Proceedings of the 5th WSEAS International Conference on Applied Computer Science, Hangzhou, China, April 16-18, 2006 (pp919-924)

Supporting Software Design based on Comments in Codes

Tetsuya Yoshida
Hokkaido University
Graduate School of Information Science and Technology
N-14, W-9, Sapporo, Hokkaido 060-0814
JAPAN

Abstract: This paper proposes a method for supporting software design based on comments which are usually inserted into a source code. It is believed that a software designer often leaves hints or clues with respect to the rationale for each module or software component as comments when he/she designs and implements software as a source code. Comments in the source code are utilized for organizing the pieces of the codes into a tree structure so that the overall structure of the code can be explicitly represented. Case based support is then invoked for enabling the effective reuse of other software (source codes) based on the constructed structure for the source code, especially at the upper process in software design. A prototype system with the method has been implemented on a personal computer with Java language, and experiments were conducted to investigate its effectiveness. The results indicate the effectiveness of the proposed approach.

Key–Words: Software Design Support, Comment, Design Rationale, Case Based Approach, Interactive Support

1 Introduction

In contrast to the rapid performance improvement in hardware, it has often been pointed out that performance improvement in software design is rather marginal. To tackle this issue, various research efforts have been conducted to improve the productivity of software design. For instance, besides the widely used sequential computation in von Neumann model, one branch of research fields tries to pursue other computation models. Functional computation model [1] treats a program as a set of function definitions, and conducts computation by calculating the value of the functions. Logic programming model treats a program as a set of logical formulas [9]. Removing side effects, which are utilized to conduct computation in Von Neumann model, enables to make computer programs simple and easy to debug, and thus contributes to improving the productivity of software design. As another research effort, component-ware [7, 10] has been pursued to make software components replaceable and pluggable with respect to software architecture. Component-ware also contributes to improving the productivity of software design since it facilitates the reuse of established software components.

As for software design support, many support tools have been developed and provided. For instance, a lot of CASE (Computer Aided Software Engineering) tools are available and widely utilized. These tools mainly focus on automating routine works within the framework of visual programming by providing easy to use GUI (Graphical User Interface) interfaces [2]. These tools are useful for semi-automated code insertion/completion and debugging, and thus software design at the lower process can be supported. However, compared with the support for the lower process, decision making at the upper process in software design has not been well supported yet. Decision making at the upper stream is also important since it greatly affects the overall structure and organization of software.

This paper proposes a comment-based software design support method by utilizing comments which are usually inserted into a source code. Our hypothesis is that a software designer leaves hints or clues with respect to the reason or rationale for each module or component as the comments on the source codes when he/she designs and implements software [5, 8]. Comments are utilized for organizing the pieces of source codes into a tree structure. Based on the constructed structure for the source code, case based support is then invoked for enabling the effective reuse of past software (codes), especially at the upper process in software design. A prototype system has been implemented with the proposed support method and experiments have been carried out to investigate the effectiveness of our approach.

In our approach the reason or rationale for software design is treated as a kind of context in software design, comments are utilized to capture the rationale

in source codes. Utilization of designer's overall ir age for the artifact in design is also proposed in [1 for web page design support. Utilization of "cor ments" is also utilized to facilitate cooperation amoı computational agents in [12].

The proposed method aims at working as an inte active software design support during design proce and is expected to enable the effective reuse of sour codes by exploiting the partial modules for the curre design. Admittedly the proposed support framewo seems weak as the software design support when it compared with the framework of automated prograr ming from the viewpoint of utilizing the formal s mantics of software. However, since our approach c be utilized for software design support even if a so ware designer just leaves his/her vague and nebulous idea at the upper stream in software design as comments in his/her source codes, it is expected to work as more flexible and robust software design support.

This paper is organized as follows. Section 2 describes our framework of software design support and the system architecture of our system. Section 3 explains the details of the support method based on comments in a source code. Section 4 describes the implementation of the system and reports the result of experiments for the system. Section 5 gives brief concluding remarks and indicates future directions to extend our approach.

