VICTORIA UNIVERSITY OF WELLINGTON

Te Whare Wananga o te Upoko o te Ika a Maui

School of Engineering and Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Milestone 1 Report

Joshua Lindsay

Supervisor: Professor James Noble

Submitted in partial fulfilment of the requirements for the Bachelor of Information Technology fourth year research project.

Abstract
This document contains a report on my progress of milestone 1.

Table of Contents

1 Introduction / Background

Much of software engineering is focused on how software could or should be written, and how it 'should' be structured. A consequence of much of this offered advice is the *Lego Hypothesis*; which says that software can be put together like Lego out of lots of small interchangeable components [1].

If this were the case then we would expect the size of the classes to stay the same, no matter how large the program – just as fixed sized Lego bricks can form buildings of any size [2].

So any large program can (according to this theory) consist entirely of a large number of small classes, which should be a natural consequence of good object-oriented development [9].

Yet these "big" Java classes still exist.

And certain "big" Java classes even have a name – "God classes" [5]

Which makes us wonder why? Why is it when the design of classes is consistently declared to be central to the OO paradigm [4] do they still exist in the real world? Is it because programmers have missed the OO paradigm boat? Or are there simply situations where the creation of a "big" class is inevitable?

These are the questions I hope to answer during my research, which will hopefully give the Java community (and perhaps the OO community in general) some idea as to what can be done about them.

1.1 Research goal

Just to be crystal clear (and if you didn't read the last section), the goal of my research is to:

Identify *why* big Java classes exist, and *what* (if anything) can be done about them?

1.2 This milestones goal

My goal for this milestone was to:

Attempt to identify what makes a class "big"

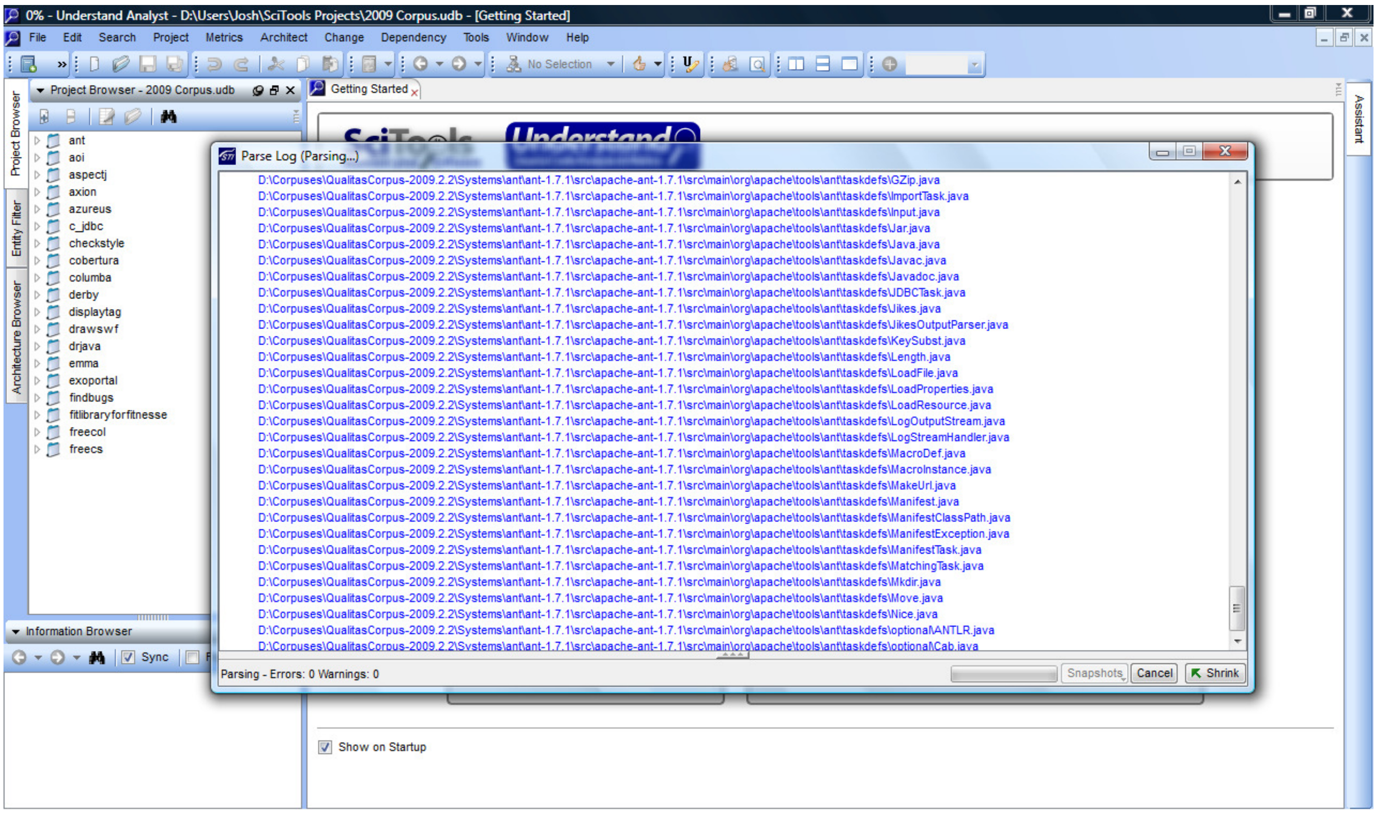
Which can be broken down further into steps:

- Identifying a range of size metrics to use for analysing a large corpus of non-trivial Java programs.
- Running statistical tests on the corpus to measure those metrics.
- And finally; use the results of the tests to identify and define what makes a Java class "big"

2 What has been achieved

2.1 Selecting a software metrics tool

Rather than writing my own software for calculating software metrics, I did some research into possible existing software that can calculate the metrics I am interested in. Two stood out; Semmle's product *SemmleCode* and SciTools *Understand*. For this milestone, *Understand* was the best choice as it is able to analyse entire directories of programs and offers a large selection of software metrics that can (when run on the selected directories containing java source files) export its results to .csv files.



```
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\GZip.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\ImportTask.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Input.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Jar.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Java.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Javac.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Javadoc.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\JDBCTask.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Jikes.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\JikesOutputParser.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\KeySubst.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Length.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\LoadFile.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\LoadProperties.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\LoadResource.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\LogOutputStream.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\LogStreamHandler.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\MacroDef.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\MacroInstance.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\MakeUrl.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Manifest.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\ManifestClassPath.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\ManifestException.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\ManifestTask.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\MatchingTask.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Mkdir.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Move.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\Nice.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\optional\ANTLR.java
D:\Corpuses\QualitasCorpus-2009.2.2\Systems\ant\ant-1.7.1\src\apache-ant-1.7.1\src\main\org\apache\tools\ant\taskdefs\optional\Cab.java
```

```
Parsing - Errors: 0 Warnings: 0
```

Fig. 1: Screenshot of Understand 2.0 (build 477) analysing the corpus.

2.2 Metrics

For the first part of this milestone I needed to select several software metrics that give an indication of the size of a class. *Understand 2.0* contained these metrics and many more, but for this step of the project I focused on a select few.

CountDeclMethod
: The number of local (not inherited) methods (includes both static and instance methods).

CountDeclInstanceVariable
: The number of instance fields in a class.

CountDeclClassVariable
: The number of static fields in a class.

CountLine
: The number of lines in a class.

CountLineCode
: The number of lines of source code. A useful subset of CountLine. This is any line that isn't blank, or isn't a pure comment (i.e. non-trailing comment).

CountLineCodeExe
: The number of executable lines of code. A useful subset of CountLineCode.

2.3 Calibration

Having chosen *SciTools Understand 2.0* as my tool and selected the metrics required, I needed to confirm the accuracy of *Understand* for these metrics. For this I wrote a number of test classes (in Java of course) and manually worked out what the result for each metrics test should be.
I then ran *Understand* over these test classes, and confirmed that the results do indeed match.

```
public class Class0 {
}
```

```
public class Class1 extends AbstractClass {
	int arg;

	public Class1(){
	}

	// comment
	public Class1(int arg){
		this.arg = arg;
	}

	public int foo(){
		return arg + p;
	}
}
```

```
public class Class2 {
	public static void main(String[] args) {
		System.out.println(new Class1(10).foo());
		System.out.println(Class4.hello_world);
		Class5.getInstance();
	}
}
```

```java
// Some random example
public class Class3 {
	private Class4 class4;

	public Class3(){
		class4 = new Class4();
	}

	/**
	 * Returns some sudo-random pre-established number
added to i.
	 * @return some pre-established random number + i
	 */
	public int doSomething(int i){
		return class4.compute(i);
	}

	class Class4{
		private int foo = ((int)(Math.random() + 1 *
100000));

		private int compute(int bar){
			return foo + bar;
		}
	}
}
```

```java
public class Class4 {
	public static String hello_world = "Hello World.";
	public static int one = 1;
}
```

```java
public class Class5 {
	private static Class5 instance;
	private String value = "blah";

	private Class5(){}

	public static Class5 getInstance(){
		return (instance == null ? (instance = new
Class5()) : instance);
	}

	public String getValue(){
		return value;
	}
}
```

```java
public abstract class AbstractClass {
	protected int p = 100000;
	public AbstractClass(){
}
```

