Examine The Evolution Of Metrics By Mining Repositories

Project Type 2

Xunrong Xia
Computer Science and Software Engineering Department
Concordia University
Montreal, QC, Canada
xunrongxia0722@gmail.com

Yifei Zhang
Computer Science and Software Engineering Department
Concordia University
Montreal, QC, Canada
yifeizhang1016@gmail.com

Hongrui Guan
Computer Science and Software Engineering Department
Concordia University
Montreal, QC, Canada
guan.hongrui@gmail.com

Siddharth Ravalia
Computer Science and Software Engineering Department
Concordia University
Montreal, QC, Canada
siddharth.ravalia@gmail.com

***Abstract*— Examine thing evolution of metrics by mining repositories is one type of case studies in software measurement. In this paper, we talk about this kind of study by using design flaw detection strategies to assess ten versions of an open source project, Jfree Chart. At first, we introduce the meaning of this study and related works for doing this study. After that, we discuss some metrics used in detection strategies and how to implement those metrics. Then, we give the detail of this empirical study including data and analysis. At the end of this paper, several conclusions are made base on the analysis.**

Keywords— metrics; detection strategies, Jfree Chart

INTRODUCTION

Maintenance effort is reported to be more than half of the overall development effort and most of the maintenance effort is spent in adapting and introducing new requirements, rather than in repairing errors [1]. To detect the evolution of metrics by mining repositories, we can assess whether the quality of a project improves or deteriorates in time. In this project, we use design flaw detection strategies to find the design changes that affected the project. In section 2, we discuss related works, and find the formal definitions of two design flaw detection strategies, god class and data class, which will be used in the following research. Section 3 defines 7 metrics that used for the design flaw detection strategies. They are Weighted Method Count (WMC), Tight Class Cohesion (TCC), Access to Foreign Data (ATFD), Weight of a Class (WOC), Number of Methods (NOM), Number of Public Attributes (NOPA), Number of Accessor Methods (NOAM).We then apply our approach in a free Java chart library, which is called JfreeChart in section 4. In this section, we also present and discuss in detail the results we obtained on the latter case study. Section 6 summarizes the contribution of the paper and points to the conclusion and the main issues that we intend to address in the future.

RELATED WORK

Detection Strategy

The concept of detection strategy is raised by Radu Marinescu in his paper "Detection Strategies: Metrics-Based Rules for Detecting Design Flaws". A detection strategy is the quantifiable expression of a rule by which design fragments that are conforming to that rule can be detected in the source code. A detection strategy is therefore a generic mechanism for analyzing a source code model using metrics. Note that in the context of the previous definition, "quantifiable expression of a rule" means that the rule must be properly expressible using object-oriented design metrics [2].

The use of metrics in the detection strategies has two main parts. The one is filtering mechanism and the other is composition mechanism. The purpose of filtering mechanism is to reduce the initial data set, so that only those values are retained that present a special characteristic. There are two types of filters: Marginal Filter and Interval Filter. A detection strategy should be able to quantify entire design rules. As result, we need the composition mechanism to support a correlated interpretation of multiple result sets. This mechanism is depend on a set of logical operators that "glue" together different metrics in a user-defined rule.

God class and Data class

The concept of God Class is also raised by Radu Marinescu in his Ph.D. thesis "Measurement and Quality in Object-Oriented Design". An instance of a god-class preforms most of the work, delegating only minor details to a set of trivial classes and using the data from other classes.

The concept of Data Class is given by D.Ratiu, S.Ducasse,T.Girba, and R.Marinescu in their paper "Using History Information to Improve Design Flaws Detection". Data Classes are classes that have fields, getting and setting methods for the fields, and nothing else. Such classes are dumb data holders and are almost certainly being manipulated in far too much detail by other classes.

METRICS

Metrics Definition

In this project, we use detect strategies to evaluate two kinds of flaws, God Class and Data Class. God Class refers to those classes that tend to centralize the intelligence of the system. An instance of a god-class performs most of the work, delegating only minor details to a set of trivial classes and using the data from other classes. Data Classes are classes that have fields, getting and setting methods for the fields, and nothing else. Such classes are dumb data holders and are almost certainly being manipulated in far too much detail by other classes. For God Class, we use a strategy come from Radu Marinescu. This strategy includes three measurements, WMC, TCC and ATFD.[2]

WMC: Weighted Method Count (WMC) is the sum of the statistical complexity of all methods in a class. We considered the McCabe's cyclomatic complexity as a complexity.

