A Tool for Diagnosing the Quality of Java Program and a Method for its Effective Utilization in Education

HIROAKI HASHIURA†, SAEKO MATSUURA†, SEIICHI KOMIYA*
Graduate School of Engineering
Shibaura Institute of Technology
†307 Fukasaku, Minuma-ku, Saitama, 337-8570
*3-7-5 Toyosu, Koutou-ku, Tokyo, 135-8548
JAPAN
{hashiura@komiya.ise., matsuura@se., skomiya@}shibaura-it.ac.jp
†http://www.sayo.se.shibaura-it.ac.jp/

Abstract: In the software education imparted in universities, especially in the evaluation of beginners' practice assignments, functionality of software is emphasized, while evaluation of code quality such as adequateness of modular partitioning and readability of code usually takes the back seat. It is desired to feedback quality related evaluation to students from early stages of learning as the adverse effect of quality related problems comes to surface as the size of software increases. Although instructors understand this, since this requires investment of large amount of resources for development and maintenance of tools for measuring the quality of code, it has become a bottleneck resulting in the aforementioned situation. In the present study, we applied source code quality diagnostic tool, developed as an open source program, to the programs submitted by students. While describing the method for detecting typical mistakes which students are prone to commit based on the data obtained, we have described the method for effectively using highly general purpose source code quality diagnostic tool in education.

Key-Words: Software Engineering Education, Code Quality, Static Code Analysis

1 Introduction

In recent years, due to advancement of technology and diversification of system configurations, software programming projects undertaken in real world are increasingly becoming large scale and complex. Over and above that, development needs to be completed within short time period and at low cost. As a result, environment surrounding software development is increasingly becoming more demanding and challenging. Against such social backdrop, it has become important to cultivate human talent equipped with more practical knowledge and techniques of software development. However, companies in IT industry do not adequately recognize the urgency of cultivating such human talent; rather they are fully occupied with near term tasks. Software development projects are executed by putting fresh employees on live projects without adequate technical training and education, and by procuring developers from vendors. Stated differently, the fact is, software technicians are acquiring technical skills through actual projects rather than through in-house training and education. Therefore, it has become increasingly important for universities responsible for software engineering education to nurture human talent that can develop high quality software and not merely write lines of software code.

Requirements to be met by such resources include, complete acquaintance of fundamental knowledge and techniques of programming and software engineering and ability to use them in an integrated manner. However, acquiring these skills during 4 years of university education is not easy even for the students specializing in IT.

During the training of programming, students develop programs to achieve certain objectives; however, it is difficult for students as well as instructors to continuously pay attention to the quality of software code. Reason is main objective of such programming related trainings is to learn how to realize various functions required from programming using the programming language after understanding the structure of the programming language used by students.

In the present study, we introduced quality diagnostic tool for programs during programming education to achieve spontaneous improvements by
making the students and instructors aware about program quality from very early stages through auto-analysis of programs provided during training and giving the feedback of such quality related analysis to students. Moreover, by analyzing the data obtained in this manner, we have also discussed the method of detecting typical mistakes that students are prone to commit.

Finally, structure of the present paper is as follows. In Section 2, we have touched upon the issues of evaluating deliverables in software education and in Section 3; we have described the outline and configuration of quality diagnostic tool introduced in the present study. In Section 4, we have described a method of effectively applying this quality diagnostic tool in education, and experiment including its application and discussion of results has been covered in Section 5 and Section 6. At the end, in Section 7 we have presented a summary of this paper and touched upon topics for further research and investigation.

2 Current state of software engineering education and issues in evaluation of deliverables

As mentioned in Section 1, one of the approaches to nurture engineers with practical skills is a method called PBL (Project-Based Learning). In real world, usually software is developed by forming a project team. Given that, feature of PBL is, small projects are selected for education of software development conducted by dividing the team in several groups during exercise sessions, and due to that, practical software development allows students to have simulated experience. Effectiveness of such methods has been recognized since past and authors have also been using similar approach [1].

However, it can not be said that only PBL will result in adequate training of software engineering education. Matsuura[2] pointed out that it is necessary to consider the following points when conducting PBL.

- It is necessary to efficiently impart the knowledge required for a certain scale of software development in 3rd year of education (which is the main timing for conducting PBL).
- Although skills for proactively analyzing the problems by using the knowledge gained during lectures would be nurtured, it would be difficult to evaluate individual performance as activities would be done in group.