2.4 The corpus

[10] The corpus is maintained by Ewan Tempero at the University of Auckland. It is a large collection of open-source Java software systems, often containing many versions of each system.
The corpus version I used was *20090202* but the programs outlined below could also be acquired from their homepages as well.

The following (19) programs were selected from the Qualitas Corpus:

Application Name	Version	Homepage URL
ant	1.7.1	http://ant.apache.org/
aoi	2.5.1	http://www.artofillusion.org/
aspectj	1.0.6	http://www.eclipse.org/aspectj/
axion	1.0-M2	http://axion.tigris.org/
azureus	3.1.1.0	http://azureus.sourceforge.net/
c_jdbc	2.0.2	http://c-jdbc.ow2.org/
checkstyle	4.3	http://checkstyle.sourceforge.net/
cobertura	1.9	http://cobertura.sourceforge.net/
columba	1.0	http://www.columbamail.org/
derby	10.1.1.0	http://db.apache.org/derby/
displaytag	1.1	http://displaytag.sourceforge.net/
drawswf	1.2.9	http://drawswf.sourceforge.net/
drjava	20050814	http://www.drjava.org/
emma	2.0.5312	http://emma.sourceforge.net/
exoportal	1.0.2	http://exo.sourceforge.net/
findbugs	1.0.0	http://findbugs.sourceforge.net/
fitlibraryforfitnesse	20050923	http://fitnesse.org/
freecol	0.5.1	http://www.freecol.org/
freecs	1.2.20060130	http://freecs.sourceforge.net/

Table 2: Applications included in the metrics

Why 19 systems you may ask? Simply it is because all of these programs contained 'src' in their archives' file names. Since SciTools cannot read byte code this was desired. However the included systems represent a good range of differences in application and system size.

2.5 Gathering results from the corpus

Running SciTools over the selected systems was easy. Once SciTools had finished gathering its results, these could then be exported to a .csv (comma separated values) file which could then be opened with excel.

2.5.1 Initial Results

	CountDeclClassVariable	CountDeclInstanceVariable	CountDeclMethod	CountLine	CountLineCode
Total	20239	41559	134945	2552886	1704649
Min	0	0	0	0	0
Max	315	184	1143	23140	20004
Average	1.025	2.106	6.840	129.404	86.407

