Continued Development of a Framework for the Analysis of Java Source Code Comments

Joseph A. Brigandi
Faculty Mentor: Dr. DePasquale
brigand2@tcnj.edu

ABSTRACT

With a framework for the analysis of Java [4] source code comments already in place, this paper details the continued development of a prototype software system, Comment Mentor, that will provide a method for professors and students to process code and receive detailed reports containing information about the documentation. The software is web-enabled, and contains a student login system integrated with a back-end database. The database is used to archive reports created by the system each time a student uses the comment analyzer. By accessing login information, professors have the capability to track their students' performance over time. Included in the paper are sample reports and a source code appendix of the software developed for this project.

1. BACKGROUND

The detailed use of documentation is one of the most overlooked ways of improving software quality and speeding implementation. However as students begin to program, they typically see commenting as a burden, often leaving it out completely. With these issues in mind, a software system was developed in the fall 2006 semester at TCNJ to allow students and professors to process code and receive detailed reports containing information about the documentation. The detailed reports allow the user to check whether comments exist, as well as run analysis on the quality and quantity of the documentation.

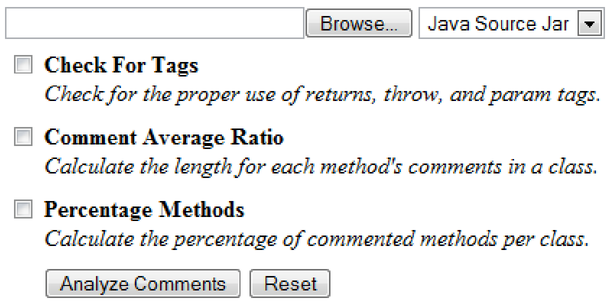
With the software system already in place, the next phase in development was to prepare Comment Mentor for full-time use. In order to achieve this, a few key goals were established. First, the system needed to be web-accessible. Web software is beneficial because it provides easy access to Comment Mentor for the students and professors. Secondly, the software needed a login and archival system. The login system accomplishes two extremely important ideas; it prevents access to Comment Mentor without a validated account as well as adding the capability to track the users' system usage. Once these goals were in place, the system was ready to use.

2. WEB-ENABLED FEATURES

The Comment Mentor software system was developed on the web using Hypertext Preprocessor (PHP) [8], HyperText Markup Language (HTML) [2], and Cascading Style Sheets (CSS) [1]. The basic interface of the system is a PHP form used to provide the user with functionality to upload source code and select preferences. The creation of the user's final report is in HTML. The HTML comes from a tool called Javadoc [5], which is used for generating application programming interface (API) documentation into HTML format. Javadoc is a computer software tool from Sun Microsystems, and is currently the industry standard for documenting Java classes. Once Javadoc creates the HTML, CSS is added to enhance the look and feel of the report.

2.1 Interface

Once a user is logged into Comment Mentor, they are presented with a form to upload code and select options to personalize the resulting report. The first step is to upload a jar file [3] containing Java source code. A jar is a ZIP file used to distribute a set of Java classes. This makes the process of uploading numerous files very easy. Next, the user is presented with a list of analyzers to choose from. Each item in the list has a checkbox so the user can pick any number of analyzers. Then, a text file is formulated with the list of selected analyzers. Lastly, once the user submits the form, the jar file is uploaded and the list of analyzers is processed into a text file for later use.



☐ **Check For Tags**
Check for the proper use of returns, throw, and param tags.

☐ **Comment Average Ratio**
Calculate the length for each method's comments in a class.

☐ **Percentage Methods**
Calculate the percentage of commented methods per class.

Figure 1. The Comment Mentor interface.

2.2 Creating the Report

Now that the user has selected their preferences and submitted the form, two scripts are used to process the data and display the final report. First, a PHP script (see Appendix A) is used to upload the jar file and complete errors checks. Next, the script calls a shell script (see Appendix B) to do the following: A temporary directory is created with a unique name used to store each user's source code and report. The uploaded jar file is moved to the temporary directory, and unzipped to the 'src' directory. Each temporary directory contains a 'src' directory to store the source code. Once the jar file is unzipped, a text file, doclets.txt, with the list of selected analyzers is created. Each item in the list analyzes the source code when it is run through the system. The final report is created and stored in the temporary directory.

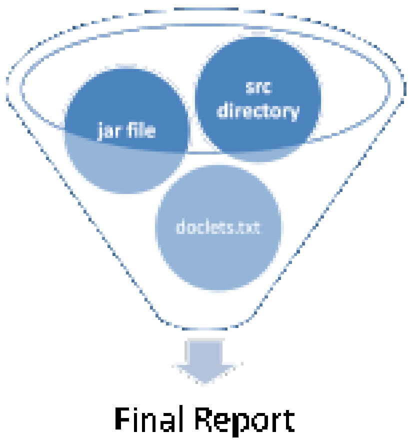


Figure 2. Contents of each temporary directory.