Given a single method or function:

Cyclomatic Complexity = D + 1

, where D = is the number of control predicate (or decision) statements.

In this definition, we simply have to count the number of if, switch cases, for, while, do-while statements

TCC: Tight Class Cohesion (TCC) is the relative number of directly connected methods. We determine two methods are directly connected base on these two measures:

First, two methods are directly connected if they are directly connected to an attribute. Second, a method m is directly connected to an attribute, when the attribute appears within the method's body or within the body of a method that is directly or transitively invoked by method m.

NP(C) is the maximum possible number of direct or indirect connections in a class C. If there are N methods in a class C, NP(C) is N * (N – 1)/2.

Let NDC(C) be the number of direct connections.

TCC(C) = NDC(C)/NP(C)

ATFD: Access to Foreign Data (ATFD) represents the number of external classes from which a given class accesses attributes, directly or via accessor-methods.

The higher the ATFD value for a class, the higher the probability that the class is or is about to become a god-class

The final form of the detection strategy for God-Classes is formally described in Equation as below,

$$GodClass(S) = S' \left| \begin{array}{c} S' \subseteq S,\ \forall C \in S' \\ (WMC(C), TopValues(25\%)) \wedge (ATFD(C), HigherThan(1)) \wedge (TCC, BottomValues(25\%)) \end{array} \right.$$

For Data Class, the strategy we use includes five measurements, WOC, NOM, WMC, NOPA, NOAM.[1]

WOC: Weight of a Class (WOC) is the number of non-accessor methods in the interface of the class divided by the total number of interface members.

NOM: Number of Methods (NOM)

NOPA: Number of Public Attributes (NOPA) is defined as the number of non-inherited attributes that belong to the interface of a class

NOAM: Number of Accessor Methods (NOAM) is defined as the number of the non-inherited accessor-methods declared in the interface of a class

The Data Class detection strategy can be described in one equation as below,

$$DataClass(S) = S' \left| \begin{array}{c} S' \subseteq S \\ \forall C \in S' \\ \left(\left(\frac{WMC(C)}{NOM(C)} < 1.1\right) \wedge (WOC(C) < 0.5)\right) \wedge ((NOPA(C) > 4) \vee (NOAM(C) > 4)) \end{array} \right.$$

B Metrics Implementation details

In this project, we implement those metrics by using eclipse JDT. The Eclipse Java Development Tools (JDT) provide APIs to access and manipulate Java source code. It allows to access the existing projects in the workspace, create new projects and modify and read existing projects. It also allows to launch Java programs. JDT allows to access Java source code via the

Abstract Syntax Tree (AST), which is a Document Object Model similar to the XML DOM.

We using AST framework to get the information of one project's source code, like how many lines of code in a class or the number of methods in the class. The most difficult point is not writing code but using APIs from AST framework appropriately.

For example, to implement WMC metric, we should analyze the structure of the code in methods because we should know the count of decision points. Fortunately, the AST framework has the API to get the number of decision points, like if, for, while and switch. So we can directly use them to get the number of them. However, to know the functions of the framework is the difficult part.

```
List<CompositeStatementObject> ifStatementObjects = compositeStatementObject
        .getIfStatements();
List<CompositeStatementObject> switchStatementObjects = compositeStatementOb
        .getSwitchStatements();
List<CompositeStatementObject> forStatementObjects = compositeStatementObjec
        .getForStatements();
List<CompositeStatementObject> whileStatementObjects = compositeStatementObj
        .getWhileStatements();
List<CompositeStatementObject> doStatementObjects = compositeStatementObject
        .getDoStatements();

int count = ifStatementObjects.size() + switchStatementObjects.size()
        + forStatementObjects.size() + whileStatementObjects.size()
        + doStatementObjects.size();
```

FIG 1. Source code of counting decision points

For implementation of WOC(Weight of a Class), which is the number of non-accesseor methods divided by the addition between accessor methods and public attributes of the class, we add 1 for both the numerator and the denominator to avoid the class that has no interface members.

