JAWS2012

JAWS2012

Passive Code Review with Automatically-Generated Model Diagram

Yoshizane Hirose — College of Information and Systems, Muroran Institute of Technology
12061014@mmm.muroran-it.ac.jp

Shun Hattori — College of Information and Systems, Muroran Institute of Technology
hattori@csse.muroran-it.ac.jp

keywords: Code review. Flowchart. Software engineering. Model diagram.

Summary

Today, it is getting more important to manage the quality of source code in more large-scale software development. In such system development, code review is one of effective means to guarantee the minimum quality of source code. However, the traditional techniques of code review are often criticized as inefficient or unproductive, because they have the risk of endless review. Therefore, this paper proposes a novel support system for code review to generate an effective procedure of "Passive Code Review" within a reviewer-given period of time. And also our research implements a prototype tool to generate a flowchart of source code to help reviewers understand the source code, and validates the effectiveness of its functions based on subjective evaluation by the reviewers.

1. Introduction

Because Information and Communication Technology supports our life as one of social infrastructures, it is indispensable for our society. Software technology plays important role as the core of ICT. Software continues to require not only higher reliability and integrity but also larger scalability, more sophistication, and higher functionality. Because of such an environment, we expect much of Software Engineering. Software engineering was born after invention of computers, so its history has yet been not long. However, it has achieved great results such as clarification significance of code review process in software development and proposals of methods of the process [Ferreire 10]. For example, how to organize review's points of view as to each process (e.g., design process or coding process) and how to develop the worksheets for review have been proposed and applied to practical situations. Even today, many other researches and developments continue to be carried out.

Code review is an immemorially-used method expected to improve the quality of software and to detect its vulnerable points by inspecting it from various view points by human hands [Laitenberger 02, Aksulu 10]. They represent that code review accomplishes few fruitage for invested human resources, compromises the reliability and productivity of software development, and also requires additional processes such as hearing from software developers to accomplish more fruitages. Today, to make more considerable achievements in code review, derivative model diagrams while development and static code analysis tools are often utilized [Remillard 05]. But it is difficult to substitute one of them for hearing from software developers because of their potential problems.

From the above-mentioned findings, we focus on "Code Review" that tends to be often made light of even though it is very important for software development processes, propose a novel tool for code review, and verify the basic effectiveness of our developed tool.

2. Proposed Method

This section extracts and analyzes problems of "Active Code Review" which seems to be the major method of existing code reviews, and proposes the novel concept of "Passive Code Review" to solve them.

2·1 Extracting Problems of Active Code Review

Many reviewers often browse their targeted source code by a text editor or print out it for code review. This method (called "Active Code Review" in this paper) is that reviewers select the targeted area of source code by themselves all at once at will, and read and try to understand it.

The Active Code Review method is dependent on each reviewer's previous knowledge and programming experience, and is expected to give her enough feedback after

her seeing through the whole composition of her targeted source code. Therefore, Active Code Review seems to be aimed at intermediate level of software engineers, because it needs reviewer's high accomplishment and experience as software engineers. But reviewers cannot always obtain enough feedback by Active Code Review, even if they have high accomplishment as software engineers. At actual development sites, development programmers often set up a meeting to explain the logics of their source code for the others. Active Code Review is nothing more than local (not global) analysis of source code, and so it is not always effective in association with higher functionality and more multiple functions of software.

By analyzing the above-mentioned problems of the existing Active Code Review, in which reviewers read and understand one line of their targeted source code at a time, it is found to be difficult for them to see through and interpret the whole of their targeted source code. Based on these findings, this paper proposes a novel code review method (called "Passive Code Review") to enable reviewers to see through the whole of their targeted source code by passive browsing [Hourai 03].

2·2 Passive Code Review

The Passive Code Review method is that reviewers specify their code-review time, a support system automatically generates an effective procedure for code review in their specified time, and they could understand their targeted source code by passively browsing the procedure.

A Passive Code Review system can monitor more precisely where a reviewer is browsing in her targeted source code, and offer the reviewer supplementary information for the source code more timely and appropriately. In addition, the system needs more effective and novel method for code review, because a reviewer has to review her targeted source code in a limited time. And moreover, as an effective code review method for Passive Code Review, this paper proposes a novel method to provide reviewers with "a visualization to see through their targeted source code" and "a visualization of logics of the source code according to their browsing location" by using model diagrams. Methods to generate model diagrams from a reviewer's targeted source code and enrich her understanding of the source code were also used concurrently by source code analysis tools in the existing Active Code Review. However, the existing code analysis tools have not yet been able to extract valuable information from logics of the source code intricately-intertwined with its algorithms and external systems etc., and are far cry from achieving successful results. Meanwhile, in Passive Code Review, a review of source code and its visualization by model diagrams are processed concurrently as shown in Figure 1. The code review complements information discarded by the model diagrams, and the model diagrams are adapted to a reviewer's browsing location of the source code. This method could give reviewers high feedback in their specified limited time.