2.3 Display the Report

The final step of system is to display the generated report. The PHP script automatically redirects the user to the HTML report. From there, the user has the option of printing the report or saving the file as HTML. The report contains the results from the system analyzing the user's source code. A sample report is shown in Figure 7.

3. USER LOGIN SYSTEM

3.1 Basic Users

The user login system is made up of five main components. First, a user registers for an account, using their name, a valid email address, and school. In return, the user receives an email with an encrypted temporary password. The user is then able to login to the system using their email address and password. Once they are logged in, they can change the password to one of their choosing. Next, the user can take full advantage of Comment Mentor by analyzing their source code. After uploading a jar file [3], the user receives a detailed report of results in the form of an HTML file, which they can either save or print. Aside from this main use of Comment Mentor, users also have the option to view reports they've run in the past, as well as track their overall system usage. Lastly, the user can logout and end their session.

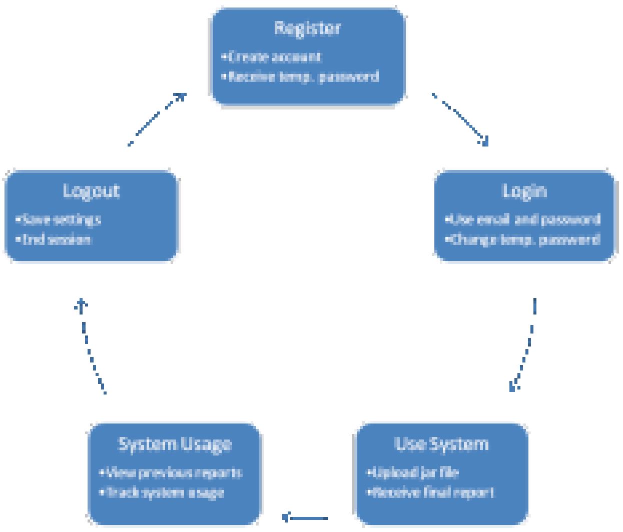


Figure 3. The user login system design.

3.2 Administrators

Although a majority of the accounts in the system will be basic users, or students, some accounts will have administrative privileges. When a user first logs in, the system checks the account type to decide what functionality is provided. If the user has a student account, they receive the previously mentioned basic functionality of the system. However, if the user has an admin account, they receive additional functionality.

Administrators can view the overall system usage for Comment Mentor. This includes the number of times each analyzer was used. More specifically, they can view the system usage for each individual user. Also, administrators can view all account information for each user such as name, email, school, etc. They also have the privileges to change users' account types or remove them from the system completely.

3.3 Functionality

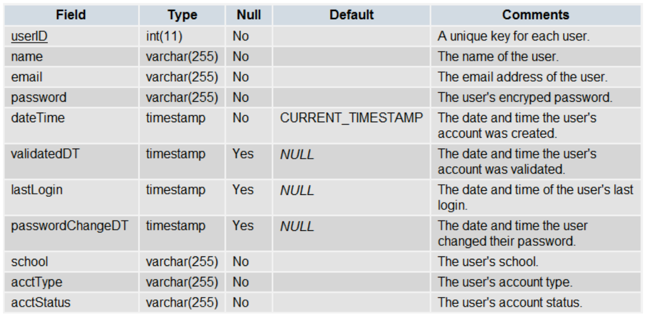
Functionality	Admin	Student
Change Password	√	√
Password Recovery	√	√
View Own Previous Reports	√	√
View All Previous Reports	√	
View Own System Usage	√	√
View Total System Usage	√	
Enable/Disable Accounts	√	
Request Account Deletion	√	√
Remove Accounts	√	
Change Permissions	√	

Figure 4. Admin & Student functionality table.

4. BACK-END DATABASE

4.1 Table Structure

A MySQL [7] database is used to archive data from the generated reports. Using phpMyAdmin, [9] three tables were created, a user table, a data table, and a report table. The user table contains a list of users registered in the system, as well important information about them. Key fields of the user table include, name, password, email, and school. The data table is made up of data pairs from the final report. Each data pair is stored as an entry in the table, along with the current date and time, as well as reportID and userID fields. The reportID field is determined by which analyzer the data pair is from and the userID field keeps track of which user's report the data is from. Finally, the report table is the list of analyzers available for the user to select from. The reportID field in the table is used to lookup the name of the analyzer.

