A Framework of Source Code Comprehension for Software Maintenance

Hnin Pwint Phyu
University of Computer Studies,
Yangon, Myanmar

Thi Thi Soe Nyunt
University of Computer Studies,
Yangon, Myanmar

ABSTRACT
In maintenance of object-oriented software, one of the most important concepts is inheritance, which organizes classes into a hierarchy. The presence of the inheritance increases the number of potential dependencies within a program. Moreover, the comprehension of an existing software system can consume half or more of the maintenance time. The relationships among packages, classes, access modifiers, inherited classes and methods can affect on modification of the software. So, the proposed system uses the concept-based approach, Formal Concept Analysis (FCA) in order to comprehend the overall system (software project), and graphically visualize the modifier-based dependencies in terms of packages, classes, methods and inheritance relationship. The proposed system focuses on the problem how to provide an understanding of the software. Static java source code is analyzed in this system.

Keywords
Formal Concept Analysis, Software Maintenance, Program Comprehension

1. INTRODUCTION
In software maintenance, it is difficult to make modifications without understanding all relevant codes in a system. As a software maintenance point of view, a key difficulty in the maintenance and evolution of complex software systems is to recognize and understand the implicit dependencies that define contracts that must be respected by changes to the software**.** The developers spend considerable time in reading and comprehending programs in order to implement changes. The aims of the proposed system are – (i) to save time and effort for maintenance activities, (ii) to increase the overall comprehensibility of the system for software maintenance, (iii) to do reasonable estimates for modification without doing half the software maintenance effort. There is no clear means of identifying the required dependencies so that FCA is proposed to describe the dependencies before making maintenance by the developers (maintainers). The various applications of Formal Concept Analysis to software maintenance vary on their inputs, the concept lattices they create, and the use to which they put the concept lattices.

2. RELATED WORK
In this section, the related work of each phase of the proposed framework is presented. The three techniques of static concept location in terms of their respective strengths and weaknesses are analyzed in [4]. It discusses detail three techniques of static techniques- pattern matching, dependency search and information retrieval. It focuses on if the techniques for concept location are still needed and whether Object-Oriented structuring. It illustrates that concept location is an important programming activity even in OO programs.

Tilley et al. [8] presents a broader overview by describing and classifying academic papers that report the application of FCA to software engineering and application languages for the 47 papers in the survey [2, 7]. The use of FCA in the programming languages: C, C++, COBOL, Fortran, Java, Modula-2, Smalltalk, and the design or specification languages: OMT, UML and Z are surveyed. Both procedural and OO languages are represented. The attribute values record the size of any reported target application in KLOC (thousand lines of code).

The Algorithm CMCG [9] finds all lower neighbors of concept by using the rank of attributes in concept-matrix and generates corresponding Hasse graph. It is a novel notion is proposed for building concept lattice according to the concept-matrix. Then, the algorithm was validated in theorems and proofs. They implemented algorithm using machine learning dataset.

Kuznetsov and Ob"edkov [3] presented several algorithms that generate the set of all formal concepts and diagram graphs of concept lattices. Algorithmic complexity of the algorithms is studied both theoretically (in the worst case) and experimentally. The main parameters of a context K =(G, M, I) seem here to be the (relative to|M|) number of objects |G| and the (relative to |G|) number of attributes, the (relative, i.e. compared to |G||M|) size of the relation I, average number of attributes per object intent (resp., average number of objects per attribute extent) is presented.

3. FORMAL CONCEPT ANALYSIS TECHNIQUE
Formal Concept Analysis (FCA) provides a formal framework for identifying groups of elements sharing sets of properties. FCA is a method of exploratory data analysis that aims at the extraction of natural clusters from object – attribute data tables. It forms the clusters of objects having common attributes. These clusters, called formal concepts, are naturally interpreted as human-perceived concepts in a traditional sense and can be partially ordered by a subconcept – superconcept hierarchy. The hierarchical structure of formal concepts is called a concept lattice that represents structured information. FCA is composed of Formal Context, Formal Concept, and Concept Lattice.