How to output the results of those metrics is also an important problem. If we just choose to use the console, it will be so hard for us to analyze. So, we choose excel to store the results of metrics. In this project, we use Java Excel API framework to generate excel file automatically to save all metrics values for each class. We implement 2 classes. Metricinfo, and classinfo to save all metrics values and the detected class's information. All value of metricinfo will be transformed into classinfo. Classinfo would be used in the generation phase and each classinfo object contains the data of one row. So, it is easy for us to generate excel files.

```
public class MetricInfo {
    private String metricName;
    private Map<String, Double> metricValue;

    public MetricInfo(String metricName, Map<String, Double> metricValue) {
        super();
        this.metricName = metricName;
        this.metricValue = metricValue;
    }

    public String getMetricName() {
        return metricName;
    }

    public Map<String, Double> getMetricValue() {
        return metricValue;
    }

}
```

FIG. 2: Source code of Metricinfo class

```
public class ClassInfo {
    private String className;
    private Map<String, Double> metricValue;

    public ClassInfo(String className, Map<String, Double> metricValue) {
        super();
        this.className = className;
        this.metricValue = metricValue;
    }
```

FIG. 3: Source code of Classinfo class

```
public void generateExcel() throws Exception {
    labels = new ArrayList<String>();
    labels.add("Class Name");
    for (String metricName : classInfos.get(0).getMetricValue().keySet()) {
        labels.add(metricName);
    }

    // ///create file
    File file = new File("C:/temp/" + fileName + ".xls");
    WorkbookSettings wbSettings = new WorkbookSettings();

    wbSettings.setLocale(new Locale("en", "EN"));
    WritableWorkbook workbook = null;

    workbook = Workbook.createWorkbook(file, wbSettings);

    workbook.createSheet("Report", 0);
    WritableSheet excelSheet = workbook.getSheet(0);

    // create headers
    createLabel(excelSheet);
    // create the content
    createContent(excelSheet);

    workbook.write();
    workbook.close();

}
```

FIG. 4: Source code of process to create an excel file

After we got the excel file, we write some excel formulas to check whether a class is a godclass or dataclass. For example, if one class is a God Class, its value in the column of godclass will be 1.

	Class Name	NOPA	ATFD	NOAM	NOM	WOC	WMC	TCC		GODCLASS	DATACLASS
6	org.jfree.chart.renderer.xy.CyclicXYItemRenderer	0	4	0	1	1	32	-1		1	0
149	org.jfree.chart.ChartFactory	0	42	0	34	35	126	0		1	0
181	org.jfree.data.general.DatasetUtilities	0	25	0	40	41	199	0.01667		1	0
182	org.jfree.chart.ChartUtilities	0	10	0	26	27	43	0.04308		1	0
183	org.jfree.chart.plot.XYPlot	5	56	35	182	1	539	0.04742		1	0
184	org.jfree.chart.plot.CategoryPlot	8	49	27	154	1	420	0.04745		1	0
185	org.jfree.chart.plot.ContourPlot	0	37	27	77	0	164	0.04785		1	0
187	org.jfree.chart.plot.PiePlot	10	44	41	119	0	249	0.06523		1	0
190	org.jfree.chart.renderer.category.AbstractCategoryItemRenderer	0	39	10	51	1	146	0.06745		1	0
193	org.jfree.chart.renderer.xy.AbstractXYItemRenderer	0	41	12	53	1	163	0.0762		1	0
194	org.jfree.chart.axis.ValueAxis	10	22	19	67	1	154	0.08051		1	0
195	org.jfree.chart.renderer.xy.XYLineAndShapeRenderer	0	29	10	50	1	112	0.08735		1	0
196	org.jfree.data.statistics.Statistics	0	7	0	13	14	55	0.08974		1	0
198	org.jfree.data.xy.DefaultHighLowDataset	0	5	0	18	2	33	0.0915		1	0
200	org.jfree.chart.axis.NumberAxis	4	24	6	36	1	100	0.09365		1	0
201	org.jfree.chart.renderer.category.BarRenderer	2	28	11	32	0	103	0.09476		1	0
202	org.jfree.chart.renderer.AbstractRenderer	8	20	34	169	1	351	0.0958		1	0
203	org.jfree.chart.axis.CategoryAxis	2	36	7	45	1	120	0.09899		1	0
207	org.jfree.chart.LegendItem	0	9	25	30	0	52	0.10115		1	0

FIG. 5: A sample of excel file

In this project, we use detection strategies to evaluate two kinds of flaws, God Class and Data Class. This part has two sections, the first section will talk about the details of detection strategies and the second section will give the details of God Class and Data Class respectively.