2 A Framework of Software Design Support

2.1 Preliminaries

In general, software design includes conceptual design, algorithm or procedure/function design, and implementation into a source code. We call a set (or, sequence) of a set (or, sequence) of machine interpretable sentences as a source code in this paper. A source code can be simply referred as a code if it is clear from the context. A pair of a source code and the description of the rationale for the code is referred as software in this paper. We assume that the rationale is represented as the comments in the code. Our method aims at supporting software design which includes conceptual design as well as implementation into the source code. Note that our method deals with the support for middle or small scale software design, and it is assumed that a software designer conducts the above process. Support for large scale software is beyond the scope of this paper.

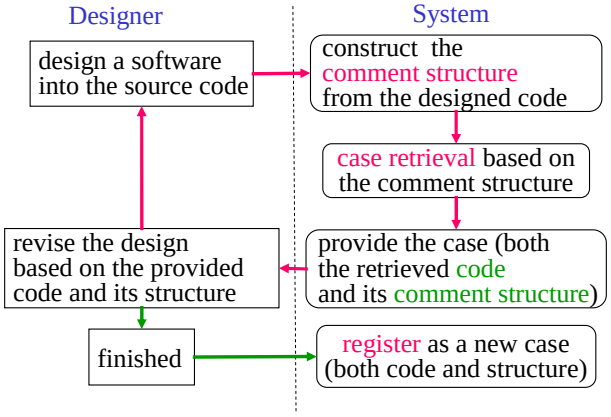

Figure 1: Framework of Software Design Support.

2.2 Comments in Source Codes

This paper proposes to utilize the comments which are often inserted into a source codes. The reasons for utilizing the comments in a code for software design support are:

- i). comments are inserted by the software designer, who implemented the source code and thus should understand the software and the source codes.
- ii). comments are often inserted for explaining a specific module or function in the source code.

From i), it is likely that the reason or rationale behind the software is left as a kind of hints in the comments. From ii), it would be possible to effectively reuse past codes based on the comments by partitioning them into several modules with specific functions.

2.3 Overview of Software Design Support

In this paper, we propose a software design support system which utilizes comments in a source code. Overview of support process in the proposed system is shown in Figure 1. First, a software designer implements the tentatively designed software into a source code and triggers the process (upper left in Figure 1). In our approach, it is assumed that the software designer is cooperative and inserts comments onto the source code. For the code, the system constructs what we call a comment structure based on the inserted comments. A comment structure is defined as a tree structure which organizes the comments. The details of construction is explained in Section 3. After that, the system retrieves a similar case (source code) with respect to the comment structures, and provide them to the designer. Here, a case is defined as a pair of a source code and the corresponding comment structure.

If the retrieved code is similar to the currently designed one, then the user (in this paper, a user plays

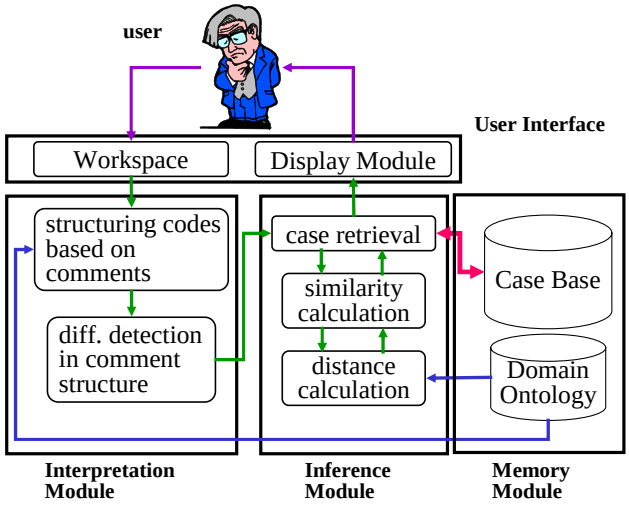

Figure 2: System Architecture.

the role of software designer) carries on the current software design by referring to it. If not, the user conjectures the difference between the retrieved software and the currently designed one in terms of the comment structure, and modifies the code as well as the comments. The above processes are repeated interactively between the designer and the system in order to gradually brush up the designed software.