FCA produces two basic outputs from the formal context:

- concept lattice: a hierarchical structure of conceptual clusters hidden in the data
- attribute implications: dependencies among attributes

A formal context is a triple of sets (G, M, I), where G is called a set of objects, M is called a set of attributes, and $I \subseteq G \times M$.For $A \subseteq G$ and $B \subseteq M$: $A'=\{m \in M \mid \forall g \in A\ (gIm)\}$, $B' =\{g \in G \mid \forall m \in B\ (gIm)\}$ [10]. A formal concept of a formal

context (G, M, I) is a pair(A, B), where A ⊆ G, B ⊆ M, A' = B, and B' = A. The set A is called the extent, and the set B is called the intent of the concept (A, B). For a context (G, M, I), a concept X =(A, B)is less general than or equal to a concept Y =(C, D)(or X ≤ Y)if A ⊆ C or, equivalently, D ⊆ B. For two concepts X and Y such that X ≤ Y and there is no concept Z with Z ≠ X, Z ≠ Y, X ≤ Z ≤ Y, the concept X is called a lower neighbor of Y, and Y is called an upper neighbor of X. This relationship is denoted by X p Y [3]. The (directed) graph of this relation is called a diagram graph. A plane embedding of a diagram graph where a concept has larger vertical coordinate than that of any of its lower neighbors is called a line (Hasse) diagram. The covering relations can be drawn where x<y and there does not exist z such that x<z<y. However, the problem of drawing line diagrams is not discussed here.

4. THE PROPOSED FRAMEWORK

The framework of the system consists of three phases as shown in Figure 1. Each phase contains two portions. Three main phases are Extraction phase, Relation identification phase, and applying FCA phase. Extraction phase consists of preprocessing step and extraction using pattern matching technique. Relation identification phase consists of collecting each class's information and identifying binary relation for constructing formal context. The last phase is applying Formal Concept Analysis (FCA) in the proposed system. In applying FCA, the first step is construction formal context according to the relation of the identification phase. The next step is computing formal concepts from the formal context. Finally, the resulting concepts are shown in concept lattice view.

4.1 Source Code Extraction

In extraction phase, the preprocessing part processes comments removal, split patterns, and split words. The source code files can be preprocessed using a set of different techniques including stop word removal, splitting identifiers, special token elimination and stemming. The information that the proposed system automatically extracts from software project is described below.

- names of classes
- modifiers of classes
- names of package
- names of methods
- modifiers of methods
- inheritance relation

This information is extracted using regular expression (pattern) matching technique. A regular expression is entered as part of a command and is a pattern made up of symbols, letters, and numbers that represent an input string for matching (or sometimes not matching). Matching the string to the specified pattern is called pattern matching. The proposed system searches using some of regular expression features from Table 1 and keywords of the java source code. Then, it retrieves a list of source code elements (class names, method (function) names, method modifier names and inheritance relationship between classes within a software project. The extracted information is used for identifying binary relation of formal context. For example, the proposed system extracts the class names with (^class) pattern.