I. EMPIRICAL STUDY

Reasearch Question

According to the results of the above metric data, and examining behavior of the God classes and Data Classes, we have formulated some research questions. These will help us to follow the analysis. They are:

- What are the trends of the number of God Classes and Data Classes during the development of the project?
- Is the improvement or deterioration of the project related to the number of God Classes and Data Classes?
- Why did God Classes and Data Classes appear?
- Do all the God Classes and Data Classes be fixed?

Experiment design

Project - JfreeChart

JFreeChart is an open-source framework for the programming language Java, which allows the creation of a wide variety of both interactive and non-interactive charts.

JFreeChart supports a number of various charts, including combined charts:

- X-Y charts (line, spline and scatter). Time axis is possible.
- Pie charts
- Gantt charts
- Bar charts
- Scatter plots
- Combined plots and more

JFreeChart also works with GNU Classpath, a free software implementation of the standard class library for the Java programming language[5].JFreeChart can be used in applications, applets, servlets and JSP. This project is maintained by David Gilbert.

Version	Modified	Size
1.0.5	2007-03-23	5.94MB
1.0.6	2007-06-15	6.14MB
1.0.7	2007-11-14	6.46MB
1.0.8	2007-11-23	6.48MB
1.0.9	2008-01-04	6.50MB
1.0.10	2008-06-09	6.60MB
1.0.11	2008-09-19	6.83MB
1.0.12	2008-12-31	6.89MB
1.0.13	2009-04-30	7.19MB
1.0.14	2011-11-20	7.39MB

Fig 6: JFreeChrt Versions

Data collection

We have collected data for the metrics NOPA, AFTD, NOAM, WOC, WMC and TCC for every classes of all the versions used for this project. One of the version's data is shown below.

Using the plug-in to detect the 10 versions of the project and generate the metrics information excel.

Then we use excel functions to get god classes and data classes. This table shows our result, the first column is the version number and the second one is the total number of god class and the following columns are the total number of data class and total number of class as the header of the table shows.

Class Name	NOPA	ATFD	NOAM	NOM	WOC	WMC	TCC
org.jfree.data.DataUtilities	0	7	0	9	10	40	0
org.jfree.chart.renderer.RendererUtilities	0	3	0	3	4	34	0
org.jfree.data.general.DatasetUtilities	0	33	0	57	58	326	0.0188
org.jfree.chart.plot.CategoryPlot	8	66	39	215	1	560	0.0326
org.jfree.chart.plot.XYPlot	5	64	44	224	1	653	0.03652
org.jfree.chart.ChartUtilities	0	12	0	27	28	44	0.03989
org.jfree.chart.plot.ContourPlot	0	37	27	77	0	164	0.04785
org.jfree.chart.plot.PiePlot	10	54	49	143	0	317	0.05801
org.jfree.chart.renderer.category.AbstractCategoryItemRenderer	0	44	10	56	1	175	0.06039
org.jfree.chart.plot.PolarPlot	4	40	15	92	1	221	0.06163
org.jfree.data.statistics.Regression	0	5	0	6	7	39	0.06667
org.jfree.chart.renderer.category.BarRenderer	2	33	16	48	0	132	0.06826
org.jfree.chart.renderer.xy.AbstractXYItemRenderer	0	44	10	58	1	181	0.07018
org.jfree.chart.axis.ValueAxis	10	23	20	71	1	160	0.07606
org.jfree.chart.plot.ThermometerPlot	11	31	19	75	0	159	0.08541
org.jfree.chart.renderer.xy.XYLineAndShapeRenderer	0	28	10	50	1	111	0.08735
org.jfree.data.statistics.Statistics	0	7	0	13	14	55	0.08974
org.jfree.data.xy.DefaultHighLowDataset	0	5	0	18	2	33	0.0915
org.jfree.chart.axis.CategoryAxis	2	40	7	49	2	138	0.09269
org.jfree.chart.axis.NumberAxis	4	25	6	36	1	109	0.09365
org.jfree.chart.LegendItem	0	11	35	46	0	77	0.09372
org.jfree.chart.renderer.AbstractRenderer	8	21	41	192	1	393	0.09479
org.jfree.chart.renderer.xy.XYBarRenderer	0	33	13	42	1	126	0.0964
org.jfree.chart.renderer.category.LineAndShapeRenderer	0	20	11	42	1	92	0.0964

