Bugs Pattern Detection Application in JDBC using Static Analysis Non-Linear Method

Shadrach Jabonir\textsuperscript{1} and Ford Lumban Gaol\textsuperscript{2}

\textsuperscript{1}Departement of Magister Information Technology, Binus University Anggrek Campus, Jakarta, Indonesia
\textsuperscript{2}Departement of Magister Information Technology, Binus University Anggrek Campus, Jakarta, Indonesia
\textsuperscript{1}shadrach.jabonir@live.com, \textsuperscript{2}fgaol@binus.edu

Abstract

In this paper, it will be explained about an application which able to analyze the quality of java program code in implementing Java Database Connectivity (JDBC). The analysis will be based on existing best practice in implement JDBC. To analyze JDBC, bugs patterns are needed to be compared with existing best practice. Various methods are available to be used in constructing this kind of application, but in this paper, static analysis non-linear method is used in designing this application. Static analysis means the java program code will be read and analyzed without executing. Non-linear means the java program code will not read sequentially, but it will follow the flow of the program code itself. And this application will be called Bedhigasan, which able to detect bugs pattern in implementing JDBC and it will report those bugs to the web page including with location of the bugs.

Keywords: Bugs pattern, Static Analysis, Non-linear, JDBC, Java

1. Introduction

Development of science in the field of programming languages in recent years is very rapid. But due the rapid era of programming language, there is no offset for the standardization of quality checking about good program code. It will be deathly if the program which has been lived and it had a bug and the compiler did not able to found the bugs before running the program. Therefore we need a detection system which able to find bugs patterns in the program code. This detection refers to the empirical or based on experience. Methods which can be used are static code analysis and scan the program code in non-linear way.

Java is one of the popular programming languages nowadays. Java was enthused because of their concept, once it created, it could be run anywhere. Java is a programming language which used by many companies because of their open source concept. This concept makes company did not need to obtain a java certified programmer, if they want to use his/her services. Java is also can be regarded as the foundation for the type of network applications and also as the global standard for the manufacture of mobile applications, games, web-based content, and also as enterprise software. Java is able to interact with any kind of database products via JDBC (Java Database Connectivity). With certain driver, JDBC would able to bridge database with java program. JDBC has an API (Application Programming Interface) for bridging Java with all database platforms. JDBC provides an API that can be invoked for SQL-based database access.
In fact, JDBC has not been proved whether is able to maintain the integrity or not. If there is a problem then the java code would made the integrity of data threatened or a deadlock program and then the program must be restarted roughly. It needs an application that could find the variety of bug patterns in JDBC. It’s almost impossible if it detected with dynamic code analysis. Because of the program cannot be seen buggy unless executed.

So with this conclusion, good “bugs pattern” detection is better to use static code analysis with non-linear approaches to find bugs before execution. And those bugs may potentially problems for database or other programs. Non-linear was chosen because it scans program code not as sequentially. This is good, because the concept of java is “object-oriented programming”, it means that the programs are made in modular way. Linear method will be difficult to scan the java code because it scans only by what is visible for each line from top to the bottom. This is may caused an error while scanning the program code, because it does not scan the whole of the program code.

2. Related Works

According to reference journals [1] bugs pattern occurs for several reasons, such as difficulty of program language features itself, miss understanding of API semantics, miss understanding when the code is modified during maintenance, or typos. In another journal [2] presented to detect bugs pattern, it need to design a simple detector which can effectively detect the patterns of errors.

According to book [4], using JDBC is simple, and it needs to take only a few steps to add database functionality to your application. The steps involved are as follows:

1. Select/obtain a JDBC driver, or use the JDBC-ODBC bridge driver.
2. Obtain a database connection using DriverManager or a DataSource and a URL that’s appropriate for the driver you’re using.
3. Create a Statement or an instance of one of its sub interfaces (in other words, PreparedStatement or CallableStatement), and use it to execute SQL commands.

Findbugs scans code in linear way when analyzing the java code, as it described in their journal [3], the detectors is scanning in linear way (see Table 2.1) through bytecode for the method in the class that were analyzed, using the instruction that is visited then it directed to the state machine. Bedhigasan has a different way to scan code program, which try to inspiring another way of code scanning in static analysis method. It helps Bedhigasan to eliminate false negatives when analyzing java code. And also it helps Bedhigasan to analyze the architectural of the code program (see Table 2.1).