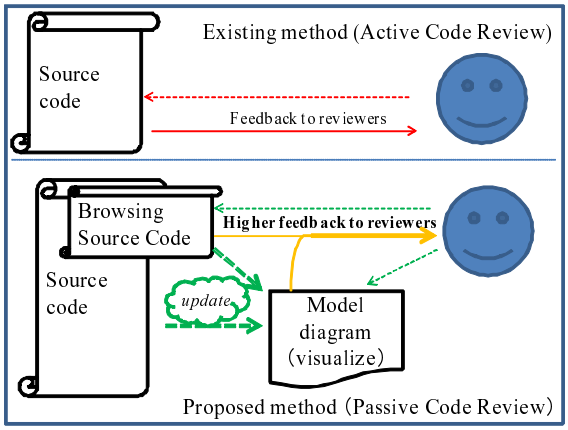

Fig. 1 Passive Code Review

3. Automatic Flowchart Generation for Passive Code Review

This section studies differences between model diagrams generated by the existing source code analysis tools and drawn by software developers themselves, specifies for our proposed Passive Code Review, and makes a selection of additional functions to flowchart diagrams.

Our proposed model diagrams for Passive Code Review are assumed to be combined with only code review of a targeted source code. They can allow reviewers to avoid such mistakes as false recognition of their abstract information by using the source code together, and also can be drawn in abstract notation. In addition, our system monitors a reviewer's browsing location in her source code, and enables her to understand it by generated model diagrams appropriate for the location. They include the following factors useful for parallel browsing both a reviewer's source code and its automatically-generated model diagrams:

- Model diagram update adapted for a reviewer's browsing location in her targeted source code,
- Complementary notation of model diagrams based on a reviewer's source code,
- Divisional association of a reviewer's source code with its generated model diagrams,
- Manifestation of system behaviors such as changing-over of a reviewer's browsing location,
- Browsing location handle in model diagrams,
- Review-time admeasurement for Passive Code Review.

3·1 Model Diagram Update for User's Browsing Location

This function helps a reviewer a lot to understand her targeted source code by updating model diagram adapted to her browsing location in the source code.

The Passive Code Review system enables a reviewer to perceive simplified correspondence relationship between her targeted source code and its generated flowchart diagram, by emphasizing where she should browse in the flowchart diagram of the whole source code as shown in Figure 2. The right is the current area of codes that a reviewer should browse in her targeted source code, while the left is the current of symbols that she should browse simultaneously in its generated flowchart diagram.

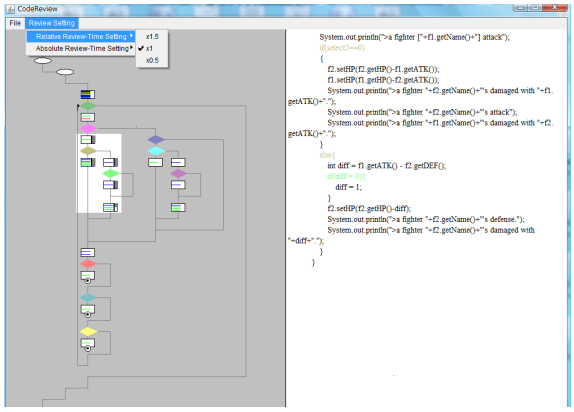

Fig. 2 An Implementation of Passive Code Review

3·2 Complementary Notation of Model Diagrams based on Source Code

In Passive Code Review, information that we can acquire from a model diagram is more abstract. This function provides a reviewer with more information from the generated model diagrams of her source code by complementing them with such abstracted information based on her source code. In our generated flowchart diagrams, process symbols are divided into three kinds of categories such as Input, Output, and General process based on their content of codes as shown in Figure 3, which are drawn by a red line, green line, and blue line respectively as shown in Figure 4. This categorization uses the following lists of codes ready beforehand: `readLine` as examples of Input process, and `System.out.print` and `System.out.println` as examples of Output process.

