CODE INSPECTIONS: A WEB CRAWLER EXERCISE FOR STUDENTS*

Steena Monteiro, Renee Bryce
Department of Computer Science
Utah State University
Logan, UT
Renee.Bryce@usu.edu

ABSTRACT
It is well known that peer reviews are often effective in finding software bugs. This paper provides a code inspection exercise in which students work in teams to identify problems in code for a web crawler. The code is intentionally poorly written and the code used to work until the website that it visits changed its format! Students must identify poor design and documentation issues in the code. We gave this assignment to student teams in two separate semesters and found that all students identified design and documentation issues. Students were then given an assignment to either modify the existing code or to rewrite a new web crawler from scratch. Even though the modifications would have been relatively small, every team chose to rewrite the code from scratch and cited poor design and documentation issues for this decision. Our in-class discussion of the exercise revealed that the exercise illuminated the importance of good documentation.

1. INTRODUCTION
Code inspections are a common practice in professional software development teams, but can also be useful to students. Early work by Meyers found that code walkthroughs with pairs of subjects that independently reviewed code and then pooled their findings was the most effective form of testing in their experiments [6]. In the classroom, Trytten found that code reviews not only helped students with better understanding of programming and code comprehension, but also assisted in team

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.
building [7]. Hundhausen et al. saw an improvement in the quality of design and implementation when they integrated a formal code review process into a lower level Computer Science course [3]. Turner et al. helped to accelerate the learning of new programming concepts such as object-oriented programming by using peer reviews [8]. Clark et al. using peer testing which resulted in positive collaborations among students and increased student interest in software testing [1]. Of course, a large body of work on pair programming exists in which students work together on code and consequently review each other's code. Pair programming students often produce better work in less time, have more confidence, and enjoy the experience more than working alone [4,5,9]. Of course, personality is an issue in the ability to review code [2]. Our work differs from previous work in that students do not review their own code, but rather review intentionally poor code that the instructor created for the exercise and the code that we use is a web crawler which is a topic that is outside of the scope of our curriculum, yet attainable for our students to understand.

2. CODE INSPECTION EXERCISE

The exercise is conducted as follows: First, students individually review the web crawler code and strive to identify at least 10 problems with the code. Second, the students meet in class to hold a peer review of the code. During this session, each team has a team leader and a recorder. The team leader makes sure that they review each page and that every team member has the opportunity to speak. The recorder keeps a log of each problem in the code. Finally, the students meet outside of class to prepare and type their report. The reports include: a list of the team members and their roles, a list of the total number of defects found and a description of each fault for each team member, a summary of the team recommendations to improve the code, and a summary of the effectiveness of each reviewer. The summary of effectiveness includes: the amount of time that each student spent reviewing the code and the number of defects that each student identified.

3. CODE TO REVIEW

3.1 Problem Statement

The following code is a web crawler that visits CNN (http://www.cnn.com) to pull the headlines for my favorite four categories – Technology, Health, Science & Space, and Education. The code was originally written by Jack Hacker who is famous for writing code quickly (although some complain that his code is not very easy to read). Well, this code worked fine until recently – CNN changed the format of their website and the web crawler stopped working! Even worse, Jack Hacker is nowhere to be found! Rose, a local software engineer, was assigned to fix the code. She is known for her good design and coding skills. While she is usually very rational, she has decided that it is not worth fixing Jack Hacker's code below. She'd rather rewrite it from scratch! Your first task is to identify any 10 problems with this code that make it bad code. You will then either modify the code or write your own code to work with the new format on the CNN website.
3.2 Code

The code assigned to the students is as follows:

```java
import gnu.regexp.*;
import java.net.*;
import java.io.*;
public class cnnCrawler{
    public static void main(String[] args)
    {
        StringBuffer basePage = new StringBuffer();
        // Connect to CNN and get the document
        basePage = getBasePageContents("http://www.cnn.com");

        // Look at the area of interest (The "MORE FROM CNN" section)
        basePage = initialIsolateBasePageContents(basePage);

        // Pull all of the URLs out
        basePage = getInfo(basePage, "&nbsp;\&lt;a href="\[^"\]*|/b&gt;\&lt;a href="\[^\]*|/b&gt;&lt;a href="\[^\]*|/
        basePage = getInfo(basePage, "\"/[^\(\"]*\"");
        basePage = getInfo(basePage,"\"[^&\]*");

        // Go to the URLs and pull out the information of interest and
        // write to file.
        goToURLs(basePage);
    }
}
```