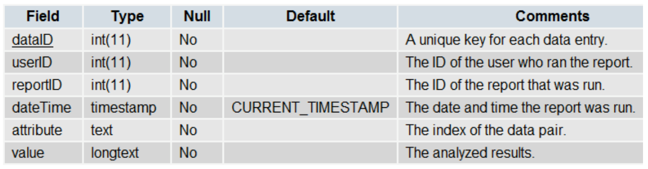


Field	Type	Null	Default	Comments
userID	int(11)	No		A unique key for each user.
name	varchar(255)	No		The name of the user.
email	varchar(255)	No		The email address of the user.
password	varchar(255)	No		The user's encryped password.
dateTime	timestamp	No	CURRENT_TIMESTAMP	The date and time the user's account was created.
validatedDT	timestamp	Yes	*NULL*	The date and time the user's account was validated.
lastLogin	timestamp	Yes	*NULL*	The date and time of the user's last login.
passwordChangeDT	timestamp	Yes	*NULL*	The date and time the user changed their password.
school	varchar(255)	No		The user's school.
acctType	varchar(255)	No		The user's account type.
acctStatus	varchar(255)	No		The user's account status.

Figure 5. The user database table.

4.2 Data Pairs

The key to effectively archiving reports and tracking the progress of students relies heavily on data pairs. Each report contains pairs of data that provide the user with feedback. Half of the data pair provides indexing capabilities, while the other half is the data itself. Each data pair is taken from the report and stored in the data table. Each pair also has a reportID and userID field to go along with it. The userID is the username and the reportID is the analyzer that the data pair came from. This method of storing data allows the system to track all generated reports for a given user, or specific analyzer.



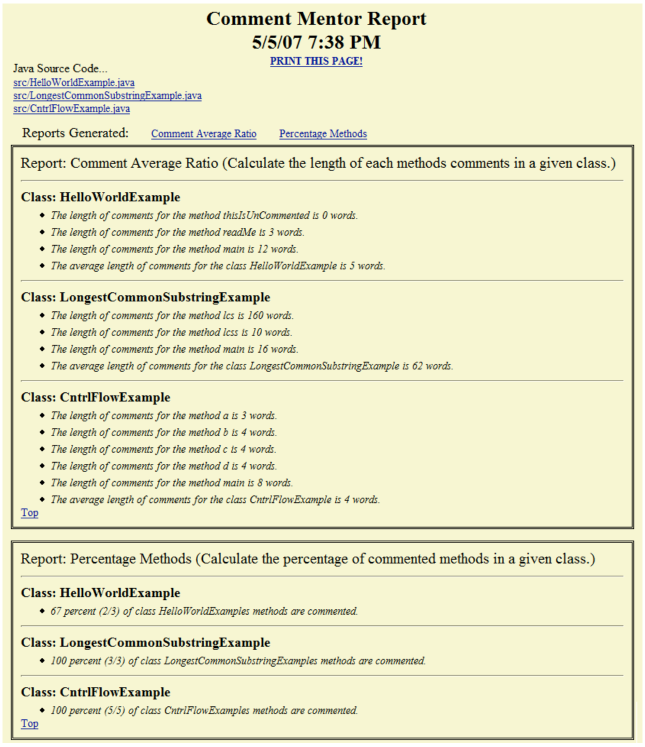
Field	Type	Null	Default	Comments
dataID	int(11)	No		A unique key for each data entry.
userID	int(11)	No		The ID of the user who ran the report.
reportID	int(11)	No		The ID of the report that was run.
dateTime	timestamp	No	CURRENT_TIMESTAMP	The date and time the report was run.
attribute	text	No		The index of the data pair.
value	longtext	No		The analyzed results.

Figure 6. The data table.

5. SAMPLE REPORT

The figure below is a sample report generated in a test run of Comment Mentor. Each report has the same basic formatting including the header, a list of original source code files, navigation links, and analyzed results. At the very top, the header contains the date and time of the report, as well as a link to print the page. Below the header is a list of the original source code that was submitted in the jar file. Each source file is a link that expands the source code underneath. This trick uses CSS to toggle the visibility of the source code. At first, all the source files are hidden, but when one of the links is clicked, the source for that file is displayed directly below it. By clicking on the link again, the source code will disappear and look like the original report. Just below the source code is a section called reports generated. These are anchor links that can take the user straight to the report they want to see.

The main section of the report is the analyzers, where all the data is found. Each analyzer selected has a double-bordered box that contains the name and description of the report, as well as all the results. In the example below, the Percentage Methods and Comment Average Ratio analyzers were selected. Each box consists of the classes that were analyzed, with the corresponding results as bullets underneath. This format for displaying the results makes the report easy to read and understand.



