Apples Vs. Oranges? An exploration of the challenges of comparing the source code of two software systems.

Daniel M. German Julius Davies
Department of Computer Science, University of Victoria, Canada
dmg@uvic.ca juliusd@uvic.ca

ABSTRACT

We attempt to compare the source code of two Java IDE systems: `Netbeans` and `Eclipse`. The result of this experiment shows that many factors, if ignored, could risk a bias in the results, and we posit various observations that should be taking into consideration to minimize such risk.

1. QUESTIONS ADDRESSED

Ever since the beginning of empirical software engineering, we have dreamed of experiments in which large, industrial-quality, equivalent software systems are compared, with the goal of minimizing the threats to validity. The competitive nature of open source software, where similar products fight against each other in attempts to maximize market share, has provided natural (and observable) experiments where this dream can be realized. It is not hard to find research comparing the BSD-kernels, Linux and Android; or MySQL and PostgreSQL. There is, however, lack of research on the challenges that arise with such comparisons, and the consequent traps that researchers risk falling into. In this MSR challenge report, we compare the source code of two industrial grade Integrated Development Environments (IDE): `Netbeans`, developed by Oracle Corporation, and `Eclipse`, developed by the Eclipse Foundation. In the same spirit as [1], our goal is not to identify how similar or different they are, but to identify differences that, if not taken into consideration, might result in biased, and potentially erroneous conclusions.

2. INPUT DATA

For this study we used the source code of `Netbeans` and `Eclipse` IDEs from two different sources: their version control (VC) repositories, and their source code packages as made available in Ubuntu 10.10.

1. **VC repositories**. We downloaded the CVS (`Eclipse`) and Mercurial (`Netbeans`) snapshots offered through the MSR conference website[1] for the Mining Challenge. To the best of our knowledge, these contained complete version history for both `Netbeans` and `Eclipse` up to June 1st, 2010.

2. **Ubuntu packaged sources**. We downloaded the source packages for Ubuntu 10.10 from Canonical's archive [2]. These original sources are usually present in the archive as `.tar.gz` files. We used version 3.5.2 of `Eclipse` and version 6.5 of `Netbeans`.

MSR '2011 Waikiki, Honolulu, Hawaii

3. RESULTS AND INTERPRETATION

3.1 Version control or Source Distribution, what do we compare?

In the version control (VC) repository of `Eclipse` there are 60,300 Java source files. Are all these files part of the `Eclipse` SDK, as we know it? Can we directly compare their development to one of the source code files in the `Netbeans` VC repository? The `Eclipse` IDE is developed by the Eclipse Foundation, and its source code lives in the only VC repository of the Foundation. We could expect to see other projects under the Eclipse Foundation umbrella to be stored in the same VC.

In contrast, `Netbeans` is one of many open source products developed by Oracle. Before we can compare the VC and defect repositories of each, we need to better understand how the IDEs source code relates to the source code files stored under their corresponding VC repository.

Unfortunately, knowing which files are part of either IDE is not trivial. Both IDEs offer several versions, configured with different features. To simplify our analysis, we use as proxies the source code bundles created by Canonical, from which its Ubuntu binaries are created. These bundles (tar files) are created from the same files under their VC repositories, but contain only what is necessary to build, test and run either IDE under Linux—as decided by Canonical. We used Ubuntu 10.10's source code bundles: version 3.5.2 of `Eclipse` (`eclipse_3.5.2.orig-eclipse.tar.bz2`), and version 6.5 of `Netbeans` (`netbeans_6.5.orig.tar.gz`). While these versions are configured only for Linux, we presume that they are configured in such a way as to provide a similar experience to its users. Furthermore, we will use these bundles as a starting point for further analysis, reducing the threat to validity incurred by making such choice.

[1] `http://www.msrconf.org/`
[2] `http://archive.ubuntu.com/ubuntu/pool/universe/`

Eclipse's source code bundle for Ubuntu contains only 16,361 different Java source files (compared to 60,300 in its VC repository). In other words, the packaged source code contains only 27% of the files under Eclipse's VC repository. Similarly, the Netbeans repository contains 53,360 Java files, and the Ubuntu's source code bundle contains only 14,330 (27% of the files).

If we start from the assumption that the files in the source archive are the ones truly required to build the binaries—at least under Linux—then what are the other 73% of the files in the version control system of these products? And perhaps equally important, can we gain any insight into either project and their development practises by inspecting the files that are in the repositories but are not used to build the IDEs.

Observation No. 1. The VC repository of a software product might contain a large proportion of code that is not required to build and run such product.