FIG : 7 Value Of Different Metrics for different Classes.

Statistical analysis

GOD CLASS

Here we represent our calculation of Total number of God Class in all versions of the project and how it evaluates from first version to last.

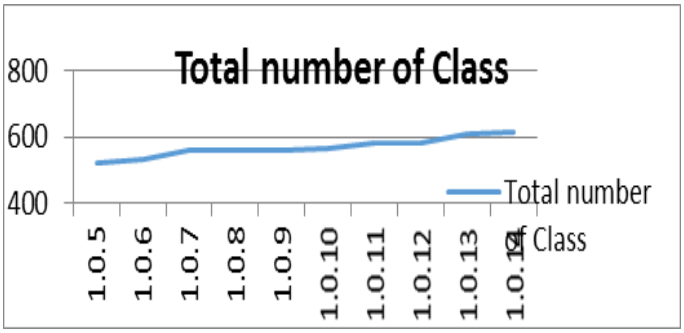

FIG 8. Evaluation Of Total Number Of God Classes

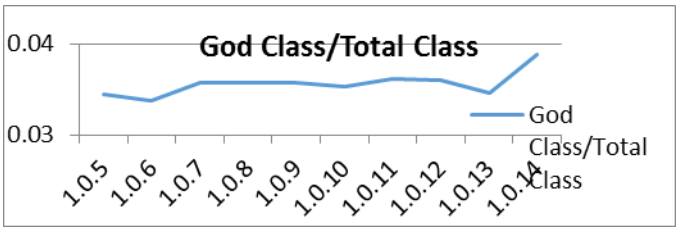

FIG 9. Evaluation Of Number Of God to Total Classes

The above graph represents number of God Classes in the project to the total number of classes in the project.

From these two graphs, we could see they have the similar increased trends in all versions. In the first line chart, we could see that the number of class was increased during the development. In order to get the change trend of god class, we use the number of god class divided the total number of class to get the second graph. This graph indicates there is a big increase occurs between version 6 to version 7 and version 12 to 13. By reading their ChangeLogs, we found there were some design changes in version 1.0.7 and 1.0.13, like add or remove methods or classes, which affected the number of god classes in those versions.

Specifically, we classified god classes into 4 types.

Type 1 – They are the god classes which exists in all 10 versions but harmless. Some classes were designed to implement complex but independent function. They are god class, but harmless so these kind of class is not flaws. For example, the PiePlot class, it aims to draw pie graph.

Example : *org.jfree.chart.plot.PiePlot*

Existing in all 10 versions but harmless

Example : org.jfree.data.xy.DefaultHighLowDataset

Just to build a data structure, do not need to decompose

Type 2 – They are the god class which exists in all versions and we consider them harmful. For example, the chartUtilities class. It is a collection of utility methods for jfreechart, and includes methods for converting charts to image formats and something like that. We advise to decompose it by moving the static method to a special class depends on its business logic.

Example: *org.jfree.chart.ChartUtilities*
org.jfree.data.general.DatasetUtilities

Existing in all 10 versions but harmful

1. It should not be abstract
2. We can decompose it by moving the static methods to the special class depend on its business logic

Example: *org.jfree.chart.ChartUtilities*

1. Can be divided into 2 class , all the protected methods move to the class ValueAxis . Because these methods cannot be used by the other external class.

Example: *org.jfree.chart.ChartUtilities*

1. If we decompose this class , all the protected methods move to a new class. That means we will add an additional level between this class and its subclasses.