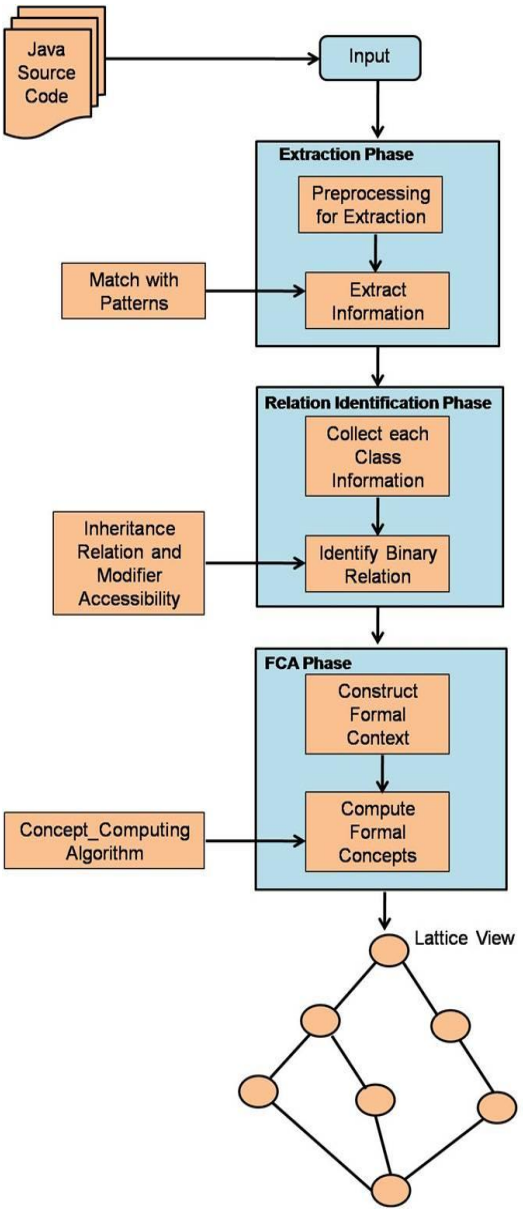


Fig.1 Proposed System's Framework

Table 1. Features of Regular Expression

Syntax	Meaning	Example
^	Pattern to be matched at the start of the input	^AB means the input starts with AB. A pattern without '^', e.g., AB, can be matched anywhere in the input.
|	OR relationship	A|B denotes A or B.
.	A single character wildcard	0.0 matches 0×0 and 020
?	A quantifier denoting one or less	A? denotes A or an empty string
*	A quantifier denoting zero or more	A* means an arbitrary number of As.
{}	Repeat	A{100} denotes 100As.
[]	A class of characters	[lwt] denotes a letter l, w, or t.
[^]	Anything but	[^\n] denotes any character except \n.

4.2 Identification of Binary Relation

Methods are identified depending on access modifiers, levels of accessibility between packages, and the inheritance relation. Default, private, public and protected modifiers are considered to set access levels for dependencies. The inheritance relation between classes is considered as "has a" relation by the property of inheritance [5]. The multi level inheritance is also considered in this system for getting the original hierarchy between classes. The identification of binary relation procedure is shown in Figure 2.

```
Begin
  for each class in project
    if (exist inheritance relation)
      if (check super class in project's classes )
        if (check super class in same package)
          if (check modifiers)
            process add methods except private modifier;
        else if (check modifiers)
          process add methods of public and protected modifiers;
      else show methods of each class;
    else  show methods of each class;
  end for
End
```

Fig.2 Binary Relation Identification Procedure

4.3 Applying FCA

In this phase, the proposed system constructs the formal context according to the binary relation from the identification phase. The rows represent class names (objects) and the columns represent method names (attributes). Table entries being ×'s and blanks indicate whether a class has or does not have the corresponding methods. For every input formal context, the system finds formal concepts common in formal context. In this case, the formal concept is computed using proposed Concept_Computing algorithm [6].

From the formal context, the first step of the algorithm finds independence of object sets (I) as initial concepts. The second step of the algorithm computes iteratively the next concepts under each concept of the concepts of the previous step. The algorithm terminates each concept is not further divided or the number of object set is equal to one.

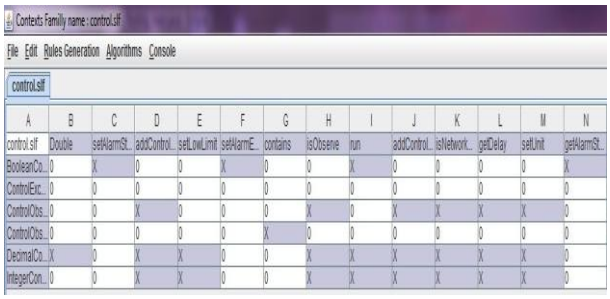
The proposed algorithm is similar with the process flow of the Bordat algorithm [1] except finding cover concepts. The concepts are computed by object sets union and attribute sets intersection. The proposed algorithm is considered the cover concepts (independent) by using subset function for object sets. So, the worst-case complexity of the proposed algorithm is O ($|G|^2|M||L|$) according in [3]. The subset function of the algorithm to find (Independence) cover concepts (see Figure 3) is only presented in this paper.

1. for all $b_j \in M$ do
2. $A_j \leftarrow$ Max (A);
3. end
4. for all $A_k \in M$ do
5. for ($k \neq j$) do
6. if ($A_k \subseteq A_j$) then
7. $I \leftarrow A_j$;
8. go to line 16.
9. else $I \leftarrow A_j$;
10. $M \leftarrow M - A_j$;
11. $A_j \leftarrow A_k$;
12. go to line 4.
13. end if.
14. end
15. end
16. return I{A}.