Neither Netbeans nor Eclipse have a simple directory organization of their VC repositories. Netbeans is composed of almost 1,000 different top-level directories, while Eclipse contains just under 500. A proper analysis of these top-level directories is beyond the scope of this paper. A quick review, however, sheds some light on their organization.

In Netbeans, of the 998 directories under VC, only 28% are present in its source code archive (280). Of these 998, 827 directories contain Java source code files (239 in the source archive). Even though Eclipse's VC contains more files than Netbeans, its VC repository is organized into fewer directories: 462 directories (248 containing java source files– 3.4 times less than Netbeans and Eclipse contains 1.28 times more Java files).

We were surprised to discovered that the source archive of Eclipse is organized in a very different way than its VC repository. The maintainers of the source code package have reorganized the modules of Eclipse into a more meaningful structure of only 4 directories, with only 2 containing Java source code: ecf-src, and plugins. Under these two directories we found 199 directories, corresponding to top level directories in the VC repository (43% of the top-level directories in the VC repository).

Observation No. 2. The logical structure of the files that compose a product might not be reflected in the way such files are organized in its VC repository.

Table 1 shows a comparison of the largest directories in Netbeans, and the number of files found in the archive from such directories. As it can be observed, the largest directories under VC are not present in the archive. The immediate questions to ask is *why?* and *What is in such directories?* A quick manual analysis of the largest directories, showed that they are optional plugins. Whether a plugin should be included (or not) in a runnable binary (such as it is done by Ubuntu) is a decision that packagers do.

Observation No. 3. There might exist several potential ways to package and build a software system, each requiring a different subset of the source code.

Such variations might be a response to the environment it will run (such as operating system support–Windows vs Linux) or selected optional features. In some instances the person creating such instance package is a member of the development team; in others it might not—such as in Linux distributions. Regardless, the corollary of observations 1 to 3 is that for a more accurate view of a software system and its relationship between the runnable code (its binary) and its source code (including its history) one needs to take into account both, the version control repository, and the bundles of source code used to build them.

Top-level directory	VC	Source Package
uml	2485	
performance	1964	
installer	753	
bpel.editors	619	
etl.editor	575	
compapp.projects.jbi	557	
java.source	526	467
form	515	503
java.editor	505	487
java.hints	484	327
jemmy		393

Table 1: Netbeans: Largest top-level directories (in number of Java files) in the VC repository and the number of the corresponding files in the source archive. The module jemmy was not present in the VC repository. The directories with files listed in the Package are the largest in it.

As previously mentioned, we found that the number of top-level directories in which Netbeans is broken down is remarkably large (998). The distribution of the number of files in the top directory of the VC repository of shows a median of 26 files (with a first quartile of 8, and a third of 72), We believe that a more in depth analysis of its architecture, and how its developers work on them might highlight interesting facts, such as if this is a way for them to minimize communication and a reflection of Conway's law; or whether other reasons are behind it. Nonetheless, we believe is it very hard for any developer to keep track of the contents of so many directories.

Table 2 compares the largest directories in Eclipse under its VC repository and in the source archive. The median of the number of files per top-level directory is 30 files (with a first quartile of 9, and a third of 100). Even though these numbers are close to those of Netbeans in the first two quartiles, they grow faster for the third and forth quartile.

Similar to Netbeans, the largest directories (in terms of source code files) under VC do not make it to the source archive. However, the reasons are remarkably different. The largest, e4, is where the development of core features of a version 4.0 of Eclipse are being developed (hence they are not part of the current distribution). The equinox-incubator is an umbrella under which new products are being incorporated into Eclipse (usually projects already developed by the community of Eclipse developers and users that others consider worth of including into Eclipse). One can speculate that the Eclipse Foundation seems to be more eager to experiment, and incorporate products from its community than Oracle, but this requires further research.

Some top-level directories include some products that can

Top-level directory	VC	Source Package
e4	9,264	
equinox-incubator	8,739	
org.eclipse.jdt.ui.tests.refactoring	7,004	
pde	5,074	916
org.eclipse.jdt.core.tests.model	4,500	
pde-incubator	2,214	
org.eclipse.jdt.ui	2,071	2,055
org.eclipse.swt	1,768	1,711
org.eclipse.ui.workbench	1,336	1,319
org.eclipse.jdt.core	1,182	1,190
org.eclipse.debug.ui	792	759

Table 2: `Eclipse`: Largest top-level directories (in number of Java files) in the VC repository and the number of the corresponding files in the source archive. As it can be seen, the largest directories are not in the package. The directories with files listed in the Package are the largest in it.