In several real life examples, we come across cases where PBL is conducted for somewhat complex projects of a certain scale. However, as pointed out by Matsuura, among the pre-required knowledge that should be acquired before the start of PBL, especially since the understanding of quality is not adequate, students end up developing low quality code, which hampers the successful completion of the exercise assignments. Moreover, since PBL is conducted in group, where code is shared by many people, even if the issues with the quality of code are pointed out, it would not necessarily mean that the knowledge and skills for software development would be acquired at the level of individual student.

Such problem is directly related to the problem of “when exactly to teach about quality of code?”. Students learning the software development for first time will start from beginner level programming and slowly start taking on difficult problems by using the knowledge gained during lectures.

Software quality characteristics are largely divided in 6 categories in ISO/IEC9126-1[3]. Although the evaluation of quality of exercises is done without getting biased by these categories, they are important from software engineering education point of view.

As described by Takeda et al. [4] in their study, there is a method for indirectly making the students aware of quality issues by changing the specifications in PBL; however, authors believe that it is necessary to teach about the program quality from functionality and usability point of view from the very fundamental exercises stage before conducting PBL, and point out trouble areas.

Some of the likely reasons for the quality evaluation to be biased towards functionalities are; (a) functionalities can be relatively easily evaluated by operating the program, (b) even in case of practice assignments; software is developed for a specific purpose. Program that does not meet its objective can not be practically used even the other quality characteristics may be quite excellent. In addition, in the initial stages of programming, emphasis is on learning the technology for achieving the objective, and as a result techniques are often simplified to clearly define the problem as well as objective, which accelerate this kind of trend.

On the other hand, evaluation regarding maintainability and efficiency tends to take a back seat. Damage caused by quality issues concerning these characteristics come to surface as the size and complexity increases. However, in the aforementioned situation, it is difficult for students to recognize quality
related issues on their own, and not only that, for instructors also it is difficult to teach such quality related issues due to problems mentioned above.

Moreover, merely operating the software will not do for evaluating these quality characteristics. Rather, it is essential to study the detailed content of the actual source code. Quality issues faced by each student are different. Lot of resources would be needed to instruct several students about such tasks, which has become a serious bottleneck for instructors.

Inoue et al. [5] have mentioned that to make students correct typical mistakes that occur during object oriented design process, it is sufficient to show the places where mistakes have occurred without actually indicating the correction method. Therefore, in the present study we have proposed to auto-analyze the source code of exercises, feedback the results of analysis to students and implement the environment that can be shared by all the students.

Introduction of such tool will not only prompt the students to develop high quality software on their own, but it will also allow the instructors to provide individual guidance to students about the problems that they can not correct on their own. Benefits to students as well as instructors are described below.

**Expected benefits to students**

- Can recognize and become aware of quality characteristics of their own software
- Since students can compare the quality characteristics of their own software and others’ software, even if there are no problematic areas in their own software, they can still check what exactly could be the problematic areas

**Expected benefits to professors**

- Help in identifying those students who are making mistakes
- Can be helpful in follow ups and adjusting the level of difficulty

### 3 Quality diagnostic tool for Java program

In this paper, we have described about Sonar [6], a quality diagnostic tool for source code used in the present study. Sonar is an open source tool developed largely by SonarSource SA of Switzerland, and it is a quality diagnostic tool mainly for Java programs.

#### 3.1 Structure of Sonar

Sonar can be largely classified in two parts. (1) Server part (Fig. 1) that works as a web application and that mainly handles visualization of reports, and (2) Client part that analyzes the code and generates report information. Overall structure of Sonar is shown in Fig. 2.

![Sonar Server](image)

**Figure 1 Sonar Server**

Client part is mounted as a plug in of Maven, which is an open source built tool, and different kind of quality information is acquired when source code is built. When acquiring quality information, Checkstyle, PMD, Findbugs etc called through Maven[7] and executed. After that, Sonar feedbacks these results of quality diagnostic tools to server side. Each of these quality diagnostic tools has specific features, and various reports can be generated through their combination. In addition, information is accumulated when generating the reports. This also allows
generating time-series data of quality information, which is another significant feature of Sonar.

3.2 Checkstyle[8]
Checkstyle is a static analysis tool mainly for checking superficial expressions in source code and it is developed as an open source program. It can also be checked whether programmer has developed Java code as per the coding standards or not.