3·3 Divisional Association of Reviewer's Source Code with Model Diagrams

In Passive Code Review, a reviewer has to simultaneously browse a part of her source code on the left as well as its generated model diagram on the right. Therefore,

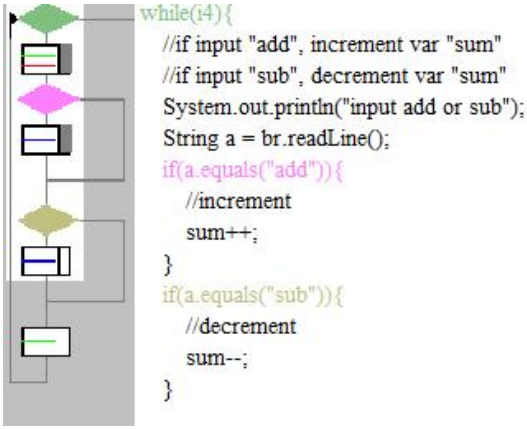

Fig. 3 A Magnification of Passive Code Review

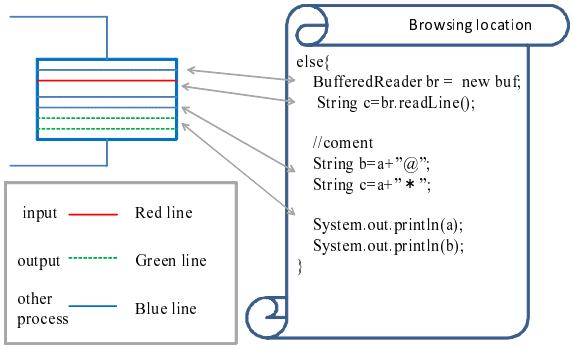

Fig. 4 Complementary Notation of Model Diagrams based on Source Code

this function needs to assist mutual complement of information by assigning the same display color to each statement of control flow such as `for`, `while`, and `if` in both her source code and its flowchart diagram and partially emphasizing information on screen as shown in Figure 5.

3·4 Manifestation of System Behaviors such as Changing-over of Reviewer's Browsing Location

Our Passive Code Review system needs to manifest some kind of timing for its abrupt system behaviors such as changing-over a reviewer's browsing location. Therefore, this function enables a generated flowchart diagram to manifest where the system is currently tracing and also expects a reviewer to browse in her source code, and information about the timing of changing-over her browsing location.

As shown in Figure 6, the right part of a process symbol (rectangle) has a black line on the left and a gray line on the right. The black line shows the codes that the system has already loaded from a reviewer's targeted source code, while the gray line shows where the system is now tracing and expects her to browse. When the gray line reaches the end of the black line, the system changes over her browsing location, expands the black line to the process symbols of newly-loaded codes, and zero-initializes the gray line.

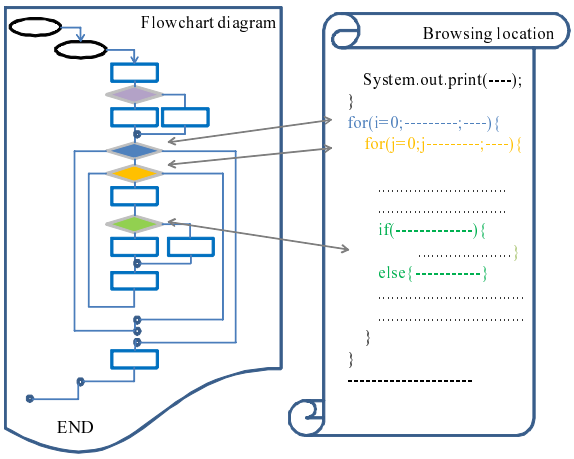

Fig. 5 Divisional Association of Reviewer's Source Code with Model Diagrams

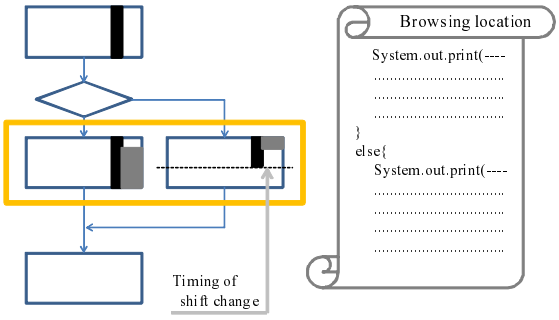

Fig. 6 Manifestation of System Behaviors such as Changing-over of Reviewer's Browsing Location

3·5 Browsing Location Handle in Model Diagrams

While code review combined with a model diagram, this function to handle a reviewer's browsing location in the model diagram enables her to choose her browsing location in her source code more freely. Our system has not only passive functions but also this active function to change over a reviewer's browsing location in her source code, adapted to her actively-clicked flowchart symbol in its generated flowchart diagram, as shown in Figure 7.