be considered infrastructure needed to run `Eclipse`. Perhaps the best known example is `swt` (the Standard Widget Toolkit). `swt` competes directly with Oracle's `awt` (Abstract Window Toolkit) and `swing` (the primary Java GUI widget toolkit)[3]. Oracle maintains a different repository for `awt` and `swing`; and hence, it does not make sense to directly compare `Eclipse`'s `swing` to any part of `Netbeans`. Nonetheless, the presence of `swing` under `Eclipse` provides interesting clues into its evolution; for example, the inadequacy of `awt` for `Eclipse` purposes, and the need of the `Eclipse` developers to boostrap their own widget toolkit to fill that gap—before `swing` was mature enough for production use. `swt` is clearly independent and can be packaged as an separate product (as is done by Ubuntu, under the name `swt-gtk`, where `gtk` is the X11 widget systems upon which `swt` is based upon). Other stand-alone products that are also stored in `Eclipse`'s VC repository are `aspecj` and `eclipse-emf` (`Eclipse`'s modeling framework) and `eclipse-rse` (Target Management to administer remote systems). The fact that the Eclipse IDE has given its name to its eponymous Foundation and its repository can create confusion among researcher. Eclipse's VC repository contains products that are not part of the IDE (such as `aspectj`) and infrastructure that one can debate is part (or not) of the IDE such as `swt`.[4]

Observation No. 4. A VC repository might contain several products that are not part of each other. Similarly, an organization might split the source code of one of its products into several VC repositories.

Of course, this does not mean that one cannot compare `Eclipse` (including `swt`) against `Netbeans` (without `swing`). The fact that `Eclipse` developers had to create a widget toolkit from scratch is telling of important aspects of its history (and the effort invested). The comparison, however, should take such differences into consideration to get a more accurate view on how each IDE is being created.

3.2 File Size

As we have already discussed, each VC system might not be directly comparable because each contains different products. Nonetheless, we can investigate the distribution of file sizes between both to evaluate if the file sizes vary significantly between the two projects.

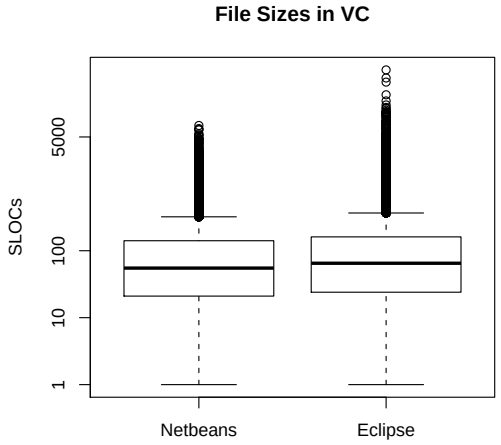

Figure 1: Distribution of file sizes in VC repos.

Figure 1 shows the distribution of the file sizes of both products. As it can be seen, distributions follow what appear to be a power-law distribution, consistent with research by [2]. However, as software developers, we believe the median number of SLOC per file is small (55 SLOC for `Netbeans`, and 64 SLOC for Eclipse)[5]

Plotting the density distribution of the sizes (in SLOC) shows that large files, even though few, contribute significantly to size of the project. For instance, figure 2 shows the density distribution of `Netbeans` code in its VC repository. Its repository contains 6.7 MSLOC (median 55 SLOC, average 125). However, the largest 5% of the files contribute 2.3 MSLOC (almost 1/3 of the source code), and files larger than 279 SLOC contribute 50% of the code.

`Eclipse` shows similar results. As shown in figure 2, its density curve is similar to `Netbeans`, although skewed to the right. The total SLOC is 10M, with a median of 64 SLOC and an average of 167. The largest 5% of its files contribute 4.2MSLOC, 42% of the code, 50% of the code in `Eclipse`'s VC can be attributed to files that are larger than 444 SLOC.

As it can be seen from figure 1, in both cases, `Netbeans` and `Eclipse`, 50% of the code in both systems comes from files that could be considered outliers, according to their size boxplots. This result is consistent with research that has been found that few files have the most modifications (see [3] for a summary of research in this area).

[3]Originally developed by Sun, they are now property of Oracle.

[4]This is perhaps similar to the confusion that occurs when researchers mention they have analyzed `Apache`. Do they refer to Apache's `httpd` server (the original Apache) or to the entire VC repository of its eponymous Foundation?

[5]Common Java constructs and traditions such as POJO's, Beans, Exceptions, and Interfaces could be contributing to a suprisingly small median file size.