Our system aims at supporting a software designer in the following situations:

- the designer has his/her clear image for software design, but he/she wants to consult other concrete examples (codes) in order to realize it.
- the designer is familiar with the details of implementation, but he/she wants to consult how various modules are organized in other source code consistently.

For the former situation, the designer can utilize or refer to the concrete code in the retrieved case. For the latter situation, the designer can refer to the overall structure, not the actual implementation, in the retrieved case.

2.4 System Architecture

Figure 2 shows the architecture of our system. Basically, it consists of three modules:

- ∗ An interpretation module: this module analyzes the current source code and organizes it into a tree structure based on the comments in it.
- ∗ A memory module: this module contains 1) a case base, which stores the past designed software in conjunction with the structure based on the comment on the software and 2) a domain ontology with respect to the words which are usually used as the comment in the domain at hand.
- ∗ An inference module: this module contains 1) the distance calculation module for the comment words, 2) the similarity calculation module for the comment structures based on the distance, and 3) case retrieval module from the case base.

In addition, a user interface module is provided, which contains 1) a workspace for editing the source code and the comment structure, and 2) a display module to show the retrieved case.

3 A Comment based Support Method

For organizing a source code into the corresponding comment structure described in Section 2.3, two approaches, a syntax based approach and a semantic based one, are utilized.

3.1 Syntax based Structuring

A software designer often inserts or arranges indents in a source code to separate it into the functionally meaningful chunks of sentences. Thus, the syntax based approach exploits the indentation of sentences in a source code. Note that indents can be inserted into a code with respect to 1) sentence and 2) comment. Although the indent of a comment can depends on the “style” or “preference” of designers and thus can vary depending on designers, that of a sentence can be conceived as reflecting the role of the sentence in the software. Thus, the indent of sentence is utilized in syntax based approach. This approach deals with:

- indent position: the position of the sentence (or, the set of sentences) on which the comment is put.
- recursive comment: the comment which is inserted inside the sentence with other comment.

Based on our preliminary analysis, the correspondence between a comment and the set of sentences to which the comment refer is treated as

- ○ a comment beside a sentence: it refers to the set of sentences bellow the comment.
- ○ a comment on the top of a sentence: it refers to the set of sentences until the comment with the same indent position appears.

The comment structure is constructed by the following procedure:

- ⋆ 1) a node is created for each comment, and a keyword in the comment is treated as the node label.
- ⋆ 2) as for a recursive comment, a tree structure is constructed for each nested comment, the constructed

tree structures are treated as the subtrees of the node for the outmost comment.

Figure 3 illustrates an example of tree structure for the comments in a code.

```
// connect with database
Connection[] sqlInit() throws SQLException,ClassNotFoundException {
   //DriverManager.setLogStream(System.out);
   //JDBC driver
   Connection cons[]={null,null};
   Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
   // connect with Oracledatabase
   cons[0]=DriverManager.getConnection (
   "jdbc:odbc:oracletest", "SYSTEM", "MANAGER");
   // connect with Accessdatabase
   cons[1]=DriverManager.getConnection (
                "jdbc:odbc:dream_acs", "Admin", "");
   return cons;
w2}
//initialize
public void init(){
   setLayout(null);
```

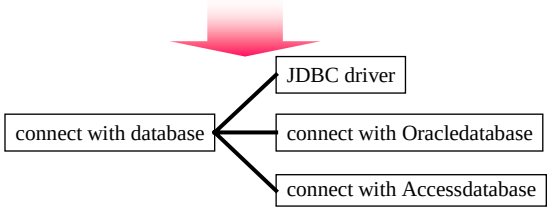

Figure 3: An example of tree structure with respect to the comments in the source code.