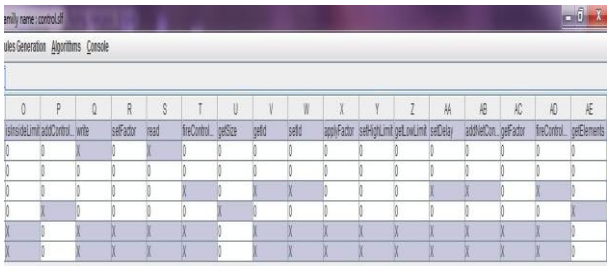
Fig.3 Independence Procedure

5. SAMPLE CASE STUDY

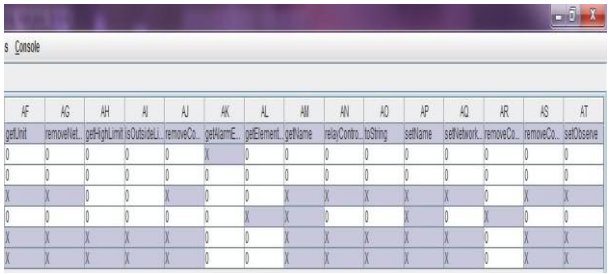
This case study is presented in Figure 4 and Figure 5 using a package (control) of Java Human Machine Interface (JavaHMI) from the site of sourceforge.net. The package consists of 6 classes (BooleanControlObserver, ControlObserver,etc) and 46 methods (contains, run, etc). The context (a), (b) and (c) in Figure 4 are described the formal context of control package.



(a)



(b)



(c)

Figure.4 Formal Context of Control Package

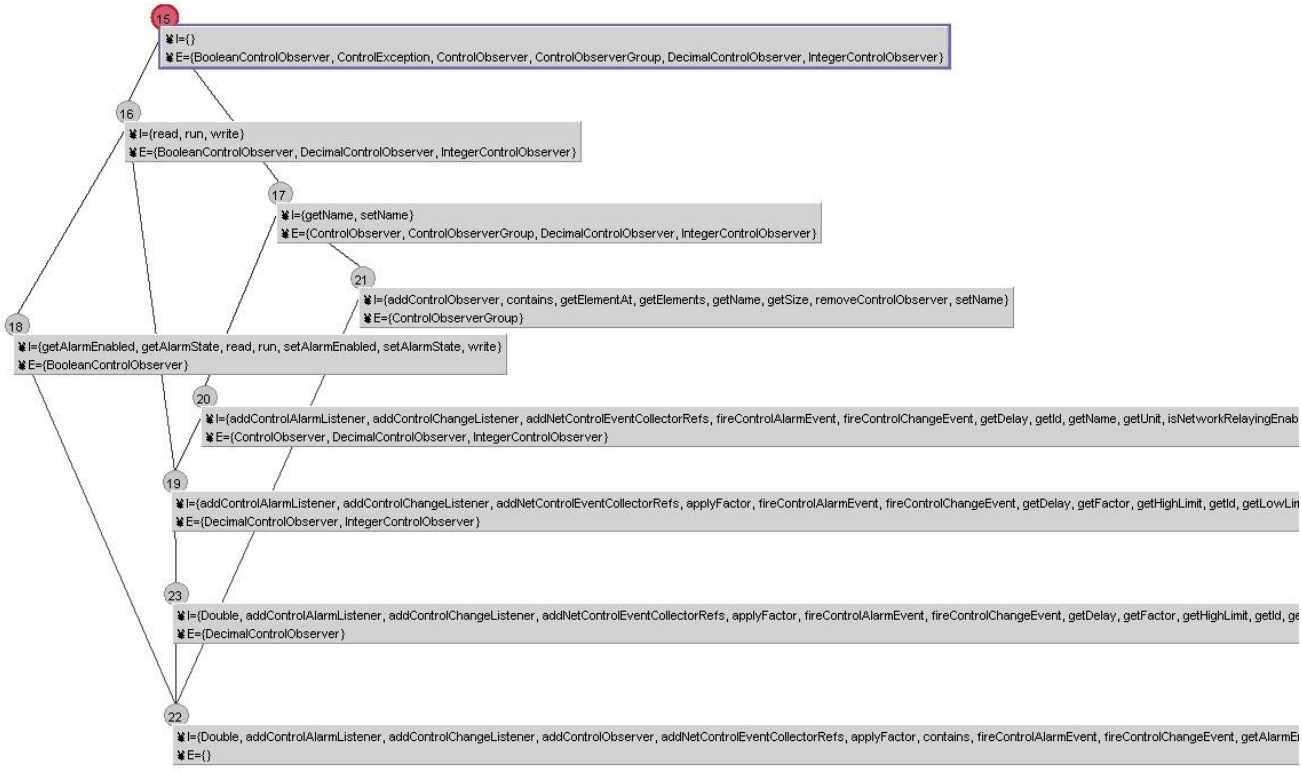


Figure.5 Concept Lattice of Control Package from JavaHMI project

Table 2. Information of the Three Projects

Projects	Names	No of Files	No of Packages	No of Classes	No of Methods	Lines of Code	Precision (Exactness)	Recall (Complete-ness)
Prj 1	Micro simulator	17	1	17	73	2278	1	1
Prj 2	Antlr parser	48	2	36	349	7703	0.90	1
Prj 3	Java HMI (Human Machine Interface)	67	9	58	621	13082	0.96	1