3·6 Review-Time Admeasurement for Passive Code Review

In Passive Code Review, this function to allow a reviewer to set her review time for her targeted source code beforehand, could disambiguate the finish time of the code review and resolve its crucial problem of time management. Our system provides the following two kinds of methods to set her review time as show in Figure 8: one method to set it relatively based on the standard review time by choosing from among multiplying factors (e.g., x1.5, x1, and x0.5), and another method to set it absolutely by choosing from the list of review time periods (e.g., 30sec, 1min, and 3min).

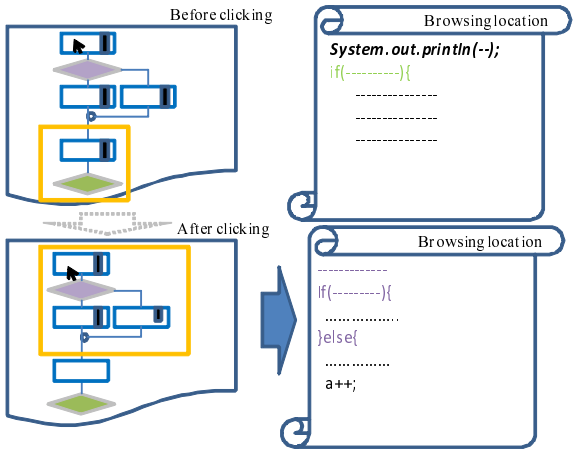

Fig. 7 Browsing Location Handle in Model Diagrams

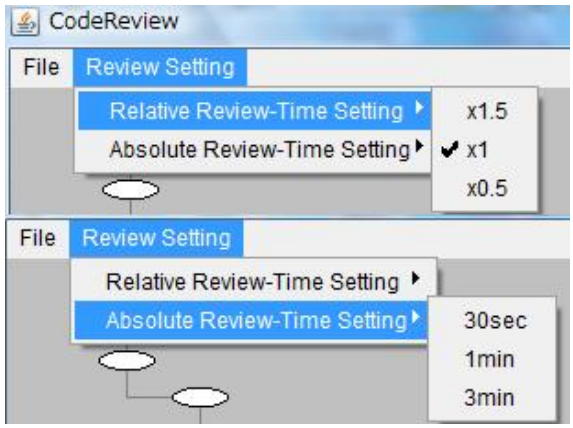

Fig. 8 Review-Time Admeasurement for Passive Code Review

4. Evaluation and Discussion

This section evaluates and discusses our proposed "Passive Code Review" system, by showing the results of its questionnaire surveys of 17 students of Computer Science.

4·1 Evaluation and Discussion of Implemented Functions

The usefulness of implemented functions into our proposed Passive Code Review system needs to be validated based on reviewers' subjective evaluations as an evaluative criterion. In questionnaire surveys, the reviewers (students) evaluated our system by five-grade evaluation (i.e., 1:not useful, 2:little useful, 3:slightly useful, 4:useful, 5:very useful) for each implemented function after they had used the system. Figures 9 to 14 show the results of the questionnaire surveys of the implemented functions. While their horizontal axis denotes the five-grade evaluation, their vertical axis denotes the count of reviewers (students) for each evaluation for each implemented function. And also Table 1 shows the average of evaluation for each implemented function.

Table 1 Average Evaluation for Each Implemented Function

Function	Average
Model Diagram Update for Reviewer's Browsing Location	4.00
Complementary Notation of Model Diagram based on Source Code	3.41
Divisional Association of Reviewer's Source Code with Model Diagrams	4.35
Manifestation of System Behaviors such as Changing-over of Reviewer's Browsing Location	3.29
Browsing Location Handle in Model Diagrams	4.24
Review-Time Admeasurement for Passive Code Review	3.12

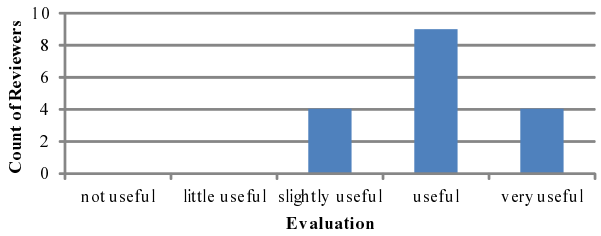

Fig. 9 Evaluation of "Model Diagram Update for Reviewer's Browsing Location"