<table>
<thead>
<tr>
<th>Table 1.1. Comparison between Bedhigasan and Findbugs</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Languages</th>
<th>FindBugs</th>
<th>Bedhigasan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Works on</td>
<td>Bytecode</td>
<td>Java Decompiled Code</td>
</tr>
<tr>
<td>Interface</td>
<td>GUI, command line, plugin</td>
<td>Web based gui</td>
</tr>
<tr>
<td>Architectural analysis</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Code Scan</td>
<td>Linear</td>
<td>Non linear</td>
</tr>
</tbody>
</table>
3. Determining Application Method

After judging from several similar applications in another journals, it can be concluded that the bugs pattern application is better to use static analysis methods. Static analysis is also more desirable because it is easier to applies. This happen because with this method the application does not need to be ran in order to found bugs. Unlike when using dynamic analysis methods, it will be very difficult to find bugs, because those errors will arise when a variety of test cases, tested on the state of executing program. And this kind of checking is very risky because it will stall the developing time and increase the cost of developing. If it using the static analysis type, then the program will be read not in running condition. Finally the program code will be seen empirically, where the view of the various possibilities that could make program code slipped in error.

![Figure 3.1. Dynamic Code Analysis versus Static Code Analysis](image)

As seen in Figure 3.1, the bugs detection process in static analysis method takes shorter order than dynamic analysis. Therefore the making of this application will use the static analysis method. Moreover code scanning method that was carried in this application does not use as many common methods of static code analysis. Most of similar applications used to read program code in a linear fashion. Application which built in this paper is using the method of non-linear code scanning, which read all code as a whole program. According to the Table 3.1, the method which used in this application is non-linear static analysis, where the code will be scanned by non-linear.

<table>
<thead>
<tr>
<th>Linear Code Scan</th>
<th>Non-Linear Code Scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus in line visited</td>
<td>Focus to whole code program</td>
</tr>
<tr>
<td>More looping</td>
<td>More looping but with fewer range</td>
</tr>
<tr>
<td>Bad control flow</td>
<td>Better control flow</td>
</tr>
</tbody>
</table>
4. Developing the Application

4.1. Base Algorithm

Application Bedhigasan built to be able to find bugs in the program code which uploaded on Bedhigasan web pages. Basic Bedhigasan application algorithm is designed based on the value of information gain. Once the information gain has been calculated (gain information calculation cannot be included in this paper due the page limitation) then the output is a decision tree which able to decide whether the program code is buggy or not. After the tree is already formed then it will be implemented into the analyzing process algorithm of Bedhigasan application. Applications Bedhigasan could test 3 varies cases:

1. Connection & Driver Manager
2. Statement
3. Transaction

All those three cases are separated each other, so that if the user of Bedhigasan application just want to do the testing with one case or more, it can be done without affecting the results of other cases.

4.1.1. Connection & Driver Manager

Connection is JDBC syntax, this syntax form an instance to accommodate the connection path to the database. To open the connection, the JDBC requires the datasource which want to be linked with java program. Usually to open a connection are based on a data source such as a file system or database. Moreover after connecting to the database, it needs to be closed with syntax close.

Temporary the Bedhigasan application only recognize DriverManager syntax which commonly used by programmer to connect java program with database. JDBC Driver Manager is the syntax for loading the driver settings to open a connection with the database. Driver manager generally requires URL, password and username, those three things are standard protocol in connecting to the database. The decision tree learning for Connection & Driver Manager case will be compiled as seen in Figure 4.1.
4.1.2. Statement

Statement itself is an instance to load the query syntax that will be sent to the database. There are 2 kinds Instance of Statement, Statement and PreparedStatement. Statement syntax is used to send a SELECT query, because it has a result related with query that sent to the database. Statement should be instantiated first then use the syntax executeQuery to execute query syntax which is sent from java into the database. After getting the result, it can be processed in a java program for displaying data or just want to know the data. After the statement finished executing the query syntax, the statement itself also needs to be closed.

PrepareStatement is also part of the statement, but PrepareStatement is used to process UPDATE or INSERT query. PrepareStatement itself can be executed the query according to the parameters which seted in java, this is one of the java syntax to prevent SQL Injection. After PrepareStatement called, it needs an executeUpdate for execution. After the PrepareStatement finished executing the query syntax, the statement itself also needs to be closed.

The decision tree learning for the case statement will be compiled as seen in Figure 4.2. The decision tree learning for PreparedStatement case will be compiled as seen in Figure 4.3.

![Figure 4.3. Decision Tree Learning of Prepared Statement](image)

![Figure 4.4. Decision Tree Learning of Transaction](image)
4.1.3. Transaction

Transaction is one of the features that cater execution JDBC query syntax for more than one process and it would like to ensure the integrity of data. Because the problem is if one query syntax has an error occurred in the java program or maybe in database, and the program has sent query syntax to the database, but there are several other processes remain in 1 transaction process that might not be completed yet, then data integrity in that database is threatened. To perform transaction processing it needs to make auto commit on the java set to false, because basically java auto commit is already true for each process that sent to the database. Commit syntax required to finish all the query syntax sent to the database as a 1 transaction process. Rollback syntax is also required to restore the data to its original position if something goes wrong in the middle of process. After that, it is necessary to returns the initial settings of auto commit to a true value. The decision tree for the case of the Transaction will be compiled as seen in Figure 4.4.