Type 3 – They are the God Classes which exist in the earlier version but are fixed in the later version, that means those god classes were fixed and the project was improved. For example, the chartpanel class which exist in all versions except in the last two versions. After reading change logs we found in version 1.0.13, the programmer changed or removed problematic methods and fixed this class.

Example : *org.jfree.chart.ChartPanel version 5, 7~12*
org.jfree.chart.ChartFactory version 5~10

Project Improved.

Type 4 - The last type is the classes become God Classes unexpectedly in some versions. It represent the design flaws in this project. For example, RendererUtilities and PolarPlot classes. We detect them as god classes in version 13 and 14. We found in these two version, the project change explosively. And these two class modified a lot, we considered they are

negatively design changes, and we advise to decompose the code. However, there is still a special class called .LineAndShapeRenderer, it used to design a renderer that draws shapes for each data item. We advise not to change it.

Example:

org.jfree.chart.renderer.RendererUtilities version 13~14
org.jfree.chart.plot.PolarPlot version 14

Flaw, bad design. Negatively Design Change.

Suggestion- Decompose the code.

org.jfree.chart.renderer.category.LineAndShapeRenderer version 7~14

Component affect project, but it is used to design a renderer that draws shapes for each data item.

Suggestion- We advise not to change it.

Versions / Class Name	1.0.5	1.0.6	1.0.7	1.0.8	1.0.9	1.0.10	1.0.11	1.0.12	1.0.13	1.0.14
org.jfree.chart.ChartFactory	√	√	√	√	√	√				
org.jfree.data.general.DatasetUtilities	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.ChartUtilities	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.plot.XYPlot	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.plot.CategoryPlot	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.plot.ContourPlot	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.plot.PiePlot	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.renderer.category.AbstractCategoryItemRen	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.renderer.xy.AbstractXYItemRenderer	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.axis.ValueAxis	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.renderer.xy.XYLineAndShapeRenderer	√	√	√	√	√	√	√	√	√	√
org.jfree.data.statistics.Statistics	√	√	√	√	√	√	√	√	√	√
org.jfree.data.xy.DefaultHighLowDataset	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.axis.NumberAxis	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.renderer.category.BarRen	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.axis.CategoryAxis	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.plot.ThermometerPlot	√									
org.jfree.chart.ChartPanel	√		√	√	√	√	√	√		
org.jfree.chart.renderer.AbstractRender		√	√	√	√	√	√	√	√	√
org.jfree.chart.LegendItem		√					√	√		√
org.jfree.chart.renderer.xy.CyclicXYItemRenderer			√							
derer.category.LineAndShapeRenderer			√	√	√	√	√	√	√	√
org.jfree.chart.plot.ThermometerPlot				√	√	√	√	√	√	√
org.jfree.chart.axis.CategoryAxis				√						
org.jfree.chart.plot.SpiderWebPlot							√	√		
org.jfree.data.DataUtilities									√	√
org.jfree.chart.renderer.RendererUtilities									√	√
org.jfree.chart.renderer.category.BarRen									√	
org.jfree.chart.renderer.xy.XYBarRender									√	√
org.jfree.chart.plot.PolarPlot										√
org.jfree.data.statistics.Regression										√

FIG: 10.1 All Occurancies In God Class

FIG: 10.2 All Occurancies In God Class

DATA CLASS

Here we represent our calculation of Total number of Data Class in all versions to the Total number of Classes in all versions in the project and how it evaluates from first version to last.

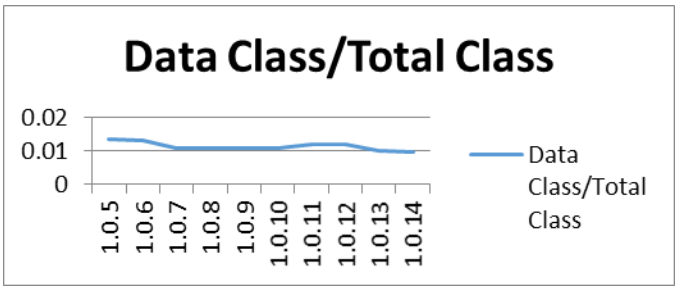

FIG: 11 Evaluation Of Number Of God to Total Classes

In our analysis, the number of data classes in all versions is stable. The result fluctuates between 6 and 7.By reading the source code, we found these data are useful, and it mainly used for transmitting data between classes. There are only 2 types of data classes. The first type is the class predefines useful information which will be used in other classes. For example, class ChartColor aims to define some public abstract attributes for color information which will be used in other classes. Another type is the class only contains accessor methods. Same as PiePlotState.

