph-5.10.0.0
azureus-2.0.8.0	eclipse_SDK-3.0-win32	jgraph-5.10.0.1
azureus-2.0.8.2	eclipse_SDK-3.1.2-win32	jgraph-5.10.2.0
azureus-2.0.8.4	eclipse_SDK-3.1-win32	jgraph-5.4.4-java1.3
azureus-2.1.0.0	egothor-1.3.003	jgraph-5.4.4-java1.4
azureus-2.1.0.2	emma-2.0.4019	jgraph-5.5
azureus-2.1.0.4	emma-2.0.5312	jgraph-5.5.1
azureus-2.2.0.0	exoportal-v1.0.2	jgraph-5.6.2
azureus-2.2.0.2	findbugs-1.0.0	jgraph-5.6.2.1
azureus-2.3.0.0	fitjava-1.1	jgraph-5.6.3
azureus-2.3.0.2	fitlibraryforfitness-	jgraph-5.7
azureus-2.3.0.4	20050923	jgraph-5.7.1
azureus-3.0.3.4	freecol-0.3.0	jgraph-5.7.3
bfograph-2.0.3	freecol-0.4.0	jgraph-5.7.3.1
bfopdf-2.6.4	freecol-0.5.0	jgraph-5.7.4
bforeport-1.1.28	freecol-0.6.0	jgraph-5.7.4.1

jgraph-5.7.4.2
 jgraph-5.7.4.3
 jgraph-5.7.4.4
 jgraph-5.7.4.5
 jgraph-5.7.4.6
 jgraph-5.7.4.7
 jgraph-5.8.0.0
 jgraph-5.8.1.1
 jgraph-5.8.2.0
 jgraph-5.8.2.1
 jgraph-5.8.3.1
 jgraph-5.9.0.0
 jgraph-5.9.1.0
 jgraph-5.9.2.0
 jgraph-5.9.2.1
 jhotdraw-5.2.0
 jhotdraw-5.3.0
 jhotdraw-5.4.1
 jhotdraw-5.4.2
 jhotdraw-6.0.1
 jmeter-1.8.1
 jmeter-1.9.1
 jmeter-2.0.0
 jmeter-2.0.1
 jmeter-2.0.2
 jmeter-2.0.3
 jmeter-2.1
 jmeter-2.1.1
 jmoney-0.4.4
 joggplayer-1.1.4s
 jparse-0.96
 jrat-0.6
 jre-1.1.8.010
 jre-1.2.2.017
 jre-1.3.1.18
 jre-1.4.2.04
 jre-1.5.0.06
 jre-1.5.0_14-linux-i586
 jrefactory-2.9.19
 jruby-1.0.1
 jspwiki-2.2.33
 jtopen-4.9
 jung-1.0.0
 jung-1.1.0
 jung-1.1.1
 jung-1.2.0
 jung-1.3.0
 jung-1.4.0
 jung-1.4.1
 jung-1.4.2
 jung-1.4.3
 jung-1.5.0
 jung-1.5.1
 jung-1.5.2
 jung-1.5.3
 jung-1.5.4

jung-1.6.0
 jung-1.7.0
 jung-1.7.1
 junit-2.0
 junit-2.1
 junit-3.0
 junit-3.4
 junit-3.5
 junit-3.6
 junit-3.7
 junit-3.8
 junit-3.8.1
 junit-3.8.2
 junit-4.0
 junit-4.1
 junit-4.4
 log4j-1.2.13
 lucene-1.2-final
 lucene-1.3-final
 lucene-1.4.3
 luxor-1.0-b9
 megamek-2005.10.11
 mvnforum-1.0-ga
 myfaces_core-1.2.0
 nakedobjects-3.0.1
 nekohtml-0.9.5
 netbeans-3.5.1
 netbeans-3.6
 netbeans-4.0
 netbeans-4.1
 netbeans-5.0
 netbeans-5.5
 netbeans-5.5-beta
 openjms-0.7.7-alpha-3
 openoffice-2.0.0
 openxchange-0.8.0.6
 oscache-2.3-full
 picocontainer-1.3
 pmd-3.3
 poi-2.5.1
 pooka-1.1-060227
 poseidon-3.1
 poseidon-3.2
 proguard-3.6
 quartz-1.5.2
 quickserver-1.4.7
 quilt-0.6-a-5
 roller-2.1.1-incubating
 rssowl-1.2
 sa4j-20040331
 sablecc-3.1
 sandmark-3.4
 scala-1.4.0.3
 scarab-1.0-b20
 sequoiaerp-0.8.2-RC1-all-
 platforms

servicemix-3.0-SNAPSHOT
 soot-1.0.0
 soot-1.2.0
 soot-1.2.1
 soot-1.2.2
 soot-1.2.3
 soot-1.2.4
 soot-1.2.5
 soot-2.0
 soot-2.0.1
 soot-2.1.0
 soot-2.2.3
 springframework-1.1.5
 springframework-1.2.7
 squirrel_sql-2.2final
 squirrel_sql-2.4
 struts-1.2.9
 sunflow-0.07.2
 tomcat-5.0.28
 tomcat-5.5.17
 trove-1.1b5
 webmail-0.7.10
 weka-3.4.12
 weka-3.5.7
 xalan-j_2_7_0
 xerces-2.8.0
 xmojo-5.0.0
 yed-2.3.1_02

Appendix II: Organization Structure of Workplace



Retrieved 8 February, 2008 from <http://www.auckland.ac.nz/uoa/about/uoa/run/introduction.cfm>