After all the decision tree of each case has been made, all of the tree used as patterns in bug searching which the program are uploaded in Bedhigasan applications. By calculation gain information and form a decision tree, bugs pattern of each sample can be used as a decision tree in 1 piece. To be able to implement the tree in the application Bedhigasan, it is necessary to use non linear scan method code to read the program code. Unlike the Findbugs, Findbugs do not see the overall flow of the process because Findbugs read in a linear fashion. So need a lot of new bugs pattern created for each case of new bugs.

4.2. Application Layout

The currently built Bedhigasan application is just a prototype, so this application have not using additional library for CSS and jQuery to enhance its GUI.

According to Figure 5.1, it can be seen there is display to select cases that consist of Connection & Driver Manager, Statement, and Transaction, and also there is “choose files” button to open file browser window.

![Figure 5.1. Home Page Bedhigasan Application](image)

According to Figure 5.2, it can be seen there is a validated file browser window so only .class file can be uploaded, and it is also possible to upload multiple files.
According to Figure 5.3, it can be seen the result is displayed as table after the process button is clicked.

4.3. Application Architecture

As a whole these are the processes that executed once the application is running. User upload file that want to be analyzed, and then the application will process the files based on the selected case. And then the application will analyze the program and see if there is any matches bugs pattern with the tree from each case. If there is any bugs, the application will mark bugs at the part in the function and tell what kind of the bug it is. After that the result will be interpreted in a more user friendly sentence. This process will be iterated as many as the amount of the cases and the uploaded files. After that, the messages will be sent back to the web, and displayed as a table in bedhigasan web pages.
This application is built with an OOP concept, moreover this application also made in web application so it implements MVC (model, view, controller) concept. This application also can support multiple users to analyze code that uploaded on the web page. Basically there are 5 main packages in this application:

1. util
2. service
3. service.impl
4. engine
5. action

Util is a package that consists of java programs to run activity outside of the application, such as upload file and decompiling java programs. In this application, the program will be uploaded as .class files, so it needs decompiler to open it. Moreover there are also java program to write and read file. Service is a generalization of programs that implemented in serviceImpl. Engine is the main packages in Bedhigasan application that written in java code. It consists of piece of codes to analyze files. Action is a package that being called by web pages to execute each case by the selected services in web page.

For more detail about each class, please see Figure 6.1:

There are 4 main actions that can be executed in the web pages:

1. FileUploadAction
2. ConnectionDriverManagerAction
3. StatementAction
4. TransactionAction

It can be seen in Figure 6.2 that all actions represent the options in the web page. FileUploadAction will always be executed because to analyze file, file must be uploaded first and then sent to bedhigasan folder, and then it will be decompiled and then read and then analyzed. For more details about each action, it will be explained after this.

4.3.1. FileUploadAction

According to Figure 6.3 FileUploadAction is a service ran by web page to upload file which want to be analyzed by user. This service will sent files which user selected and uploaded in Bedhigasan. After all files have been successfully uploaded, class fileTool will inform fileUploadAction class. Uploaded files are bytecodes(.class), so common text editor cannot read them. FileUploadAction will execute JadTool class where the files will be decompiled to String so common text editor can read it. So FileUploadAction class will get the response which will be changed to messages for the web pages. If the file is successfully uploaded the message is "File(s) successfully uploaded" otherwise the messages is "Failed to upload file(s)".

4.3.2. ConnectionDriverManagerAction

It can be seen in Figure 6.4 ConnectionDriverManagerAction will be executed if checkboxes are checked in web page after process button is pressed. After that ConnectionDriverManagerAction class will validate web page to check if Connection & Driver Manager is checked. If it is checked, then the class will get all uploaded files names. And then ConnectionDriverManagerServiceImpl class will be called with the filenames as its parameter. And then the filenames will be iterated and analyze function will be called. In analyze() function, file will be read by reader class. And then all code will converted to string,
then the reader class will return as allCode that contain string code with its line number, it also return all function names in each class and detail from each functions. Checking will be done in each function that exists in each .class file. Detail contains information about start and end linenumber for each function. After that ConnectionDriverManagerServiceImpl class will execute each analysis in analyzer class. Analyze process will be done as each pattern decision tree that has been selected before. For Connection & Driver Manager case the first being checked is if there is any DriverManager syntax in each function, because it is impossible to search the username first. If DriverManager exists then it will enter next process analyzeForwardNull (username), to check that username is null or not in this function. Then it will also check if password and url are null or not. Then analyzeDirectionContains (driverManager) will be executed. And then it will find the instance of connection. Then it will also find close syntax in that program.