// Method: getBasePageContents
// This method opens a connection to the webpage we are interested in and stores all of
// the text on the page
public static StringBuffer getBasePageContents(String myURL){
    try{
        // Set base document to CNN, open connection,
        // and copy the source text into a buffer
        URL cnnBaseDoc = new URL(myURL);
        cnnBaseDoc.openConnection();
        BufferedReader cnnBaseBuffer = new BufferedReader(new InputStreamReader(cnnBaseDoc.openStream()));
        String cnnBaseInputLine;
```
StringBuffer tempDocument = new StringBuffer();
while ((cnnBaseInputLine = cnnBaseBuffer.readLine()) != null) {
    tempDocument.append(cnnBaseInputLine);
}
cnnBaseBuffer.close();
return(tempDocument);
}
catch(MalformedURLException e) {
    System.out.println("Unable to create URL object");
    return(null);
}
catch(IOException e){
    System.out.println("Unable to open URL");
    return(null);
}
}

// Method: initialIsolateBasePageContents
// This method isolates us to store only the section we are interest in -- the "MORE FROM CNN" section
public static StringBuffer initialIsolateBasePageContents(StringBuffer basePage) {
    try{
        RE document = new RE(basePage);
        // Define the left and right isolators
        String sLeft = new String("MORE FROM CNN[/w//W]*");
        RE leftCntxt = new RE(sLeft);
        RE rightCntxt= new RE("><b>SPORTS");
        StringBuffer sLIsolator = new StringBuffer("\n");
        int iLIsolatorIndex = 0;
        RE regLIsolator = new RE(leftCntxt);
        REMatch ctxtLMatch = regLIsolator.getMatch(basePage);
        sLIsolator.append(ctxtLMatch.toString());
        iLIsolatorIndex = ctxtLMatch.getStartIndex();

        // Find the Right Isolator
        StringBuffer sRIsolator = new StringBuffer();
        RE regRIsolator = new RE(rightCntxt);
        int iRIsolatorIndex = 0;
        REMatch ctxtRMatch = regRIsolator.getMatch(basePage);
        sRIsolator.append(ctxtRMatch.toString());
        iRIsolatorIndex = ctxtRMatch.getStartIndex();
        basePage.delete(iRIsolatorIndex, basePage.length());
        basePage.delete(0, iLIsolatorIndex);
        return(basePage);
    } catch(MalformedURLException e) {
        System.out.println("Unable to create URL object");
        return(null);
    } catch(IOException e){
        System.out.println("Unable to open URL");
        return(null);
    }
}
catch(REException e){
    System.out.println("RE Exception");
    return(null);
}

public static StringBuffer getInfo(StringBuffer textToSearch, String regExp){
    try{
        StringBuffer sIsolated = new StringBuffer(""");
        int iLIsolatorIndex = 0;
        String sLeft = new String(regExp);
        RE leftCntxt = new RE(sLeft);
        RE regLIsolator = new RE(leftCntxt);
        REMatchEnumeration ctxtLMatch = regLIsolator.getMatchEnumeration(textToSearch);
        while (ctxtLMatch.hasMoreMatches()){
            sIsolated.append(ctxtLMatch.nextMatch().toString());
            sIsolated.append("\n");
        }
        return(sIsolated);
    }catch(REException e){
        System.out.println("RE Exception");
        return(null);
    }
}

public static void goToURLs(StringBuffer textToSearch){
    try{
        StringBuffer interestingDoc = new StringBuffer(""");
        StringBuffer sInfoForFile = new StringBuffer(""");
        int numPage=0;
        FileOutputStream fCnnOut;
        PrintStream pCnnOut;
        String sLeft = new String(\"/[\\"]*\"); 
        RE leftCntxt = new RE(sLeft);
        String sIsolated = new String();
        int iLIsolatorIndex = 0;
        RE regLIsolator = new RE(leftCntxt);
        REMatchEnumeration ctxtLMatch = regLIsolator.getMatchEnumeration(textToSearch);
fCnnOut = new FileOutputStream("cnnCrawlerOutput.txt");
pCnnOut = new PrintStream(fCnnOut);

while (ctxtLMatch.hasMoreMatches())
{
    numPage++;
    sIsolated = "http://www.cnn.com";
    sIsolated += (ctxtLMatch.nextMatch().toString());
    interestingDoc = connectToURLs(sIsolated);
    sInfoForFile = getDocInfo(interestingDoc, sIsolated, numPage);
    pCnnOut.println(sInfoForFile);
}
pCnnOut.close();
System.out.println("You may view the output in file:
cnnCrawlerOutput.txt.");

} catch(REException e){
    System.out.println("RE Exception");
}
catch (Exception e)
{
    System.out.println("Error writing file.");
}

/**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
// Method: connectToURLs