3.2 Semantics based Structuring

When the literal words in a comment are utilized as the node label in a comment structure, since the exactly same words are rarely utilized, it gets very hard to take matching or correspondence between comment structures. One of the reasons why software designers can understand codes despite the words in comments vary among them is that they can utilize domain-specific knowledge to understand the content implied by the comments. Thus, it is necessary to incorporate domain-specific knowledge for the designed software at hand into the system.

Ontologies are introduced as explicit specification of a conceptualization [3, 4]. Semantic based approach is conducted by utilizing a domain ontology which describes the typical vocabularies in the application domain. Keywords are extracted from comments based on the domain ontology and they are utilized as the node labels in the comment structure. Thus, it plays the role of normalizing the node labels in a comment structure so that its generalization capability can be increased to enable the matching with other structures. Figure 4 shows an example of domain ontology. Currently it is assumed that the vocabularies are structured into a tree structure to make the extraction of keywords from a comment simple.

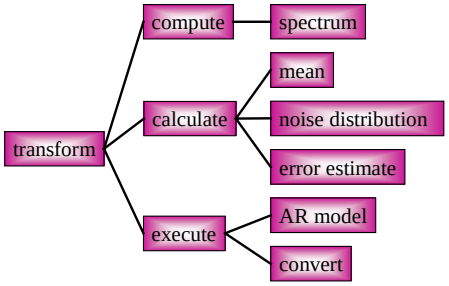

Figure 4: Example of Domain Ontology.

3.3 Case based Support

Case based support [6, 11] is utilized in our approach to retrieve the past similar codes. As described in Section 2.3, a case is defined as a pair of a source code and the corresponding comment structure. A software designer can refer to the past similar codes with respect to the organization of the codes in terms of the inserted comments. Case retrieval is conducted by defining the distance between the comment structures. Figure 5 illustrates the calculation of distance between comment structures. The distance between comment structures is defined as follows:

- the distance between words in the nodes is defined as the path length between the words in the domain ontology.
- the distance between the nodes in the comment structure is defined as the sum of 1) the number of lower nodes and 2) the distance between the words.
- the correspondence of nodes between comment structures is taken as the one with the shortest distance.
- a penalty is given to the upper node in a comment structure when it is impossible to take the corresponding node in the other comment structure.

The comment structure acts as a kind of media to represent design rationale of a code, and the distance calculation enables the case based support in terms of design rationale.

4 Evaluation

4.1 Implementation

The system with the method described in Section 3 has been implemented on a personal computer with Java language. Figure 6 shows a snapshot of Graphical User Interface (GUI) in the system[1]. Each window in the figure plays the following role:

[1]Snapshots of the GUI includes Japanese characters since the system was evaluated by Japanese students in our laboratory.

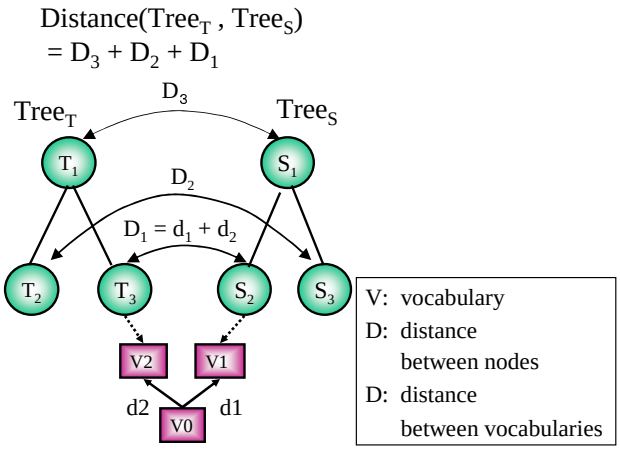

Figure 5: Distance between Comment Structures.