If that function called in other function, then that function will be searched first and will look syntax that needed. Example, DriverManager is called in other function, and then Bedhigasan Application will be looking for other syntax about the information of DriverManager. This is the different between non linear method and linear method.

After analyze process is done analyze class will return results to each analyze processes. Then ConnectionDriverManagerServiceImpl class will execute prosesMessages in interpreter class. When the pattern is being checked, it will return a boolean value. To determine the meaning of result, the message should be displayed in a more user friendly sentence. After that ConnectionDriverManagerServiceImpl class will sent that boolean value. The boolean value is used to determine if the action is success or not. After that ConnectionDriverManagerAction class will get the finalResults with getFinalResults function, where the finalResults is contained with the analysis results that have been converted into user friendly messages and its location for each function on that files. After finalResults is get ConnectionDriverManagerAction class will sent the results to web page. And in the web page, the results will be displayed as table where help user find the mistakes in the program that they uploaded.

4.3.3. StatementAction

According to Figure 6.5 almost all processes in StatementAction are similar to ConnectionDriverManagerAction, the different is only at analyzing process, so it is not necessary to explain about the uploading file process and the others. Statement in Figure 6.5 is a PrepareStatement (as explained before, Statement is divided into 2 varies: Statement and PrepareStatement). In Bedhigasan Application, to differentiate for each statement in the code program is by executeQuery or executeUpdate. If executeUpdate are exists then it looks for the INSERT or UPDATE query syntax, if it is found then looks for the prepareStatement syntax. And after that it will looks for the close syntax. If close syntax found then it will looks for the PreparedStatement instance. And just like Connection & Driver Manager, the results will be sent to interpreter class, then it will get the more user friendly messages and finally the messages will be shown in web page displayed in table.

If that function called in other function then that function will be find first and it will looking for those syntax. Example, close syntax called in other function, so Bedhigasan application will looking for that function and find the information about close syntax in that function. This is the different between non linear method and linear method.
Figure 6.1. Class Diagram of Bedhigasan Application
4.3.4. TransactionAction

According to Figure 6.6 almost similar to the StatementAction, the different is about the analyzing process. First it will looking for commit syntax in each function, if that found then it will looks for setAutoCommit(false) syntax which should be found before commit syntax. Then if it is found, then it will look for rollback syntax which should be found after commit syntax. Just like statementAction process, analyze class will sent the results and whole results will be processed through interpreter class to get user friendly messages. Then all those messages will be sent to the web page and displayed as table.

If that function called in other function, then it will looks for that function first then it will looks for that syntax. Example setAutoCommit(true) syntax is called in other function, so Bedhigasan application will be looking for that function and its information about setAutoCommit(true). And this is the different between non linear and linear method.

5. Conclusion

Due the page limitation, in this paper can not be explained more detail about the mathematical “information gain calculation”. To determine a “decision tree learning”, it requires its own mathematical calculations for each tree. As well as the testing of this application, Bedhigasan has been tested and compared to findbugs. The results are bedhigasan can increase the accuracy in finding bugs pattern, but unfortunately Bedhigasan had too many false alarms. However, these false alarms can be avoided by providing an exception in certain conditions. We expected to open minds of many people about “bugs finder application” by using a non-linear static analysis. And can be used for those who want to develop it in the future.

![Figure 6.2. General Flowchart of Bedhigasan Application](image-url)
Figure 6.3. Sequence Diagram of UploadFile

Figure 6.4. Sequence Diagram of ConnectionDriverManager
Figure 6.5. Sequence Diagram of Statement

Figure 6.6. Sequence Diagram of Transaction
References


Authors

Shadrach Jabonir received his bachelor in Information Technology-Software Engineering from Bina Nusantara University, Jakarta, Indonesia, 2011. He is Specialist in Java Programming, Software Architecture and Software Engineering.

Ford Lumban Gaol received the B.Sc. in Mathematics, Master of Computer Science and the Doctor in Computer Science from the University of Indonesia, Indonesia in 1997, 2001 and 2009, respectively. He is currently Associate Professor Informatics Engineering and Information System, Bina Nusantara University www.binus.ac.id. He is the Chair Research Interest Group Leader “Advance System in Computational Intelligence & Knowledge Engineering “ (IntelSys) Bina Nusantara University.

Dr Ford is the Vice Chair of IEEE Indonesia section. He is the Chair SERSC: Science & Engineering Research Support soCiety Indonesia Section. Dr Ford was the ACM Indonesia Chapter Chair. Dr Ford is the recipient of the Visiting Professor to Hong Kong University in 2011.