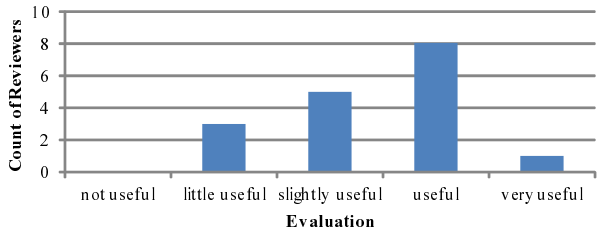

Fig. 10 Evaluation of "Complementary Notation of Model Diagram Based on Source Code"

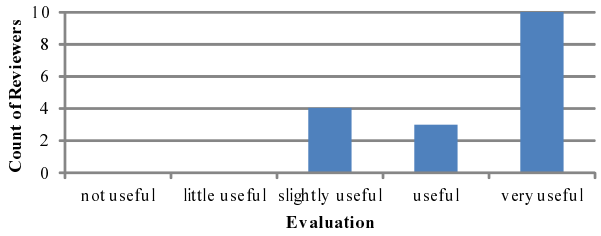

Fig. 11 Evaluation of "Divisional Association of Reviewer's Source Code with Model Diagrams"

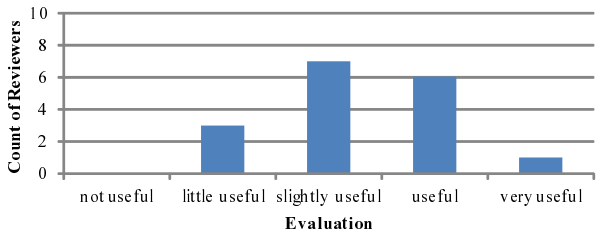

Fig. 12 Evaluation of "Manifestation of System Behaviors such as Changing-over of Reviewer's Browsing Location"

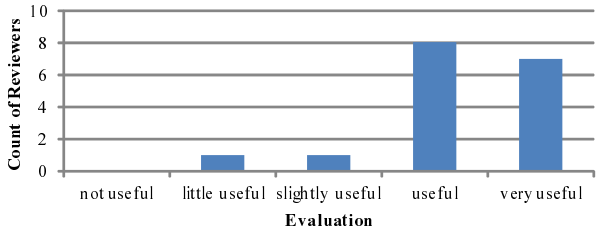

Fig. 13 Evaluation of "Browsing Location Handle in Model Diagram"

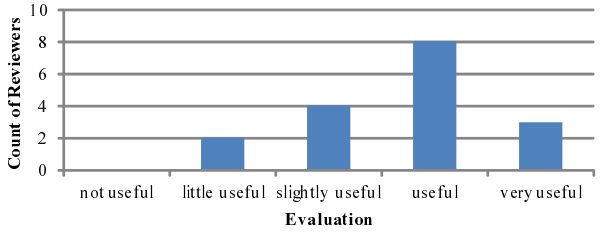

Fig. 14 Evaluation of "Review-Time Admeasurement for Passive Code Review"

In comparison with average evaluation of functions, the functions of "Divisional Association of Reviewer's Source Code with Model Diagram" and "Browsing Location Handle in Model Diagram" have high evaluation value. They seem to be useful functions for a reviewer to understand her browsing location by using both her targeted source code and its generated flowchart diagram.

In contrast, the function of "Complementary Notation of Model Diagram based on Source Code" does not seem to be evaluated as useful. This reason is because the function of a reviewer's partial complement is hard for her while simultaneously browsing both parts of her targeted source code and its generated flowchart diagram. In this case of the questionnaire surveys, the reviewers (students) did not fully understand the association of three categories of codes with three colors of lines for each process symbol. However, the usefulness of the function would also increase, when reviewers will use our proposed Passive Code Review system over and over again, and have its high literacy.

"Model Diagram Update for Reviewer's Browsing Location" is one of functions to associate a reviewer's browsing location based on the model diagram generated from her source code. But it seems to be not appropriate function of our generated flowchart diagram for Passive Code Review. In the questionnaire surveys, there is also a free-space comment that reviewers need a scaling function of flowchart diagram adapted to their browsing location.

As low evaluated functions, “Review-Time Admeasurement for Passive Code Review” is the worst, and “Manifestation of System Behaviors such as Changing-over Reviewer’s Browsing Location” is the second worst. This reason seems to be because both functions do not directly get involved with a reviewer’s browsing location in her targeted source code.

We have implemented six kinds of novel functions into the flowchart diagram generated from a reviewer’s targeted source code, and evaluated the usefulness of the novel model diagram. But it is hard not to feel that there are not only appropriate functions but also inappropriate functions for the implementation into flowchart diagrams. Therefore, in the future, we will try to implement the additional function that reviewers themselves can choose freely from among the implemented functions into model diagrams.