// This method opens a URL and returns the text of the page
~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
public static StringBuffer connectToURLs(String urlText){
    try{
        URL cnnBaseDoc = new URL(urlText);
        cnnBaseDoc.openConnection();
        BufferedReader cnnBaseBuffer = new BufferedReader(new
            InputStreamReader(cnnBaseDoc.openStream()));
        String cnnBaseInputLine;
        StringBuffer tempDocument = new StringBuffer();
        while ((cnnBaseInputLine = cnnBaseBuffer.readLine()) != null){
            tempDocument.append(cnnBaseInputLine);
        }
        cnnBaseBuffer.close();
        return(tempDocument);
catch(MalformedURLException e) {
    System.out.println("Unable to create URL object");
    return(null);
}

} catch(IOException e){
    System.out.println("Unable to open URL");
    return(null);
}

public static StringBuffer getDocInfo(StringBuffer doc, String URL, int ID){
    StringBuffer importantInfoToReturn = new StringBuffer("");
    StringBuffer Headline = new StringBuffer("");
    StringBuffer Date = new StringBuffer("");
    StringBuffer Place = new StringBuffer("");
    StringBuffer FirstParagraph = new StringBuffer("");
    URL = URL.substring(0, (URL.length()-1));
    Date.append(getInfo(doc, "name="DATE" content="\[^>]*"));
    if(Date.length() > 0){
        Date.delete(0,21);
        Date.delete((Date.length()-1), Date.length());
    } else{
        Date.append("No date Reported.");
    }

    Place.append(getInfo(doc, "<p><b>[^(<p>)]*<p><b>[^->]*"));
    if(Place.length() > 0){
        Place.delete(0,6);
    } else{
        Place.append("No location Reported.");
    }

    Headline.append(getInfo(doc, "<title>CNN.com - [^->]*"));
    if(Headline.length() > 0){
        Headline.delete(0,17);
        Headline.delete((Headline.length()-1), Headline.length());
    }
}
else{
    Headline.append("No headline Reported.");
}
FirstParagraph.append(getInfo(doc, "DESCRIPTION\" content=[^>]*");
if(FirstParagraph.length() > 0){
    FirstParagraph.delete(0, 22);
    FirstParagraph.delete(FirstParagraph.length()-1, FirstParagraph.length());
}
importantInfoToReturn.append("\n");
importantInfoToReturn.append((ID + " | "));
importantInfoToReturn.append((Headline + " | "));
importantInfoToReturn.append((URL + " | "));
importantInfoToReturn.append((Date + " | "));
importantInfoToReturn.append((Place + " | "));
importantInfoToReturn.append((FirstParagraph));
return(importantInfoToReturn);
}

4. RESULTS

4.1 Review Time and Defects Found

Students reported the amount of time that they spent reviewing the code and a list of the defects that they identified.

Tables 1-(a-e) provide the results for each team. Table 1-(f) summarizes these results. Most students spent one hour reviewing the code as we recommended. One student reported that they spent less time. This student also reported that they found only 6 bugs whereas the average student spent 78.75 minutes and found 8.58 bugs. A total of 8 students spent more than one hour reviewing the code and found 9 to 10 bugs each. The average defects found per hour ranged from 2.5 to 10, but the average of 7.25 and median of 7.75 were close. We review the types of defects that they found next.

<table>
<thead>
<tr>
<th>Reviewer</th>
<th>Review Time</th>
<th>Total Defects</th>
<th>Defects per hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45</td>
<td>6</td>
<td>8.00</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
<td>10</td>
<td>5.00</td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>10</td>
<td>5.00</td>
</tr>
<tr>
<td>4</td>
<td>240</td>
<td>10</td>
<td>2.50</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>10</td>
<td>7.50</td>
</tr>
<tr>
<td>Average</td>
<td>121</td>
<td>9.2</td>
<td>5.60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reviewer</th>
<th>Review Time</th>
<th>Total Defects</th>
<th>Defects per hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60</td>
<td>9</td>
<td>9.00</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>8</td>
<td>8.00</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>9</td>
<td>9.00</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>7</td>
<td>7.00</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>10</td>
<td>10.00</td>
</tr>
<tr>
<td>Average</td>
<td>60</td>
<td>8.6</td>
<td>8.60</td>
</tr>
</tbody>
</table>

(a) (b)
4.2 Student reported defects