Comment Mentor Report
5/5/07 7:38 PM
PRINT THIS PAGE!

Java Source Code...
src/HelloWorldExample.java
src/LongestCommonSubstringExample.java
src/CntrlFlowExample.java

Reports Generated: Comment Average Ratio Percentage Methods

Report: Comment Average Ratio (Calculate the length of each methods comments in a given class.)

Class: HelloWorldExample
- *The length of comments for the method thisIsUnCommented is 0 words.*
- *The length of comments for the method readMe is 3 words.*
- *The length of comments for the method main is 12 words.*
- *The average length of comments for the class HelloWorldExample is 5 words.*

Class: LongestCommonSubstringExample
- *The length of comments for the method lcs is 160 words.*
- *The length of comments for the method lcss is 10 words.*
- *The length of comments for the method main is 16 words.*
- *The average length of comments for the class LongestCommonSubstringExample is 62 words.*

Class: CntrlFlowExample
- *The length of comments for the method a is 3 words.*
- *The length of comments for the method b is 4 words.*
- *The length of comments for the method c is 4 words.*
- *The length of comments for the method d is 4 words.*
- *The length of comments for the method main is 8 words.*
- *The average length of comments for the class CntrlFlowExample is 4 words.*

Top

Report: Percentage Methods (Calculate the percentage of commented methods in a given class.)

Class: HelloWorldExample
- *67 percent (2/3) of class HelloWorldExamples methods are commented.*

Class: LongestCommonSubstringExample
- *100 percent (3/3) of class LongestCommonSubstringExamples methods are commented.*

Class: CntrlFlowExample
- *100 percent (5/5) of class CntrlFlowExamples methods are commented.*

Top

Figure 7. Sample report.

6. FUTURE WORK

With a framework for Comment Mentor now in place, the ability to enhance the system has endless possibilities. First and foremost, more analyzers can be developed and added to the system. Each analyzer provides greater detail for the user to choose from. This alone can greatly improve Comment Mentor.

On top of adding more analyzers, improving compatibility is at the top of the list for potential growth. Increased compatibility will allow Comment Mentor to be used on more platforms and more browsers. Currently, the system has been tested on Windows and Mac OS X using several different browsers. These browsers include

Mozilla Firefox, Microsoft's Internet Explorer, Opera, and Safari.

Another potential add-on that would drastically enhance the system is adding more programming languages. Currently Comment Mentor only supports Java source files. However, the ability to analyze source code from languages such as C, Pascal, and Bash would be a sign of great progress for the system.

Yet another prospective addition to the system would be increased security. With the increased number of web applications in use today, web security has become more important than ever. Using a book such as "How to Break Web Software" by James A. Whittaker can greatly improve the system's security and help prevent potential problems.

Potentially the biggest upcoming add-on to Comment Mentor is a student and professor system. Currently the system only supports basic users and administrative accounts. However, the addition of a student and professor relationship would drastically enhance Comment Mentor. This would allow students to sign up for an account and become part of professors' classes. In return, the professor would become the admin for their classes and could track their students throughout the semester. Professors could require students to submit all their code to Comment Mentor and use it help grade the projects.

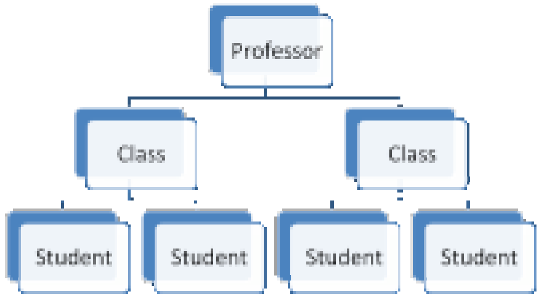


Figure 8. Student & professor relationship.

As this figure shows, students would get assigned to classes that the professor controls. When a student signs up under a professor, he or she would have to verify that the student is in the correct class. This relationship between students and professors could become even more involved with class sections, specific assignments, and much more.