4·2 Usefulness Evaluation of Model Diagrams in Passive Code Review

Figure 15 shows the evaluative results on the usefulness of flowchart diagrams in our proposed Passive Code Review. It shows that most (15/17) reviewers gave them higher evaluation than or equal to 3 (slightly useful) and the average of evaluation is 3.71 (almost 4:useful). This is the comprehensive evaluation of all functions implemented into flowchart diagrams in this paper. Even if our Passive Code Review system adopts only flowchart diagrams as model diagrams referring to the reviewers’ comments, the future design of flowchart diagrams with context-aware display styles can expect to be more useful for reviewers because of its wide array of uses. And also model diagrams could assist code review because code review combined with model diagrams has received higher evaluation than without them as shown in Figure 16.

4·3 Comparison of Passive and Active Code Reviews

Figure 17 compares the evaluations of our proposed Passive Code Review and the existing Active Code Review after the reviewers used our system. Our current Passive Code Review system assisted by model diagrams is not enough useful compared with Active Code Review. However, our future system combined more appropriately with model diagrams would become enough useful, because several reviewers evaluate our current system as more useful and it is useful to assist code review by model diagrams regardless of whether the code review is passive or active.

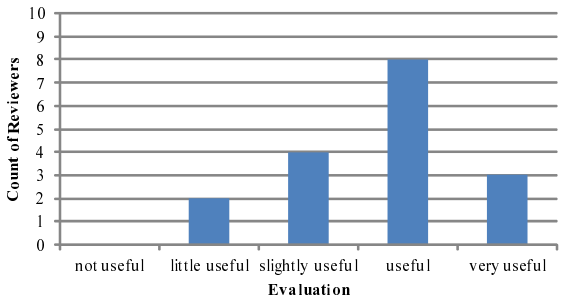

Fig. 15 Usefulness Evaluation of Flowchart Diagrams in Passive Code Review

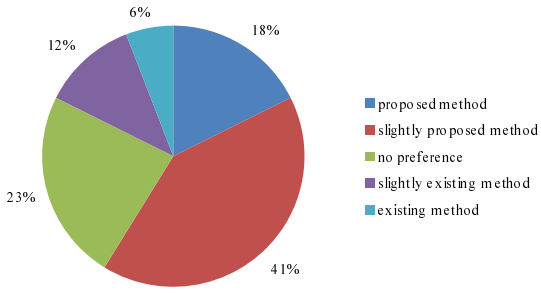

Fig. 16 Evaluation of Passive Code Review with vs. without Model Diagrams

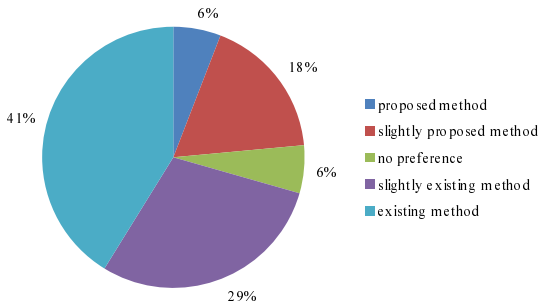

Fig. 17 Evaluation of Passive vs. Active Code Review

5. Proposal of Visual Support Functions

The most required function of Passive Code Review is “Divisional Association of Reviewer’s Source Code with Model Diagrams” as a result of the above-mentioned evaluation. And because there are some comments that it is a bigger burden to watch both different displays at once in the questionnaire surveys, our Passive Code Review remains to require “Divisional Association of Reviewer’s Source Code with Model Diagrams” and functions to support reviewers’ movement of observing point.

This section contrives visual support functions to make it easy to simultaneously browse parts of her targeted source code and its generated flowchart diagram, requests third parties to rank the functions, and determines our future direction of problems to be tackled as a result of the ranking.

5·1 List of Ideas for Visual Support Functions

Each of 10 ideas for visual support functions for our Passive Code Review system states a summary.

§1 Association Function for Colored Control Syntaxes

This function associates a branch control syntax (if-else) in a reviewer's targeted source code with its branch symbol (diamond) in the generated flowchart diagram in a same color (green) as shown in Figure 18.