The proposed system extracts the intended information from the source package by using pattern matching and identifies the binary relation by proposed identification procedure. Then, it constructs formal context according to the relation. The formal concepts are computed by the proposed algorithm from the formal context. Finally, a hierarchical structure and dependencies among attributes are presented in concept lattice view. In the concept lattice, each concept is represented by a little circle so that its extension (intension) consists of all the objects (attributes) whose names can be reached by a descending (ascending) path from that circle. The proposed system is implemented FCA by using Galicia platform.

6. EXPERIMENTAL DETAILS

The three medium java projects are analyzed and the experimental results of these are presented in this section. The detail information of the projects and their precision and recall result is shown in Table 2.

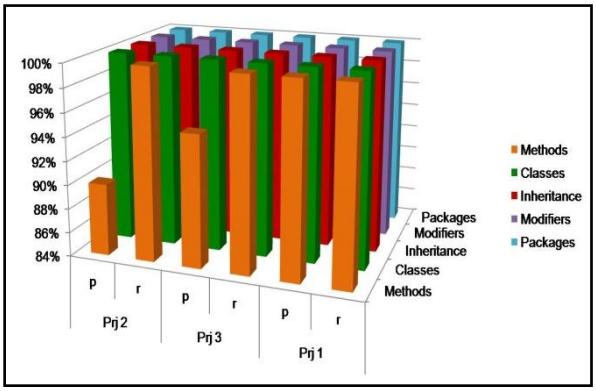


Fig.7 Correctness and Exactness Percentage of Extraction

The precision (p) represents exactness and recall (r) represents completeness in extraction. So, the overall extraction is shown in Figure 7. The largest precision result of extraction is found in project 1 as it is the simulation system that is composed of simple and similar structure. The least precision result of extraction is found in project 2 as it is the parser software that is composed complex structure. However, the recall result of extraction is linear in all projects.

7. CONCLUSION

The proposed system reduces maintenance effort by locating the relevant codes for comprehension of the existing system. The proposed system using FCA is to help the developers for doing the maintenance activities easily.

In concept lattice view, the nodes represent classes contained methods, and the edges can be seen implication of the methods. Besides, estimation of modification for which class is closely related with other classes, which methods changes can affect other parts, which methods should not be changed and so on can be made by seeing methods implication.

8. REFERENCE

[1] J. Bordat, "Calcul pratique du treillis de Galois d'une correspondence", Math. Sci. Hum.,1986, no. 96, pp. 31–47.

[2] U. Dekel, "Applications of Concept Lattices to Code Inspection and Review", 1993.

[3] S. Kuznetsov and S. Ob''edkov, "Comparing Performance of Algorithms for Generating Concept Lattices", ICCS'01 Int'l. Workshop on Concept Lattices−based KDD, 2001.

[4] A. Marcus, V. Rajlich, J. Buchta, M. Petrenko, and A. Sergeyev, "Static Techniques for Concept Location in Object-Oriented Code", 2005.

[5] D.C.C.POO, National University of Singapore, Chee Seong Tan, "Learn To Program Java", Second Edition, 2005.

[6] H.P.Phyu, T.T.S.Nyunt "Concept-based Source Code Analysis for Software Maintenance", in Proceedings of the 11th International Conference on Computer Applications (iCCA, 2013), Yangon, Myanmar, February 2013, pp. 339-343.

[7] G. Snelting, F. Tip, "Reengineering Class Hierarchies Using Concept Analysis", ACM, 1998.

[8] T.Tilley, R. Cole, P. Becker, "A Survey of Formal Concept Analysis Support for Software Engineering Activities", 2005.

[9] S. Wang, Z. Chen, D. Wang, "An Algorithm based on Concept-Matrix for Building Concept Lattice with Hasse", IEEE , 2007.

[10] R. Wille, "Formal Concept Analysis as Mathematical Theory of Concepts and Concept Hierarchies", Springer-Verlag Berlin Heidelberg, pp. 1– 33, 2005.