Each student submitted their individual list of defects. Due to space, we compiled a list of all of the unique types defects that were reported by the 24 students:

1. Many comments are missing or are not informative enough
2. The indentation style is not consistent and makes it more difficult to read the code.
3. Some variables look like class names.
4. Some comments are misleading, i.e., one example is that connectToURLs only connects to one URL.
5. Many methods are too large and should be broken into smaller methods.
6. Some variables are never used.
7. Perhaps regular expressions should be assigned to strings to improve the readability.
8. Variables are reused and the meaning changes.
9. Many exceptions are not caught.
10. The code is inefficient in that it grabs all page content and then discards what is not needed.
11. Hard coding should be avoided, i.e., regular expressions and file names are hard coded.

12. A variable assignment in a loop should be moved to come before the loop

None of the students found all of the defects that we anticipated that they would found. However, everyone was effective in identifying the major issues associated with readability of code and documentation. More than half of the students noticed variables that were never used, that exceptions were not always caught, the regular expressions could be rewritten to improve readability, one of the variable assignments should have been outside of the loop that it was in, and that hard coding should be avoided. Fewer than 5 of the students noticed that the code was inefficient in regard to collecting page content, that some variable names look like class names, and that many methods should be broken into smaller methods.

4.3 Class discussion

We reviewed the list of defects in a class discussion. The students agreed that these were all important issues, but the most interesting comments were that the exercise made them more aware of the importance of documenting code. Many students agreed that their homework assignments in their prerequisite courses were relatively small and that they had previously felt that documenting the code was too much work and unnecessary. However, the web crawler exercise made them reflect on the importance of good design and documentation. They felt that Jack the Hacker may have understood the code and may have been able to edit the code quickly, but that outsiders needed to make too much effort to understand and modify the code. They also suggested that Jack the Hacker may forget how to edit the code if he finishes it and then revisits it after a prolonged period of time! When asked about using object-oriented programming more efficiently for this code, many students also said that they did not include this in their write-up but that they addressed it in their implementation. In summary, this exercise illuminated the importance of good design and documentation.

4.4 New code

Students were asked to modify the code that they reviewed or to implement new code from scratch in any language of their choice in order to address the new format of the website that the web crawler visits. All teams created new code from scratch. Four teams wrote the new code in JAVA and one in Perl. Again, all of the teams emphasized that the code that they reviewed was poorly designed and documented and that they preferred to avoid modifying the code.

5. CONCLUSIONS

Undergraduate students in the introductory courses often underestimate the importance of good design and documentation for code. This is not surprising because most students only have experience with small programs. We gave the students an exercise in which they reviewed code that had design flaws and poor comments. We
collected data from 24 students that reviewed the code and found that the average student spent approximately 79 minutes reviewing the code and identified 8.58 defects. There were 12 types of defects that the students reported. All students identified the major documentation issues, but only some of the design issues. Finally, a class discussion revealed that many students had previously underestimated the importance of good design and documentation because their limited experience from the prerequisite courses focused only on small problems where they worked individually. This exercise made them aware of the importance of creating good design and documentation that others can easily understand.

REFERENCES