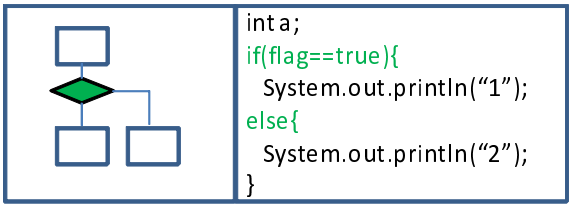

Fig. 18 #1: Association Function for Colored Control Syntaxes

§2 Association Function for Colored Sets of Process Syntaxes

This function associates a set of process syntaxes in a reviewer's targeted source code with its process symbol (rectangle) in the generated flowchart diagram by setting them in a same colored frame as shown in Figure 19.

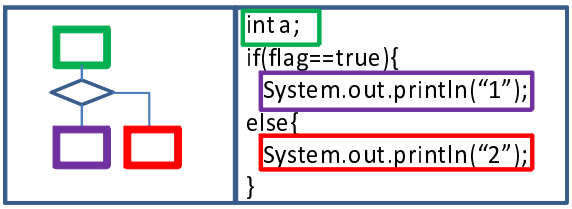

Fig. 19 #2: Association Function for Colorised Agminate Process Construction

§3 Association Function for Additional Effect to Colored Sets of Process Syntaxes

This function has a display style similar to "Association Function for Colored Sets of Process Syntaxes", and supports reviewers' view movement by color change (effect) and shift sequentially following the flow of source code.

§4 Association Function for Setting Colored Balls

This function associates a set of process syntaxes in a source code with its process symbol (rectangle) in the generated flowchart diagram by placing same colored balls at their quaternary corners as shown in Figure 20.

§5 Association Function for Labeling in Alphabet

This function assigns an alphabetical label to a set of process syntaxes and its corresponding process symbol along the flow of source code as shown in Figure 21.

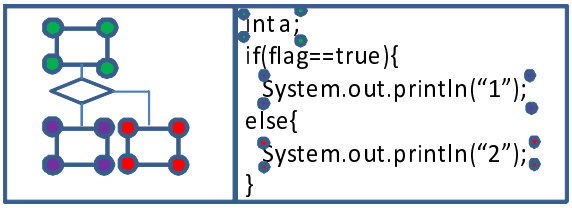

Fig. 20 #4: Association Function for Setting Colored Balls

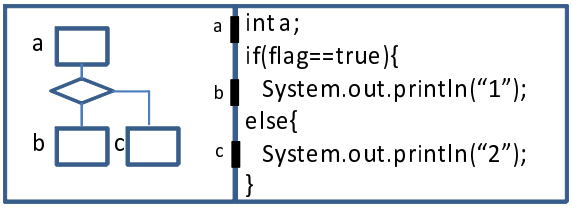

Fig. 21 #5: Association Function for Labeling in Alphabet

§6 Association Function for Drawing Lines

This function draws lines from a set of process syntaxes in a source code to its process symbol (rectangle) in the generated flowchart diagram as shown in Figure 22.

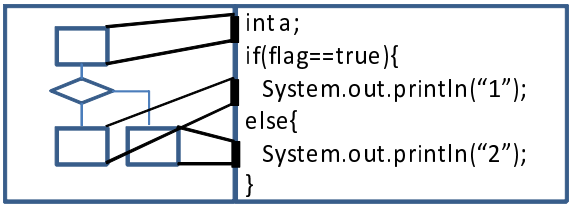

Fig. 22 #6: Association Function for Drawing Lines

§7 Association Function for Blinking Lines

This function has a display style similar to "Association Function for Drawing Lines", and blinks drawn lines to avoid the problem of frequent overlapping critical points of flowchart diagram.

§8 Association Function for Gauges

This function associates a set of process syntaxes in a source code with its process symbol (rectangle) in the generated flowchart diagram by setting a pair of temporally-changing gauges as shown in Figure 23.

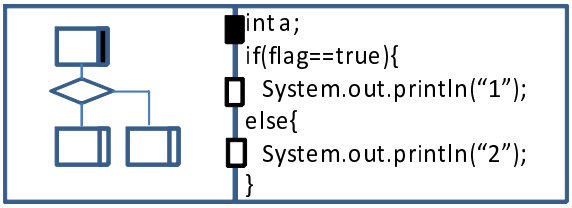

Fig. 23 #8: Association Function for Gauges

§9 Support Function for Reviewer's View Movement

This function supports a reviewer's view movement by setting a single colored ball of focal point to wander both her targeted source code and the generated flowchart diagram as shown in Figure 24.