Table 1 Checkrules of Checkstyle

<table>
<thead>
<tr>
<th>#</th>
<th>Category</th>
<th>#</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Annotations</td>
<td>9</td>
<td>Block checks</td>
</tr>
<tr>
<td>2</td>
<td>Javadoc comments</td>
<td>10</td>
<td>Coding</td>
</tr>
<tr>
<td>3</td>
<td>Naming Conventions</td>
<td>11</td>
<td>Class Design</td>
</tr>
<tr>
<td>4</td>
<td>Headers</td>
<td>12</td>
<td>Duplicate Code</td>
</tr>
<tr>
<td>5</td>
<td>Imports</td>
<td>13</td>
<td>Metrics</td>
</tr>
<tr>
<td>6</td>
<td>Size Violations</td>
<td>14</td>
<td>Regexp</td>
</tr>
<tr>
<td>7</td>
<td>White Space</td>
<td>15</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>8</td>
<td>Modifiers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Regarding check rules, total 132 types of check rules are available in the latest version 5.0. User needs to construct rule set according to coding guidelines. It is largely used in general projects, and it is bundled in the distribution package for rule set of Sun coding guidelines [9].

3.3 PMD[10]
Similar to aforementioned Checkstyle, PMD is also a static analysis tools. However, focus is on detecting the patterns in code that can easily generate bugs. It also differs from Checkstyle as it has detection rules depending on architecture and environment. With regard to its application, as in case of Checkstyle, rule set needs to be constructed in case of PMD as well.

3.4 FindBugs[11]
In contrast to aforementioned two tools, FindBugs is a tool for finding potential problems in the program by searching the byte code after compiling the code. Since it only focuses on detecting potential problems in bytecode, it can not detect problems in style or format of source code.

4 Effective method of application for education

As mentioned in Section 3, although Sonar is a very strong tool that can generate an integrated report by working in tandem with multiple open source static analysis tools, since it is developed for applying to usual software development project, it is essential to think how to modify the rule set when applying it for training and education purpose.

Three points need to be considered with regard to the definition of rule set. First of all, over total 700 types of check rules are available and if all of them are applied, it is easy to lose sight of priority ranking. Second point is, within the check rules, some of the rules are only valid for a specific operating environment and architecture, and since they may not be important from quality point of view, it is necessary to remove such rules. Third point is, although applicable conditions can be set for most of the rules (Fig. 3), one specific characteristic of static analysis tools is; if rules are overly stringent, overall number of detections will increase but false detections will increase as well. Because of this, it would be necessary to identify whether the problem detected is actually a problem or not. In case of a normal project, usually these tasks are handled by technical leader. However, since the target programs in our case are developed by students, it may be difficult to perform such tasks. Therefore, it is necessary to define rule set after understanding the level of students.

Figure 3 Example of setting applicable conditions of rule

With regard to the construction of rule set in this case, since the participants in the experiment possessed basic knowledge of object oriented approach and they could build GUI application by combining classes having multiple responsibilities, we selected applicable rules from 700 rules based on the following criteria.

- Rules where style is such that readability is adversely affected
- Unused rules
- Overly complex rules
If style is not appropriate and unused rules are mixed in the code, it will lower the readability of code thereby making it difficult for the instructor to read and understand the objective of code. Therefore, reason for setting these criteria is to let the instructor focus on higher level of problems by enabling the students to remove such problems on their own in advance. In addition, we decided that key task for instructors when focusing on higher level of problems would be to detect the overly complex segments of the program.

Complexity would be detected using cyclomatic number. Cyclomatic number is determined using the formula \(e - n + 2\), where \(e\) is number of branches and \(n\) is number of nodes when flow of program control is represented with digraph. Since it is equal to the number obtained by adding 1 to branches of control such as if, switch, for and while that exist in the program, more complex the control structure of program, higher the number.

In Sonar, level of rule violation can be classified in two categories, viz. Mandatory and Optional. Therefore items belonging to former were assigned to Mandatory and items belonging to later were assigned to Optional. Table 2 and Table 3 show the rule sets used in our experiment.

In addition, in order to avoid aforementioned false detection to the extent possible, with regard to error detection of naming convention of M-9, instead of default values of the tool, we customized such that the tool will detect inappropriateness if it falls under either of the following definitions.