Table 1: Summary of results

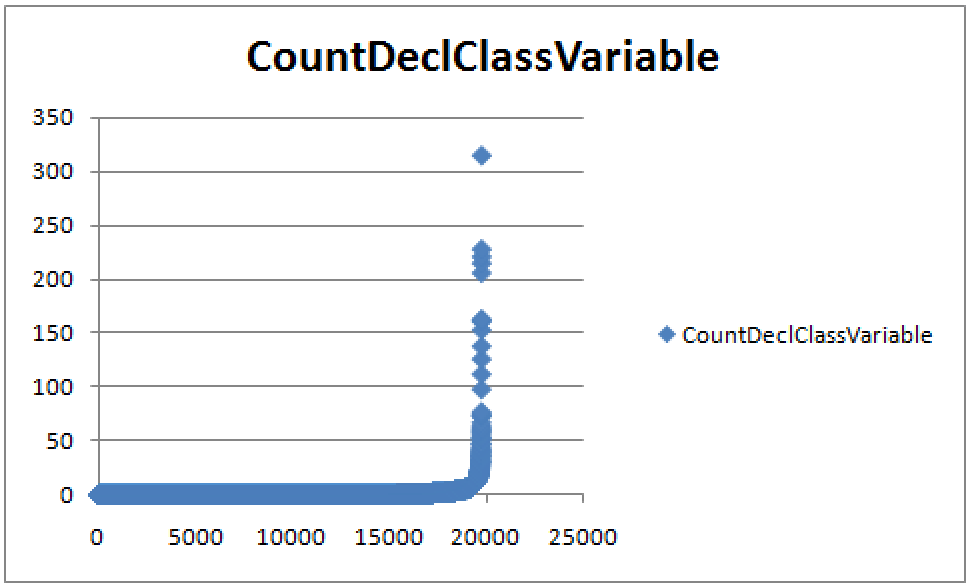


Figure 1: Number of static fields in each class

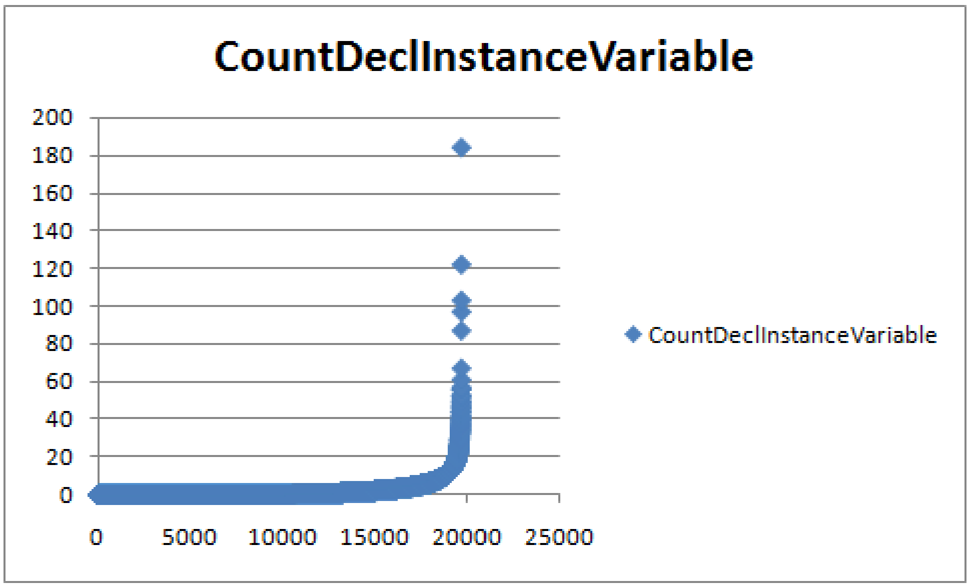


Figure 2: Number of non-static (Instance) fields in each class

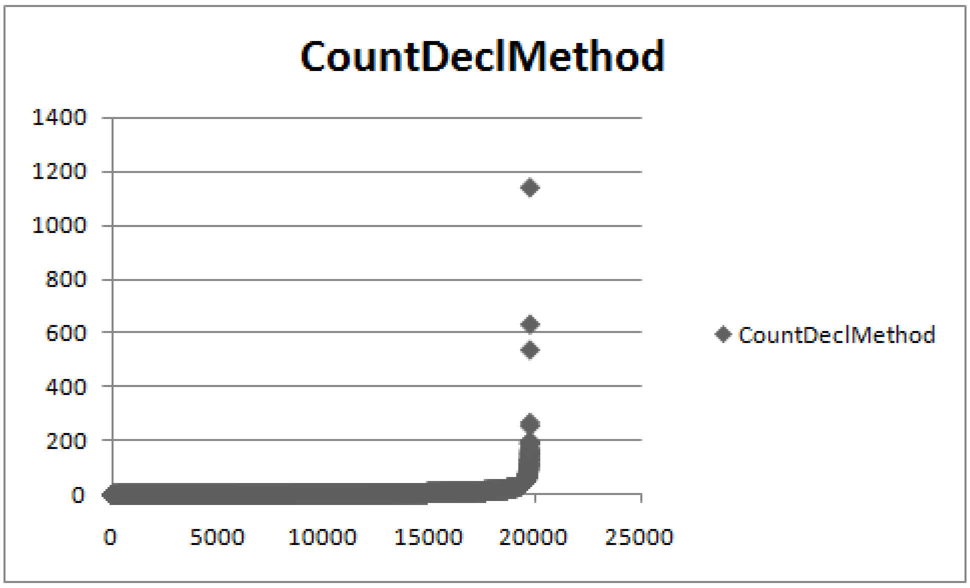


Figure 3: Number of non-inherited methods in each class

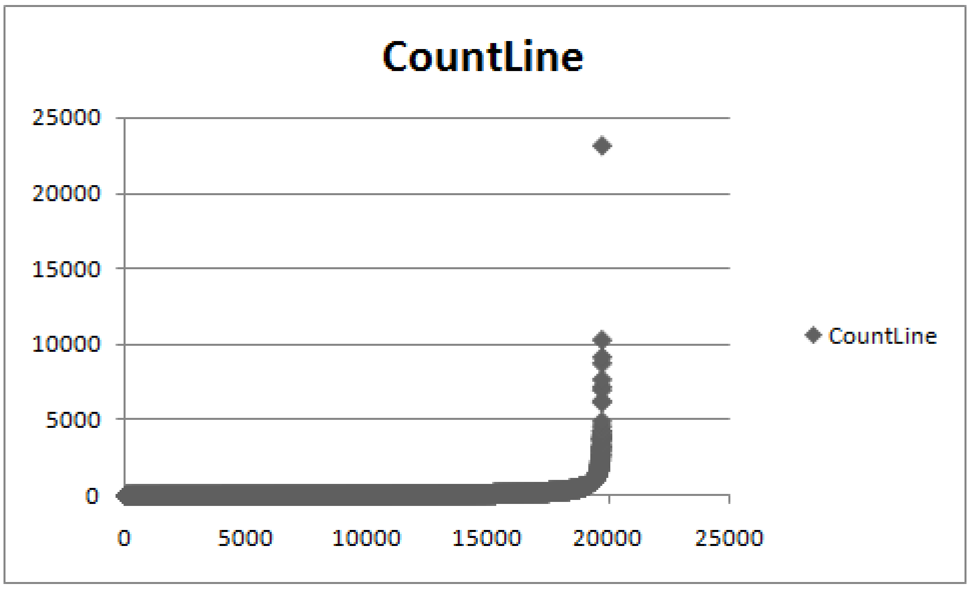


Figure 4: Number of lines in each class

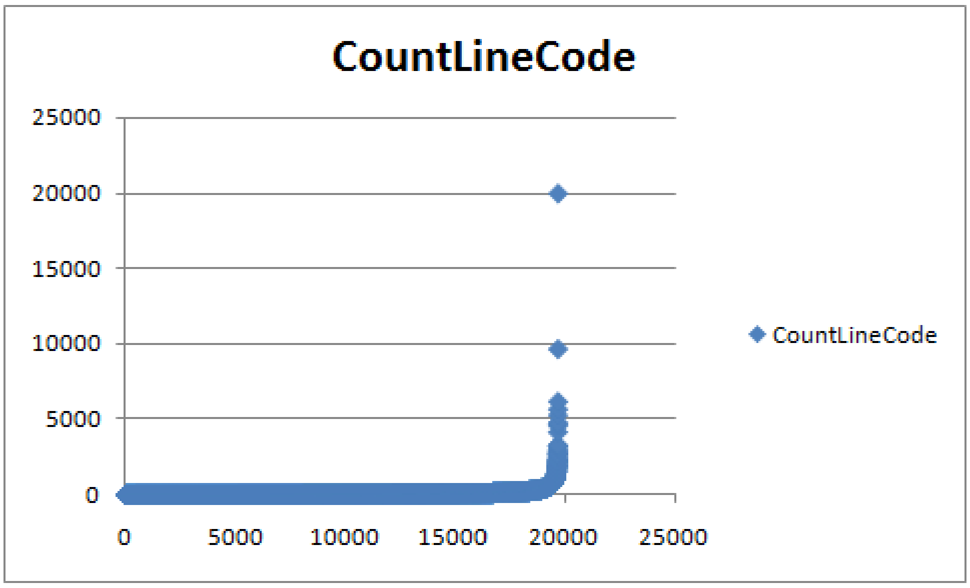


Figure 5: Number of lines of code in each class

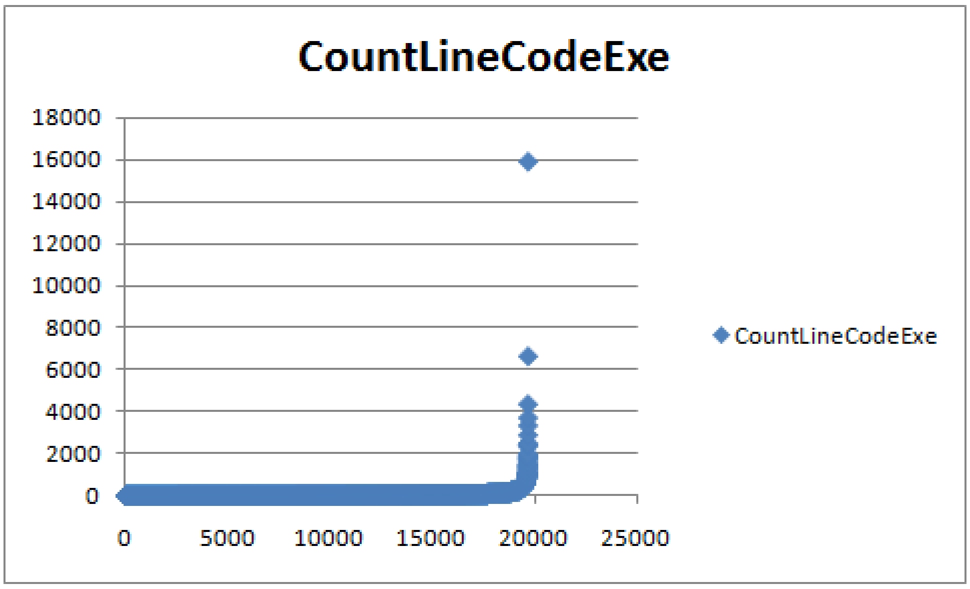


Figure 6: Number of executable lines of code in each class

2.5.2 Initial Analysis of results

All graphs show an interesting sudden jump in their metrics. It was not the gradual or steady curve I was expecting to see. Further analysis is needed but initial thoughts lean towards a) the data is very noisy / I have done something wrong or b) this is exactly the result I was expecting – lots of small classes and a band of large to very large ones.