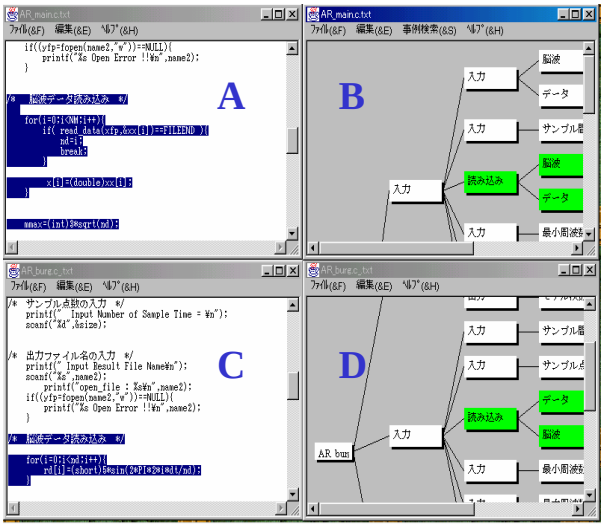

Figure 6: User Interface.

- Program Edit Window (A): The user conducts the coding of software in this window. By selecting "add case" from the menu, he/she can register the comment structure of the currently edited code in conjunction with the code as a new case into the case base.
- Comment Edit Window (B): The user can edit the comment structure for the software in (A). In addition, he/she can retrieve similar past cases from the case base by specifying the partial structure in this window.
- Program Display Window (C): This window shows the retrieved code from the case base.
- Comment Structure Display Window (D): This window shows the comment structure of the retrieved code shown in (C).

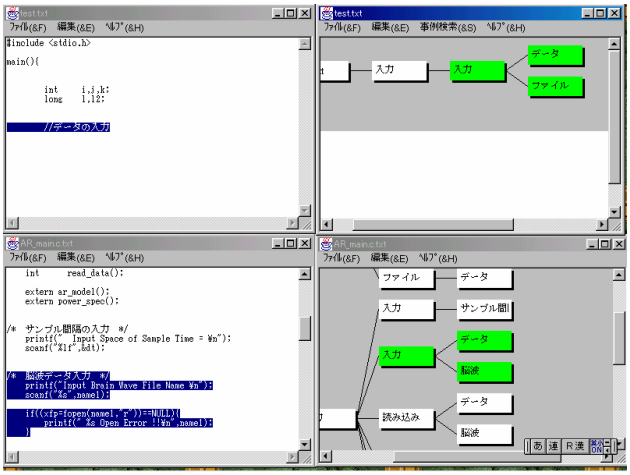

Figure 7: A snapshot of system usage (at the initial phase).

4.2 Experiments

Experiments were conducted to evaluate the implemented system. Five Japanese students in our laboratory participated in the experiment. All of them had the knowledge and experience of programming in C language. They were required to design a FFT (Fast Fourier Transfer) program based on DFT (Digital Fourier Transfer) programs in the frequency analysis. Among them, two subjects had detailed knowledge for the frequency analysis; the rest did not.

Figure 7 shows a snapshot of of case based support with respect to the data acquisition module based on the comment structure at the initial phase in software design. A Similar case was retrieved from the case base in terms of the organization of a source code based on the comment structures.

After completing the implementation of the assigned task, the subjects were asked to specify the subjective evaluation in 5 grades (5:excellent, 1:poor) with respect to:

- **retrieval**: the precision for the retrieval of preferable cases
- **effect**: the effectiveness of the system at the **initial**, **middle**, and **final** phase in software design.
- **comprehension**: to what extent the comment structure is effective to facilitate intuitive understanding of the code.
- **structure**: to what extent the comment structure reflects the organization of the code.

In addition,

- how the system was utilized
- how the retrieved cases (programs) were utilized to carry out the current software design

were also asked with a questionnaire.

