Proceedings of the 8th International Conference on Applied Informatics
Eger, Hungary, January 27–30, 2010. Vol. 2. pp. 359–365.

Providing Software Reengineering Technical Expertise Based on Similarity Metric

Mária Molnárné Nagy[a], **Norbert Bátfai**[b]

[a] University of Debrecen; Debreceni Informatikai Kutató-Fejlesztő Központ Non-profit KFT.
e-mail: nagymaria1@gmail.com

[b] University of Debrecen, Department of Information Technology
e-mail: batfai.norbert@inf.unideb.hu

Abstract

This is a case-study on eMedsolution integrated hospital information system, where the main question is, how can similarity metric help reenigneering the software. The CompLearn is a utility package, which uses a similarity metric based on Kolmogorov complexity. This package was used to analyze the Java source. With the result of the analysis is possible to find methods for helping reengineering, based on the knowledge of the software architecture.

Keywords: Kolmogorov complexity, CompLearn, Software reengineering

MSC: 68Q30 Algorithmic information theory

1. Introduction

1.1. A few words about eMedolution

eMedsolution is an integrated hospital system. It's a web based application coded in Java. The application support two browsers Mozzila FireFox and Internet Explorer. The system has many different customers, university hospitals, county hospitals, smaller clinics. It's a legacy system.

1.2. Legacy systems

Usually a legacy system is inherited, maybe old fashioned, changed many times. But a quite new program can turn quickly into legacy system, if the development is rapid, the changes are constant. A legacy system is a running system, which performs an important functionality and it seems doing it well, but in the background the maintenance of the program is hard. A legacy system has higher risk of bugs and breaks.

1.3. Sings of a legacy system

There are many symptoms, which warn, you need to act before your system broke - these are collected by Demeyer,Ducasse and Nierstrasz in [1]. The signs they defined (for example bad smells) show that a program need fix, although the functionality didn't break. Usually the symptoms don't occur in a system all together, but several at a time.

1.4. Why is eMedsolution a legacy system?

It changes all the time, because of the often changes in healthcare laws. New functions are developed, for the new technical developments, which become available in healthcare also. It has many functions, because the healthcare has many sectors − laboratory, radiology, emergency, inpatient departments, outpatient departments, . . . −, also some specialisms need special functions. All these causes generated bad smells in the code. So reengineering become a constant part task in development. Reengineering is a process that transforms one low-level representation to another, while recreating the higher level artifacts along the way.

2. Complearn and normalized distance

2.1. Complearn

What's Complearn? It's a utility package,which can help discover and learn patterns. It applies compression techniques. Complearn was written by Rudi Cilibrasi, Anna Lissa Cruz, Steven de Rooij [2] . It is based on the research of Cilibrasi, Paul Vitányi, and Ming Li about compression-based learning. They use different compression methods to approximate Kolmogorov complexity. The Complearn suite counts normalized compression matrix.

2.2. Normalized Information Distance and Normalized Compression Distance

The normalized information distance was introduced in [7], it is derived from Kolmogorov complexity ($K(x)$). A great book about Kolmogorov complexity and its appplications was written by M.Li and P.Vitányi [9].

Normailzed Information Distance:

$$d(x,y) = \frac{max\{K(x|y), K(y|x)\}}{max\{K(x), K(y)\}} \tag{2.1}$$

where $K(x|y)$ means the minimal number of bits required to reconstruct x from y. Since the Kolmogorov complexity is not computable, some approximation needed. If $K(x) \geq K(y)$, then $K(y|x) \approx K(xy) - K(x)$, where xy is the concatenation of x and y.

$K(x) \approx C(x)$, where $C(x)$ is the length of x compressed with a standard compression program. Using these approximations the normalized compression distance (NCD) is (defined also in [7]):