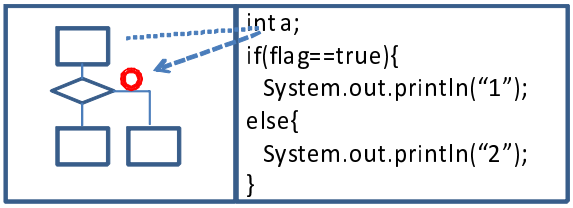

Fig. 24 #9: Support Function for Reviewer's View Movement

§10 Support Function for Reviewer's View Movement by Two Balls

This function supports a reviewer's view movement by setting two colored balls of focal point on each area of a source code and the generated flowchart diagram, and they move along the flow of source code as shown in Figure 25.

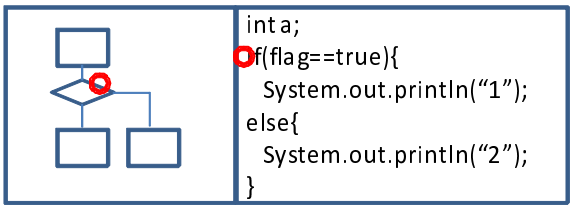

Fig. 25 #10: Support Function for Reviewer's View Movement by Two Balls

5·2 Questionnaire-based Evaluation of Proposed Visual Support Functions

This survey asks 11 undergraduate students studying in Information Technology to answer a questionnaire of comparing and ranking our proposed 10 kinds of visual support functions by their effectiveness, and tabulates and analyzes the ranking. Figure 26 shows the average of 11 ranks for each visual support function. The best function with the highest average is "#2: Association Function for Colored Sets of Process Syntaxes", while the worst function with the lowest average was "#10: Support Function for Reviewer's View Movement by Two Balls". And also "#9: Support Function for Reviewer's View Movement", which is a similar function to "#10: Support Function for Reviewer's View Movement by Two Balls", showed lower average. They (#9 & #10) were contrived to support reviewers' movement of observing point, but the questionnaire survey shows that functions to force reviewers' movement of observing point are not effective.

Based on the result of this additional questionnaire survey, visual support functions need to be designed for Passive Code Review to understand reviewers' intentions.

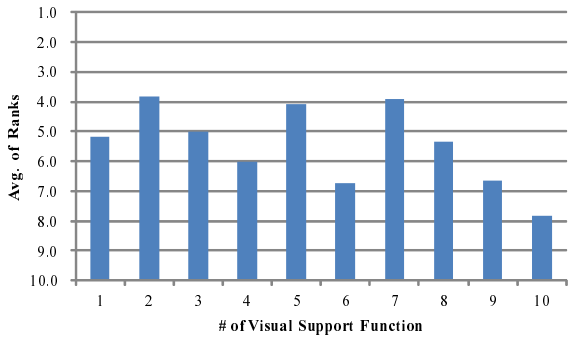

Fig. 26 Average of Ranks of Visual Support Functions

6. Conclusion

This paper has proposed "Passive Code Review" to review a targeted source code combined with its generated model diagrams, as a novel method of code review for ballooning source code at software development sites. We have proposed a novel method to give a reviewer high effects of code review within a limited amount of time, and described its basic approach, requirements, and basic architectonics. At the moment, we have already specified the basic architectonics to subserve code review by combining a flowchart diagram as a model diagram. By applying the basic architectonics to the other kinds of model diagrams, our proposed Passive Code Review system would contribute a great deal to large-scale code review without hearing from developers.

In the future, we try to increase the usefulness of Passive Code Review by combining not only flowchart diagrams but also various model diagrams. The first of our next challenges is to study how to program an expression of architectonics for class diagrams and an extraction of design patterns for sequence diagrams.

♢ References ♢

[Aksulu 10] Aksulu, A. and Wade, M.: A Comprehensive Review and Synthesis of Open Source Research, *Journal of the Association for Information Systems*, Vol. 11, pp. pp.576–656 (2010)

[Ferreire 10] Ferreire, A. L., Machado, R. J., Silva, J. G., Batista, R. F., Costa, L., and Paulk, M. C.: Proceedings of the 26th IEEE International Conference on Software Maintenance, *Machine Learning* (2010)

[Hourai 03] Hourai, H., Nadamoto, A., and Tanaka, K.: Passive Browsing of Multiple Web Pages by the Talk Show Metaphor, *IPSJ-DBS Technical Report*, Vol. 5, pp. pp.155–162 (2003)

[Laitenberger 02] Laitenberger, O.: A Survey of Software Inspection Technologies, *Handbook on Software Engineering and Knowledge Engineering*, Vol. 2, pp. pp.517–555 (2002)

[Remillard 05] Remillard, J.: Source Code Review Systems, *IEEE Software*, Vol. 1, pp. pp.74–77 (2005)