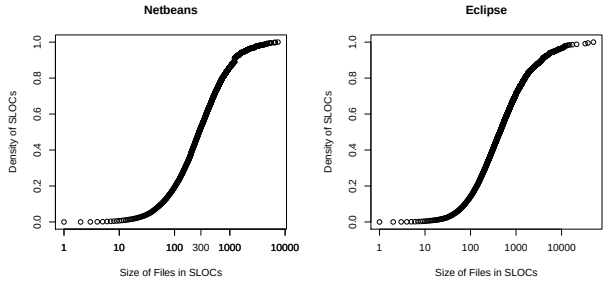

Figure 2: Density distribution of file sizes for `Netbeans` and `Eclipse`

> **Observation No. 5.** In software repositories mining, data points that appear as outliers might have a significant impact in any analysis and should not be ignored.

3.3 Cloning or Branching

Any discussion that involves size of source code eventually includes the question: how much of the code has been cloned? In previous work [4], we used clone detection to measure the proportion of copied tokens in a source file that are also present in other files. Files with a large proportion of copied tokens might be file siblings (a copy of another file) or files that tend to have generic code that needs to be repeated over and over again. Since unit tests tend to be quite similar to each other, we removed any file with a name that included *test* in its name.

In `Netbeans` we discover that, out of 42,375 different files, only 4,393 contain files with a proportion of copied tokens of at least 75%, and 2,829 have 100% ratio. Of these 2,829, 109 were automatically generated, and many others have identical filenames (though in different subdirectories). 1,360 of them were identical byte-per-byte copies of each other. These files account for .4MSLOCs), less than 7% of the SLOC total. `Eclipse`, on the hand, shows significantly larger copying. 10,558 files exhibit a copied ratio of 75% or more (1/6 of the files), and 8,239 had a 100% ratio. These files contributed 2.1 MSLOCs, 21% of the total code based in the VC repository.

To supplement this analysis we compare the fully qualified name (FQN) of each Java file. Our assumption, as well as that of Java's built-in classloader, is that if two classes share the same FQN, then they represent the same concept within the software (though with potentially different logic). If we divide the source code of `Eclipse` into three subsets: *e4* (work in progress towards the next major release of eclipse), *incubator* (speculative side projects under development that may or may not succeed in eventually joining the main product), and *eclipse* (everything else), we find there are 39,602 different FQNs in *eclipse*, 8,701 in *incubator*, and 8,851 in *e4*. Figure 3 shows a proportional Venn diagram that represents the overlap between the FQNs of these sets. As it can be seen, *incubator* and *e4* appear to be responsible for many duplications of code. This makes sense: both areas are experimental, and sources inside each may one day replace the current code in `Eclipse`.

This also highlights an important fact: the developers of `Eclipse` decided to create a copy of part of `Eclipse` and place it in another directory within the VC repository to continue its development instead of creating a branch (using the features of the VC system) for such purpose. It is likely that this decision has advantages and disadvantages that need to be further investigated.

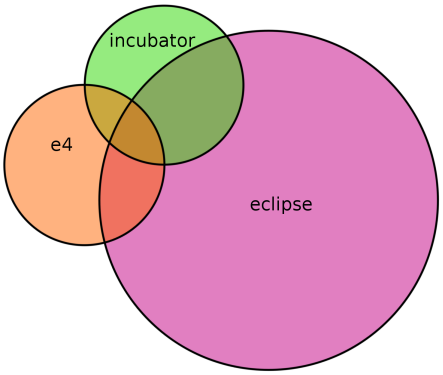

Figure 3: Distinct FQNS among 3 directories of `Eclipse`'s VCS.

> **Observation No. 6.** Copying of files within a repository can be significant, and should be evaluated to avoid bias.

4. CONCLUSION AND FURTHER WORK

We have demonstrated that a comparison between the source code of two relatively equivalent systems requires careful consideration. Many potential pitfalls exist, such as identifying the proper code of such systems from their version control systems, and the impact of cloning. Otherwise, results might be biased.

We believe that research is needed in this area. The observations made in this paper should be evaluated in various projects, to determine if they are generalizable beyond these two projects.

5. REFERENCES

[1] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The Promises and Perils of Mining Git," in *Proceedings of the International Working Conference in Mining Software Repositories*, 2009, pp. 1–10.

[2] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a large object-oriented software system," *IEEE Transactions on Software Engineering*, vol. 33, pp. 687–708, 2007.

[3] H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *J. Softw. Maint. Evol.*, vol. 19, pp. 77–131, March 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1345056.1345057

[4] D. M. Germán, M. Di Penta, Y.-G. Guéhéneuc, and G. Antoniol, "Code siblings: Technical and legal implications of copying code between applications," in *MSR*, M. W. Godfrey and J. Whitehead, Eds. IEE, 2009, pp. 81–90.