As a side note; comparing what I have found to what Zhang and Tan found in their study [9], my results showed an average of 129 lines of code vs their 114 LOC, suggesting my selected systems had a nasty habit of containing larger classes. This could be for a number of reasons (e.g. I had fewer systems with smaller classes) and further research into this is required.

3 What's next

- Further analysis of results is needed.
- Identify and define what makes a Java class "big"
- Identifying similarities and patterns between "big" Java classes in the corpus. This will help us understand *why* "big" classes exist.
    - This may involve running additional statistical tests to identify if any "micro patterns" exist in the "big" Java classes.
- Having understood the *why*, identifying *what* can be done (if anything) with these "big" classes.

4 References

[1] G. Baxter, M. Frean, H. Melton, J. Nobel, M. Rickerby, H. Smith, E. Tempero, M. Visser. Understanding the Shape of Java Software. ACM, 2006.

[2] R. Biddle, M. Frean, A. Potanin, J. Noble. Scale-free Geometry in Object-Oriented Programs.

[3] S. R. Chidamber, C. F. Kemerer. Towards a Metrics Suite for Object Oriented Design. ACM, 1991.

[4] S. R. Chidamber, C. F. Kemerer. A Metrics Suite for Object Oriented Design. IEEE, 1994.

[5] S. Ducasse, T. Girba, R. Marinescu, D. Ratiu. Evolution Enriched Detection of God Classes. CAVIS 2004.

[6] N. E. Fenton, S. L. Pfleeger. Software Metrics, A Rigorous & Practical Approach - Second Edition Revised Printing. PWS Publishing Company, 1997.

[7] J. Y. Gil, I. Maman. Micro Patterns in Java Code. ACM, 2005.

[8] H. Melton, J. Nobel, E. Tempero. How do Java Programs use Inheritance? An Empirical Study of Inheritance in Java Software. Springer-Verlag Berlin Heidelberg, 2008.

[9] H. B. K. Tan, H. Zhang. An Empirical Study of Class Sizes for Large Java Systems. IEEE, 2007.

[10] Qualitas Research Group, Qualitas Corpus Version 20090202, http://www.cs.auckland.ac.nz/~ewan/corpus. The University of Auckland, February, 2009.