$$NCD(x,y) = \frac{C(xy) - min\{C(x), C(y)\}}{max\{C(x), C(y)\}} \tag{2.2}$$

3. Using Similarity metric on software sources

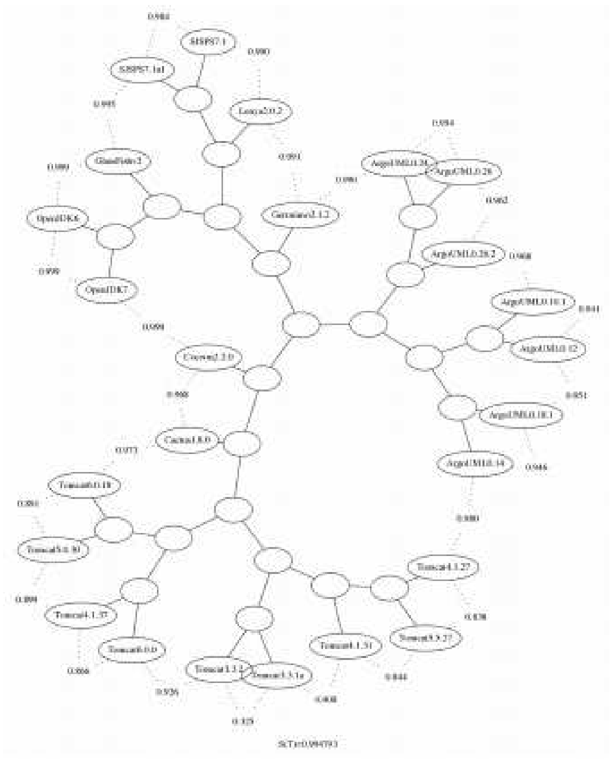

Figure 1: The generated tree of opensource software packages

3.1. Compare software packages

The ncd command in the Complearn package counts the normalized distance. Using this command on program sources can give a software metric. The first test for using similarity metric on software sources, for clustering sources was executed on Java open source projects. The source code is converted to a simplified version representing, the “OO skeleton”. Running ncd on this modified source creates a distance matrix. From the output matrix with the help of the maketree command generating an unrooted binary tree. Finally running neato – visualizer command from Graphviz package [5] – visualize the tree. This test confirms the theory, that similarity metric can help clustering. As on Figure 1. is visible, the package counted the distance between different versions of a software package smaller then between other software packages.[3]

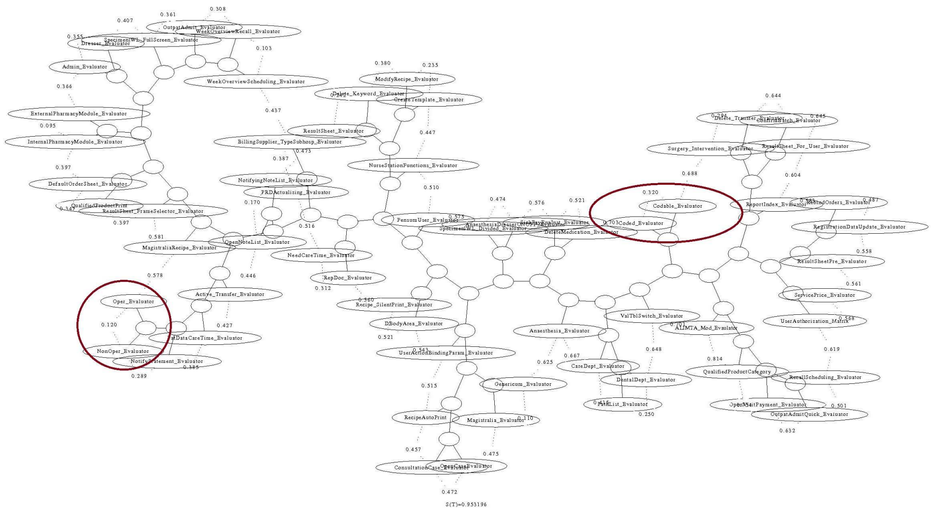

Figure 2: Tree from package com.ish.medsol.base.security.evaluator

3.2. Helping reengineering with analyzing source code

Similarity metric can help find duplications and similar object structures. If we found these points, the software engineers can decide, if these are really problems and need reengineering. The method to find the problematic points is: using Complearn on the packages of the software. If the analysis show two classes near to each other the engineer need to check the source if these classes are duplications or similar classes – which need reengineer to a new object structure.

The used commands are the next:

```
bash:  ncd -d dirname dirname > matrix.clb
```
- output matrix.clb, normalized compressed distance matix
```
bash:  maketree matrix.clb
```
- output treefile.dot, best fitting unrooted binary tree
```
bash:  neato -Tps > tree.ps
```
- output tree.ps visualization of the binary tree

Some examples found from the analysis of eMedsolution.