7. REFERENCES

[1] Cascading Style Sheets Home Page. 2007. http://www.w3.org/Style/CSS/

[2] HyperText Markup Language (HTML) Home Page. 2007. http://www.w3.org/MarkUp/

[3] JAR File Specification. 2007. http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html

[4] Java Technology. 2007. http://java.sun.com

[5] Javadoc Tool Home Page. 2007. http://java.sun.com/j2se/javadoc/

[6] Javascript: The Definitive Javascript Resource. 2007. http://www.javascript.com/

[7] MySQL: The world's most popular open source database. 2007. http://www.mysql.com/

[8] PHP: Hypertext Preprocessor. 2007. http://www.php.net/

[9] phpMyAdmin: MySQL Database Admin Tool. 2007. http://www.phpmyadmin.net/

Appendix A
Run.php

```php
<?
//set ID of current user
session_start();
$userID = $_SESSION['userID'];

//directory
include("directory.php");

//check to make sure the file is present
if(isset($_FILES['file']) == false OR $_FILES['file']['error'] == UPLOAD_ERR_NO_FILE){
       echo "You need to upload a jar file!";
}

//check file extension
function file_extension($filename)
{
    return end(explode(".", $filename));
}

$ext = file_extension($filename);
if($ext != jar){
       echo "You need to upload a jar file!";
}

//check to make sure at least one doclet was selected
if(isset($_POST['doclet']) == false){
       echo "You need to select at least one analyzer!";
}

//the tmp_name begins with '/tmp/' so only grab the remaining substring
$tempFileName = substr($_FILES['file']['tmp_name'], 5);

//upload the jar file
move_uploaded_file($_FILES['file']['tmp_name'], $dir . $fileName);

//generate list of doclets selected (Doclets.txt)
$myFile = $dir . $tempFileName . ".txt";

$fh = fopen($myFile, 'a');
if(isset($_POST['doclet'])){
       $doclet = $_POST['doclet'];
       $count = count($doclet);
       $i = 0;

       while ($i < $count){
              $nextDoclet = $doclet[$i] . "\n";
              fwrite($fh, $nextDoclet);
              $i++;
       }
}
fclose($fh);

//starts a script to run javadoc
exec($dir . "scripts/javadoc.sh " . $tempFileName . " " . $fileName . " " . $userID);

//display report
header('Content-Type: text/html; charset=ISO-8859-1');
readfile($dir . $tempFileName . '/ComtorReport.php');
?>
```

Appendix B
Javadoc.sh

```
#!/bin/sh

PATH='/home/brigand2/public_html'

#dir of class files for javadoc
CLASSES='/home/brigand2/code/classes'

#create temp dir based on tempFileName
cd $PATH
/bin/mkdir $1
/bin/chmod 755 $1
cd $1

#create src dir for all source files to be stored within the temp dir
/bin/mkdir src
/bin/chmod 755 src

#move jar file from home dir to src
cd $PATH
/bin/mv $2 $1/src

#move text file (list of doclets) to the temp dir and rename it to Doclets.txt
/bin/mv $1.txt $1
cd $1
/bin/cp $1.txt Doclets.txt
/bin/rm $1.txt

#unjar the jar file in src dir and set the permissions
cd src
/etc/java/jdk1.6.0/bin/jar xf $2
cd ..
/bin/chmod 755 -R src

#move the jar file from the src dir back to the temp dir
cd src
/bin/mv $2 ../$2

#find the list of java files within the src dir and store the list in source.txt
cd $PATH/$1/
/usr/bin/find src -name *.java > source.txt
/bin/chmod 755 *.txt

#take the contents of source.txt and store it in MYVAR
MYVAR=`/bin/cat source.txt`

#make text file with userID
/bin/cat >> userID.txt << EOF
$3
EOF

#run javadoc
/etc/java/jdk1.6.0/bin/javadoc -doclet comtor.ComtorDriver –docletpath $CLASSES $MYVAR

#run Comtor Report
cd $PATH/$1
/usr/bin/php -f ComtorReport.php
```

Appendix C
File Listing

```
|-- img
     |-- comtor.png (Comtor logo)
     |-- tcnj_logo-small.gif (TCNJ logo)
|-- scripts
     |-- javadoc.sh (Appendix B)
|-- templates
     |-- template.dwt.php
|-- about.php
|-- adminReports.php (admin functionality)
|-- changeAcctType.php
|-- changePassword.php
|-- changePasswordForm.php
|-- connect.php (connect to MySQL database)
|-- deleteAccount.php
|-- directory.php (working directory of Comtor)
|-- disableAccount.php
|-- doclets.txt (list of analyzers)
|-- enableAccount.php
|-- faq.php
|-- features.php
|-- index.php (homepage)
|-- login.php (validates login)
|-- loginForm.php
|-- logout.php
|-- manageAccounts.php (admin functionality)
|-- recoverPassword.php
|-- recoverPasswordForm.php
|-- redirect.php (redirects to homepage)
|-- register.php (create account)
|-- registerForm.php
|-- reports.php (view previously run reports)
|-- run.php (Appendix A)
```