1. One class predefines useful information which will be used in other classes

   org.jfree.charat.ChartColor

   predefine some public abstract attributes for color information

2. *One class only contains accessor methods*

   org.jfree.chart.plot.PiePlotState

Versions / Class Name	1.0.5	1.0.6	1.0.7	1.0.8	1.0.9	1.0.10	1.0.11	1.0.12	1.0.13	1.0.14
org.jfree.chart.ChartColor	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.plot.PiePlotState	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.DrawableLegendItem	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.renderer.xy.StackedXYAreaRenderer.StackedXYAreaRendererState	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.event.ChartProgressEvent	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.block.BlockParams	√	√	√	√	√	√	√	√	√	√
org.jfree.chart.entity.PieSectionEntity	√	√								

FIG: 12 Occurancies In Data Class

Threats to validity

Internal Validity

The thresholds represent the weakest point of the detection strategies because they are established mainly using the human experience. In our project, there are some God Classes which have totally same code in all versions. However, it is not detected as a God Class in one version. We consider the reason for this problem is that we use top value of 25% as threshold in WMC to calculate god class. When the total number of classes changes, it would influence the result of god class. For instance, the class legenditem was detected as god class in nearly all version except version 7.it because the total number of classes in version 7 increased so that the top value changed.

org.jfree.chart.LegendItem version 5~6, 11~14

The total number of classes in version 7 increased.

External Validity

Our project is based on Metrics in Object-Oriented Systems for detection design flaws. Thus, this approach can be extended to other OO software. However the results of our study cannot be applied to other projects, because we analyzed a particular project, and the result cannot be generalized.

CONCLUSIONS

In this paper, we mainly explore four questions. The details of these questions are as below,

- What are the trends of the number of God Classes and Data Classes during the development of the project?
- Is the improvement or deterioration of the project related to the number of God Classes and Data Classes?
- Why did God Classes and Data Classes appear?
- Do all the God Classes and Data Classes be fixed?

After some detailed research, we can give answers for these questions. For the first question, the trend of god classes is upward. Data classes are stable. The result fluctuates between 6 and 7. By reading code, we found all data classes in the detected version are useful, we consider them harmless. But they actually are data classes, because they meet the definition of data class. In the body of their codes, they just only have accessor methods and public attributes. For the second one, we think it depends on the versions of the project. Next, for causes of god classes and data classes, we think it can divide into two parts. Firstly, there are some classes were detected as god classes at very beginning, even in the first version. They were caused by bad or wrong design. Secondly, some classes, in earlier versions, are not god classes. However, they turned into god classes in a particular version. For the last question, in the last version we detected, there still are some god classes, even some classes they modified in this version were became god classes. So, we cannot fix all god classes and data classes.

Finally, we could draw the following conclusions. Firstly, since we used the metrics-based approach for detecting design flaws in the open source project- JfreeChart. The result represents how this project evolved. However, during the development of the project, it generated several other problems when the development team tried to fix the design flaws or add new features in the project.

REFERENCES

[1] D.Ratiu, S.Ducasse,T.Girba, and R.Marinescu. Using History Information to Improve Design Flaws Detection. In Proceedings of the Conference on Software Maintenance and Reengineering(CSMR 2004).IEEE Computer Society,2004.

[2] Radu Marinescu. Detection Strategies: Metrics-Based Rules for Detecting Deign Flaws. In Proceedings of ICSM'04(International Conference on Software Maintenance),pages 350-359. IEEE Computer Society Press, 2004.

[3] Radu Marinescu. Measurement and quality in Object-Oriented Design. Ph.D. thesis, Department of Computer Science, "Politehnica"University of Timisoara,2002

[4] D. Gilbert, The JFreeChart Class Library. Object Refinery Limited,2008

[5] "GNU Project Releases Latest Version of GNU Classpath" By: Enterprise Open Source News Desk. Oct. 22, 2007. SYS-CON Media.