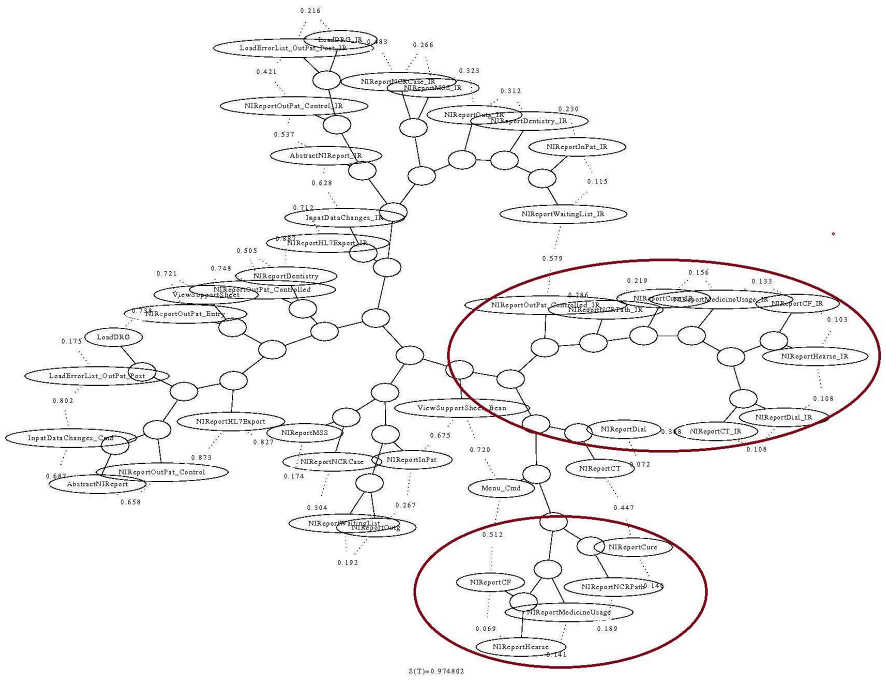

Figure 3: Tree from package com.ish.medsol.base.command.nirep

3.2.1. com.ish.medsol.base.security.evaluator package

The Codable_Evaluator and the Coded_Evaluator are quite close to each other, so they where analyzed manually. The comparison of the files made clear that, the two files make the same status checking process, only the statuses are different. So these two files can be generalized to a status checking abstract class. This reengineering can made safer the program, if the status checking process change, both files should be changed, and it's easy to forget one of them, specially if there are more then two statuses.

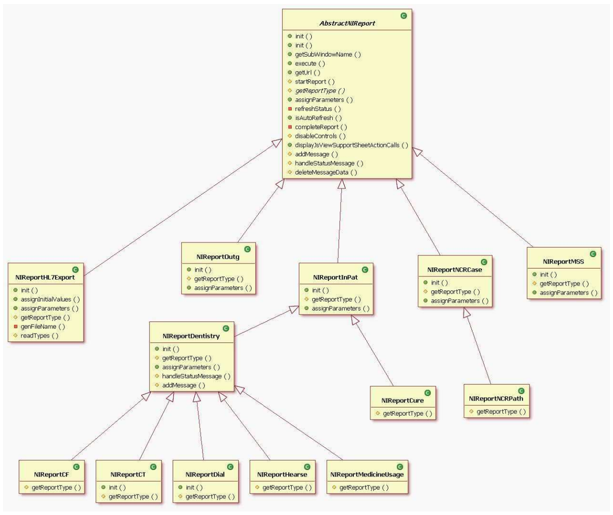

Figure 4: Partial class hierarchy tree of com.ish.medsol.base.command.nirep package

The Oper_Evaluator and the NonOper_Evaluator are the same except one constant value. They can be generalized in a parent class and, the classes could contain only the constant value. This reenginnering has the same reason then in the previous case.

3.2.2. com.ish.medsol.base.command.nirep package

The tree, generated from the package has two branches on the files are quite close to each other, see Figure 3. Examining the inheritance tree of the class group, shows that these files are on the same level with a mutual parent. On the first look its normal, when these files are similar, but if we watch the whole inheritance tree, it is maybe too complicated. So in a case like this the software engineers need to examine these object hierarchy (Figure 4) to determine, if these structures are good or not for the functionallity.

4. Summary

This process is not automated, like running a tool and the result will say we need reengineer these files. For reengineer a software – also if it is just a little change – the supervision of a professional – who know the whole system – is indispensable. This is the cause, why the automatic help maybe is not possible. The future work can find a generalization of the recognized possible points, like “similarity patterns” and make easier the work of software engineers.

References

[1] Serge Demeyer, Stéphane Ducasse, Oscar Nierstrasz *Object-Oriented Reengineering Patterns*

[2] http://www.complearn.org/

[3] Bátfai, N., Mobiltelefonos játékok tervezése és fejlesztése (Mobile Game Design and Development, hungarian),*PhD Dissertation and Thesis*, (2010), `http://www.inf.unideb.hu/~nbatfai/phd`

[4] *2008-2009Yearbook of Debrecen Developer Network* `http://dev.inf.unideb.hu:8080/c/document_library/get_file?p_l_id=10761&folderId=11505&name=DLFE-302.pdf`

[5] `http://www.graphviz.org/`

[6] Manny Lehman and Les Belady *Program Evolution: Processes of Software Change* London Academic Press, London, 1985.

[7] Ming Li, Xin Chen, Xin Li, Bin Ma, Paul M. B. Vitányi *The similarity metric*, IEEE Transactions on Information Theory, 2003, 863-872.

[8] Rudi Cilibrasi,Paul Vitányi, Ronald de Wolf *Algorithmic Clustering of Music Based on String Compression* Computer Music Journal, Winter 2004

[9] M.Li and P.Vitányi *An introduction to Kolmogorov complexity and its applications(3rded.)* Springer-Verlag,2008.

[10] Charles H. Bennett, Péter Gács, Ming Li , Paul M. B. Vitányi, Wojciech H. Zurek: *Information Distance*, IEEE Transactions on Information Theory, 1998, 44

Mária Molárné Nagy
Hungary, 4032 Debrecen, Egyetem tér 1.

Norbert Bátfai
Hungary, 4032 Debrecen, Egyetem tér 1.