Table 1: Result of experiment (frequency analysis domain, subject A,B: expert, C,D,E: novice).

	A	B	C	D	E	Ave.
retrieval	4	2	3	4	3	3.2
effect (initial)	3	4	4	4	4	3.8
effect (middle)	4	3	2	3	2	3.0
effect (final)	4	2	2	3	2	2.6
comprehension	4	2	3	4	4	3.4
structure	4	3	4	3	3	3.4

4.3 Results

Table 1 summarizes the results of the experiment. Table 1 indicates that the proposed system can be utilized as the software design support tool at the initial and middle phase. However, it might be not so effective in the final phase. Our current conjecture for these results is that, referring to other similar codes is helpful to form the overall software design at the initial or middle phase in software design. On the other hand, since the final phase is mainly devoted to bug fix, utilization of the system did not work well.

From the above results, it can be said that the interaction with the proposed system can be helpful to come up with the idea for the overall organization of the designed code at the upper process in software design The comment structure, which is constructed based on the comments inserted into the source code, seems to be effective to organize and represent the reason or rationale behind the codes.

5 Concluding Remarks

This paper has proposed a comment-based software design support method by utilizing comments which are often inserted into source codes. Comments in a source code are utilized for organizing the code into a tree structure so that the overall structure of the code can be explicitly represented. Case based support is utilized based on the constructed structure for enabling the effective reuse of past codes, especially at the upper process in software design. A prototype system which incorporates the described method has been implemented. The results of the experiments are encouraging and suggest its effectiveness. As an immediate future plan, protocol analysis and ethnomethodology analysis are planed to conduct more intensive analysis of the obtained results.

Acknowledgements: The author is grateful to Dr. Takayuki Yamaoka for the fruitful discussion about the distance calculation and to Mr. Koichi Hashimoto for the implementation of the system.

References:

[1] H. Abelson, G. J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 1996.

[2] M. M. Burnett. Visual Programming. In J. G. Webster, editor, *Encyclopedia of Electrical and Electronics Engineering*. Wiley-Interscience, 2001.

[3] N. Guarino. Concepts, attributes, and arbitrary relations – some linguistic and ontological criteria for structuring knowledge bases. *Data and Knowledge Engineering*, 8, 1992, pp.2 49–261.

[4] A. Hameed, D. Sleeman, and A. Preece. Detecting mismatches among experts' ontologies acquired through knowledge elicitation. *Knowledge-Based Systems*, 15, 2002, pp. 265–273.

[5] W.L. Johnson. Understanding and Debugging Novice Programs. *Artificial Intelligence*, 42(1), 1990, pp. 51–97.

[6] J.L. Kolodner. Improving human decision making through case-based decision aiding. *AI Magazine*, SUMMER 1991.

[7] S.M. Lewandowski. Frameworks for component-based client/server computing. *ACM Computing Surveys (CSUR)*, 30(1), 1998, pp. 3–27.

[8] S. B. Shum. Design Argumentation as Design Rationale. In A. Kent and J.G. Williams, editors, *The Encyclopedia of Computer Science and Technology*. Marcel Dekker, Inc., 1996.

[9] L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. MIT Press, 1994.

[10] C. Szyperski, D. Gruntz, and S. Murer. *Component Software - Beyond Object-Oriented Programming*. Addision-Wesley, 2002.

[11] Linda M. Wills and Janet L. Kolodner. Towards More Creative Case-Based Design Systems. In David B. Leake, editor, *Case-Based Reasoning – Experiences, Lessons, and Future Directions*, chapter 4, The MIT press, 1996, pp. 81–91.

[12] T. Yoshida, K. Hori, and S. Nakasuka. A Cooperation Method via Metaphor of Explanation. *IEICE Trans. Fundamentals.*, 81EA(4), 1998, pp. 576–585.

[13] T. Yoshida, M. Watanabe, and S. Nishida. An Image based Design Support System for Web Page Design. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 2006. in press.