**Definition of Inappropriate Name**
- When the name ends with a number
- When (items other than constant numbers) start with a capital letter

### Table 2 Mandatory Rule Set

<table>
<thead>
<tr>
<th>Type</th>
<th>Category of Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-1</td>
<td>Abbreviation of bracket “{” that shows block</td>
</tr>
<tr>
<td>M-2</td>
<td>Empty block</td>
</tr>
<tr>
<td>M-3</td>
<td>Substitution in formula</td>
</tr>
<tr>
<td>M-4</td>
<td>Inappropriate comparison</td>
</tr>
<tr>
<td>M-5</td>
<td>Hiding of fields</td>
</tr>
<tr>
<td>M-6</td>
<td>Use of magic number</td>
</tr>
<tr>
<td>M-7</td>
<td>Unused variables</td>
</tr>
<tr>
<td>M-8</td>
<td>Inappropriate visibility</td>
</tr>
<tr>
<td>M-9</td>
<td>Naming convention violations</td>
</tr>
<tr>
<td>M-10</td>
<td>Redundant expression</td>
</tr>
</tbody>
</table>

### Table 3 Optional ruleset

<table>
<thead>
<tr>
<th>Type</th>
<th>Category of Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Substitution in static field</td>
</tr>
<tr>
<td>0-2</td>
<td>Code having less than certain number of comments</td>
</tr>
<tr>
<td>0-3</td>
<td>Method having a lot of parameters</td>
</tr>
<tr>
<td>0-4</td>
<td>Substitution in parameter</td>
</tr>
<tr>
<td>0-5</td>
<td>Complex control structure</td>
</tr>
<tr>
<td>0-6</td>
<td>Complex conditional expressions</td>
</tr>
<tr>
<td>0-7</td>
<td>Use of redundant literals</td>
</tr>
<tr>
<td>0-8</td>
<td>Use of concrete class</td>
</tr>
</tbody>
</table>
Main objective of this is to detect variables simply created by appending numbers at the end or variables whose naming conventions do not follow lower camel-case. We did such customization as programming language was Java in our case; however, it is necessary to change such customization according to language selected during exercise.

5 Experiment
Rule sets described in the previous section were applied to the programs written by students.

Participants
- 50 Students from Programming Exercise II, Department of Electronic Information Systems, College of Systems Engineering and Science, Shibaura Institute of Technology

Method of conducting exercises
1. Students would write programs as per the instructions provided about problem
2. Groups with more than one student would be formed and program of each student would be reviewed. In the review sheet used for the review, apart from functionality, following items for quality assessment are also included.
   A) Naming (class, field, method)
   B) Variable scope
      (field, local, loop variable)
   C) Method size
   D) Control flag
   E) Magic number
   F) Class size
      (number of class, responsibility)
   G) Algorithm
   H) Main method
   I) Documentation comment
3. Program is corrected based on the review and final deliverables are prepared and presented.
4. Aforementioned (1)-(3) are repeated. This cycle is called “Theme”

For the exercises covered in experiment, exercise support system was already available and students would submit their assignments through this system. In order to analyze the source code with Sonar, it is necessary to re-configure the source code with the project of Maven. Therefore, a converter was developed that would convert the source code downloaded from support system to the Maven project in one go.

6 Experiment results and discussion
From the programs submitted during aforementioned exercise, we removed the programs that could not be complied correctly. We analyzed total 88,534 lines of program using Sonar. From the report generated by Sonar, we extracted information about places where 46 types of rules corresponding to Table 2 and Table 3 were violated.

For each theme, since submission deadline before review and after review was set, in order to check whether the quality is improving or not, we decided to check whether the number of violations had reduced before and after the review. However, in many cases programs were under development at the time of submission prior to the review, and in lot of cases the size of program had exponentially increased after the review. In order to minimize the impact due to change in size of programs, we compared the defect density for every 1000 lines of code excluding the comments before and after the review. Table 4 shows the rate of improvement in defect density. Although the number of persons in the table is significantly less than the total number of participants, this is mainly because programs submitted by many participants could not be compiled at the time of first submission deadline. Among items in Table 4, although “M-9 Naming convention violations” and “M-6 Use of magic number” are specifically included in the check items during review, improvement rates were only 54.9% and 50% respectively. From this it is clear that quality does not improve by conducting reviews only.

7 Summary and future work
In the present paper, we focused on the problem that evaluation of quality of source code is not done adequately in current software education, and described the method of constructing rule sets for effectively using tools in education. Moreover, we actually applied the rule sets created to actual exercises and discussed the results.

In this paper, although we only analyzed the results of the tool, in future we would directly input the feedback of quality reports to users and incorporate improvements in program so that program can clearly
point out inappropriate places. Moreover, in future we plan to develop a system so that students themselves can recognize improvements.

8 Acknowledgement

This work is supported by research grant “Program for Promoting University Education and Student Support (Theme A: Program for Promoting University Education Reform)” from the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan.

References:


