Tutorial Enhancement and Automated Code Helper

by

David Burela (BCOMP)

A dissertation submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Computing

University of Tasmania

(November, 2006)

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma in any tertiary institution, and that, to my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Signed

Abstract

This work proposes a system, entitled TEACH (Tutorial Enhancement and Automated Code Helper) which will allow students to complete programming tutorials online with the ability to submit their work to a server for instant analysis and receive meaningful feedback on any errors found with the code. This is achieved through a chosen set of analysis tools which have been brought together as a system and presented here as the "Three Spheres of Analysis". The building of TEACH will test the hypothesis that students will prefer learning in the newly developed automated tutorial system over the current tutorial system. The results show students are receptive to the system; however are not convinced that it would replace the current tutorial system.

Acknowledgements

I would like to begin by expressing my extreme gratitude to my supervisor Dr. Julian Dermoudy. Thanks to his experience and wisdom I was able to complete this thesis, without him it would not have been possible. He was very accommodating with his vision of the research allowing room to move, something he was not required to do but it is very much appreciated. His guidance began in my 1st year of computing and continued through the last 5 years.

My partner Emma Woolford deserves my gratitude next. Emma's experiences in writing a thesis last year helped prevent a lot of suffering on my part for which I am grateful. Without her love and late nights of proofreading my thesis I would not have completed it in time.

Finally a general recognition to all of my friends and family especially the ones that helped me with minor tasks in the last week! (In no particular order) James Gourley, David Hall, Conan Young, Colin Robinson, Rob Saunders and Raelene Morey.

Table of Contents

List of Figures

List of Tables

1. Introduction

The aim of this thesis is to build and evaluate the effectiveness of a web-based tutorial system. The system should allow students to enter code into a web page and have it submitted to a server for compilation and assessment, for the purpose of assisting beginner programmers learn programming basics. The hypothesis for the research is that students will prefer learning in the newly developed automated tutorial system over the current tutorial system.

"*When learning to program, it is essential that students are given the opportunity to practise in an environment where they can receive constructive and corrective feedback. Feedback is an especially important factor in the learning process when it is available on request. However, with large class sizes, it is difficult for teaching staff to synchronize their heavy schedules to provide additional help when the students need it.*"(Truong, Bancroft et al. 2005)

The proposed system, entitled TEACH (Tutorial Enhancement and Automated Code Helper), will allow students to select a tutorial question that they with to complete, the selected tutorial being loaded onto the webpage allowing the student to modify it. Upon completion of the required tutorial code the student will then submit it to the server which will then compile and execute it. A series of analysis tools will then be run against the student's code to detect and provide meaningful feedback on any errors found with the code. The set of analysis tools run against the code has been brought together as a system and presented here as the "Three spheres of analysis".

1.1. Thesis structure

Chapter 2 presents a review of the literature that will describe the context of all the technologies and tools that are considered in this thesis.

Chapter 3 explores the tools available for use by the system and the decisions on why the final tools were chosen. The "Three Spheres of Analysis" will also be introduced in this chapter.

Chapter 4 gives a detailed description on how the system was implemented and the structure of each component.

Chapter 5 describes the testing process and how the testers were recruited to evaluate the system.

Chapter 6 examines the results of the survey submitted by the testers.

Finally, Chapter 7 concludes this thesis, and presents future work.

2. Literature Review

2.1. *Overview*

The purpose of this literature review is to present an exploration of tools which would be of use in assisting students to learn more effectively. Various approaches and techniques will be explored within this review to determine which are suitable to the architecture and application. The major goal of this literature review is to provide an understanding from which to select appropriate solutions, for later investigation within the thesis.

2.2. *Structure and Scope*

Section 2.3 outlines the background of the thesis and also the context of the research.
Section 2.4 explores any related work previously conducted in this field of research.
Section 2.5 will inspect matching methods that may be of use to this research.
Section 2.6 presents style checking of Java source code.
Section 2.7 introduces static analysis of source code, and investigates available software solutions.
Section 2.8 examines dynamic analysis and its role in software testing.
Finally, Section 2.9 will investigate tools upon which TEACH is built.

2.3. *Background*

2.3.1. KXT101 Programming and Problem Solving

KXT101 Programming and Problem Solving is the first year programming unit at the University of Tasmania. The focus of this thesis is to assist these beginner programmers. KXT101 teaches students the basics of Object Oriented programming in Java, and how to test/debug programs. The unit is a requirement for certain degrees and a pre-requisite for the study of further programming units, as it teaches programming fundamentals (UTAS 2006).

"*Students learn to use a high level language such as Java to write programs which solve problems defined by a program specification. They master fundamental concepts relating to imperative, object-based programming and are introduced to concepts relating to graphical user interfaces and event driven programs. Students are required to demonstrate syntactic, logical and strategic knowledge of the programming constructs introduced in the unit. They are expected to use systematic processes to plan, document, debug and test their programs. Programming exercises are introduced in the context of small problems.*" (UTAS 2006)

2.3.2. Java Language

The Java language was initially created by James Gosling at Sun Microsystems. The initial aim of the project in 1991 was to create a tool which could be used to extend other programming languages to handle programming tasks that were traditionally hard to do. The first version of Java had been intended to be a cross platform environment for developing software for home appliances. Java was meant to simplify development as traditional languages like C++ are compiled to a processor specific binary, but home appliances have a variety of different processors, meaning they required additional development and testing time. As home appliances are usually a low cost item, keeping development costs down is a large area of conern (Savitch 1999).

In 1994 James Gosling came to recognise that his language would be ideal for the internet. In 1995 Netscape decided to make the next version of its web browser, Netscape Navigator, capable of running Java programs(Savitch 1999). Java's initial surge of popularity was due primarily to being one of the first programming languages which deliberately embraced the concept of writing programs that can be executed using the Internet (John Lewis 2001).

Java source code is not compiled into an executable binary, instead it is converted into bytecode. This bytecode is an intermediate format which allows it to be portable.

"Java byte code executes inside a Java Virtual Machine (or VM). The Java VM provides host services for the Java application/applet. These services include memory allocation, garbage collection, i/o, and basic graphics and windowing. The Java VM also enforces security, preventing Java byte code from interacting directly with the hardware on the host OS. Inside every Java-capable browser is a Java VM. There are also stand-alone Java VMs that can execute Java applications outside of a browser."
(Sun 1998)

By compiling Java programs into bytecodes it allows the programs to be cross platform, as it has not been tied to a specific processor. The first Java Virtual Machines interpreted each bytecode instruction as it encountered it. All this parsing meant that Java was much slower than traditional compiled languages like C++, but more recent versions of the Java Virtual Machine include features such as JIT compilation (Just In Time) which compile the bytecode form into machine-native, or operating-system-native instructions which greatly enhance the performance of the executing program.(Sun 1998)

Table 1 Top 20 identified programming errors of beginning students (Hristova, Misra et al. 2003)

Syntax errors	
1.	= versus ==
2.	== versus .equals (faulty string comparisons)
3.	mismatching, miscounting and/or misuse of {}, [ ], ( ), " ", and ' '
4.	Confusing "short-circuit" evaluators (&& and ||) with
5.	conventional logical operators (& and |).
6.	incorrect semi-colon after an if selection structure before the if statement or after the for or while repetition structure before the respective for or while loop
7.	wrong separators in for loops (using commas instead of semi-colons)
8.	an if followed by a bracket instead of by a parenthesis
9.	using keywords as method names or variable names
10.	invoking methods with wrong arguments
11.	forgetting parentheses after method call
12.	incorrect semicolon at the end of a method header
13.	leaving a space after a period when calling a specific method
14.	>= and =<
Logic errors	
1.	improper casting
2.	invoking a non-void method in a statement that requires a return value
3.	flow reaches end of non-void method
4.	methods with parameters: confusion between declaring parameters of a

	method and passing parameters in a method invocation
5.	incompatibility between the declared return type of a method and in its invocation
6.	class declared abstract because of missing function

2.3.3. Common Beginner Programmer Mistakes

When starting out, students new to programming frequently make the same mistakes (Hristova, Misra et al. 2003). Gathering a list of these common mistakes can help focus the learning exercises that are given, in order to help a student spot these problems in the code by themselves.

Hristova, Misra et al (2003) describe an error detection tool that was created at Bryn Mawr College, titled Expresso. The paper explains how they created a list of common programming errors that students make and what these errors were. The researchers surveyed a number of teaching staff and students and created a final list. They originally had a list of 62 errors, but cut it down to a list of the top 20 errors that they felt were essential from an educational perspective as shown in Table 1.

Table 2 Common Java programing mistakes (Topor 2002)

1.	Not specifying the size of a new array
2.	Not using correct array bounds
3.	Doing arithmetic on an instance of a wrapper class
4.	Adding a value of a primitive type to a collection (a set or list) or a map
5.	Not casting the value of type **Object** returned by **list.get(i)** or **map.get(key)** to the required type
6.	Using static components unnecessarily
7.	Not reading the next line inside a loop
8.	Creating a string tokenizer for a line before checking the line is present
9.	Reading all input before processing it
10.	Threading code
11.	Doing nontrivial computation in a class constructor.
12.	Not using common API methods
13.	Not breaking out of a loop when required
14.	Not breaking at the end of each case in a switch-statement
15.	Assigning constant values to Boolean variables in if-statements.
16.	Declaring variables globally, unnecessarily
17.	Repeating code that should be in a method called repeatedly
18.	Being too complicated
19.	Combining computation and input/output in a single, complex method

Griffith University compiles their own list of common programming errors for their beginning programming course. Table 2 shows a summary of their commonly identified errors. Many of these are similar to the errors previously discussed.

2.4. *Related Systems*

2.4.1. Environment for Learning to Program (ELP)

Environment for Learning to Program (Truong, Bancroft et al. 2003; Truong, Roe et al. 2004; Truong, Bancroft et al. 2005) was designed at the Queensland University of Technology (QUT). The purpose of the system was to assist their first year Information Technology students develop their programming skills. It is an interactive web based environment for teaching programming basics. The system provides feedback to the students via a static analysis framework, allowing the students to see potential errors in their program. Figure 1 provides an overview of the ELP architecture.

Table 3 Five distinguishing characteristics of the ELP system (Truong, Bancroft et al. 2005)

1.	The system supports fill in the gap programming exercises and customized compilation error messages which reduce the complexity of writing programs. This allows students to focus on the problem to be solved and engages them more actively in the learning process.
2.	The system is web based which eliminates the programming environment difficulties that students usually encounter. This also enables smooth integration of programming with lecture notes, tutorials and other web based content.
3.	The ELP incorporates a program analysis tool which provides instant feedback for students about the quality and correctness of their programs.
4.	The ELP supports configurable exercises i.e. the exercises and the environment can be configured for different stages of students learning.
5.	The system allows tutors to provide additional feedback through annotations on students' programs

As well as providing the tools to detect problems, it assists the students learning by allowing them to concentrate on just the code, without being required to learn how to setup a working environment and compiler. This separation of learning to program and compilation tools can assist the learning process. Further features can be seen in Table 3.

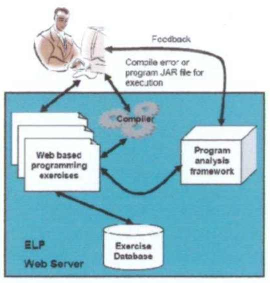

Feedback
Compile error or program JAR file for execution
Compiler
Web based programming exercises
Program analysis framework
Exercise Database
ELP
Web Server

Figure 1 ELP and the program analysis framework integration (Truong, Bancroft et al. 2003)

2.4.2. Automatic Assessment and Programming Tutor

Loosely based on ELP, the Automatic Assessment and Programming Tutor (AAPT) was created by University of Tasmania (UTAS) student Cynthia Sim in 2005. AAPT is a web-based tutorial system created using JSP and being hosted on Apache Tomcat 3.3.2. AAPT operates as follows; A student will load the exercise in a web browser and try to finish the exercise by completing all required code. Once the student thinks they have completed it, they submit the code to the server which then attempts to compile the code and execute it. If there are any compile time or runtime errors, the system will report them back to the student to review. Otherwise the system will return the output of the student's code to the screen.

AAPT has the advantage of teaching students over the traditional method of text editor and compiler, as it does not assume that the student has knowledge on how to install compilers or how to operate them. The basic feedback system also assists in students learning(Sim 2005).

2.4.3. Expresso

Expresso(Hristova, Misra et al. 2003) was created to assist beginner programmers understand cryptic compiler error messages. The researchers compiled a list of the most common Java errors that students make, by contacting professors at 58 universities as well as special interest groups. They then split all the errors into 3 groups: syntax, semantic and logic errors. Using this list of common mistakes as a base they were able to create a program that does multi-pass pre-processing on the code to detect these common errors. A verbose set of responses was created to direct the student to what the error was in a more readable message, and also providing suggestions on how to fix the error.

By assisting the student to be able to track down errors more easily than with the standard compiler error messages, it allows the student to concentrate on figuring out how to correct the code, instead of forcing them to first try to and locate where exactly the error is, and then how to correct it. They see the need for the tool diminishing as the student gains a better understanding of the language and compiler (Hristova, Misra et al. 2003).

2.5. Matching Methods

In order to allow the comparison of the student's code to possible problems and solutions, a method of matching the code to the reference code is needed.

2.5.1. Diff.exe

The GNU diff utility is used to show differences between two text files. This set of differences is often called a `diff' or `patch'. Diff compares two files line by line, finds groups of lines that differ, and reports each group of differing lines. GNU diff provides ways to suppress certain types of differences that are not relevant, changes in the amount of white space between words or lines a common option(Free Software Foundation 2000).

2.5.2. Regular Expressions

Regular expressions(Goyvaerts 2006) are used to do complex string matching on a piece of text. They are expressed as a sequence of characters which describe a pattern to be matched. The name regular expressions comes from the mathematical theory which they are based on(Goyvaerts 2006).
A simple example of using regular expressions is trying to find both spellings of the colour grey/gray, in a piece of text. The regular expression shown in Figure 2 would match all occurrences of grey and gray in the target text.

Regular expression: gr[ae]y

Target text:
The cement is coloured grey. The sky looks gray today.

The incorrect response has been greyed out.

Figure 2 Matching a regular expression on a piece of text

A much more advanced regular expression can be constructed that will find any email address. Figure 3 has an example expression that would match any email address. The expression can be modified by a programmer to verify that text entered is a valid email address by replacing the first "\b" with a "^" and the last "\b" with a "$" symbol instead (Goyvaerts 2006).

```
\b[A-Z0-9._%-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b
```

Figure 3 Regular expression for matching email addresses (Goyvaerts 2006)

Many tools that can be used to assist in the creation of regular expressions can be found at http://en.wikipedia.org/wiki/Regular_expressions.

2.6. *Style Checking*

By keeping a consistent style across a project such as coding style, indentations, methods, this will benefit the project by being easier to maintain (Hammurapi Group 2006). In addition, programming style is one of the most important foci of KXT101 — more important perhaps than correctness. It is important for tools to be developed/adopted which will aid in the identification of poor (aberrant) programming style and aid in its correction.

2.6.1. CheckStyle

Checkstyle(CheckStyle 2006) is an open source tool freely available on the internet, which is used for checking Java source code ensuring it adheres to a specified standard. Checkstyle can check for a variety of different coding conventions, including indentation, white space and naming conventions. A full list of available checks is listed in Appendix D.

Checkstyle is fully customisable in the types of checks it performs. An XML file is used to specify which style checks should be applied against the source code. Checkstyle can also be extended by a developer. Custom checks or filters can be written and then utilised in future checks. Reports can be exported and saved as either HTML or XML files (CheckStyle 2006).

2.6.2. Hammurapi

Hammurapi(Hammurapi Group 2004) is a code review tool for Java. It scans Java source files and inspects them to ensure they comply with coding standards. Reports can then be generated in either HTML or XML for the user (Hammurapi Group 2004).

Hammurapi inspects a piece of java source code reporting the findings by creating Violations, issuing Warnings, creating Annotations or gathering Metrics.
After analysis is completed Hammurapi accumulates the findings and generates a final report (Hammurapi Group 2004).

2.7. Static Analysis

Static code analysis is a technique for checking an application for potential bugs without actually executing it. It works at either the source-code level or byte code level of an application. Bug patterns are specified within the analysis program, the source that is provided by the user is then searched with any matches show where a potential problem could exist. This is useful as the code that has been checked may be syntactically correct but logic errors could be present within the code, which can be easily found with these tools.
Hidden logic errors that are commonly present are:

- Trying to dereference a null pointer
- Mistakenly using the Boolean comparison shortcuts & and | instead of the full && and ||

- Checking for a null pointer in one section of the code and then checking again shortly after (May indicate a misconception about the application)

Figure 5 shows a piece of example code which can cause a problem when the application executes, the programmer has checked if the object is null, but then invokes a method on the null object which will throw an exception, the error is obviously a mistake by the programmer and should be fixed. Standard testing procedures may not find the bug, as the execution path needed may not be called under most testing conditions, leaving a hidden bug which may be hard to track down later (Hovemeyer and Pugh 2004; Rutar, Almazan et al. 2004).

Static analysis draws on a number of techniques to detect possible code defects:

- Code pattern matching — match common bugs like if(str1==str2)
- Data flow analysis — tracks objects and their states
- Flow-graph analysis (cyclomatic complexity) — number of decisions

(Amit 2005)

```
public class  GenerateAST {
  private String printFuncName() {
    System.out.println(funcName +
"Generate AST");
  }
}
CompilationUnit
 TypeDeclaration
  ClassDeclaration:(public)

UnmodifiedClassDeclaration(GenerateAST)
    ClassBody
     ClassBodyDeclaration
      MethodDeclaration:(private)
       ResultType
        Type
         Name:String
       MethodDeclarator(printFuncName)
        FormalParameters
       Block
        BlockStatement
         Statement
          StatementExpression
           PrimaryExpression
            PrimaryPrefix
             Name:System.out.println
            PrimarySuffix
```

```
Arguments
 ArgumentList
  Expression
   AdditiveExpression:+
    PrimaryExpression
     PrimaryPrefix
      Name:funcName
    PrimaryExpression
     PrimaryPrefix
      Literal:"Generate AST"
```

Figure 4 Tree generated using parser (Amit 2005)

2.7.1. Static Analysis using Source Code

Static analysis tools that work on Java source code first need to scan through the source using a parser, after which rules are executed on that source code. Parsers turn the Java source code into a simplified tree-like structure known as an "Abstract Syntax Tree", an example can bee seen in Figure 4. Most static analysis tools execute an external parser to do the base work for them allowing the analysis tool to concentrate in detecting . After the tree is generated, the static analysis tool executes rules on the tree to find possible problems (Amit 2005).

```
if(myObject == null)
        myObject.doSomething();
```

Figure 5 Example of bug found with Static Analysis tools

2.7.2. Static Analysis using Java Byte Code

Instead of running on the source code, some tools instead work on the bytecode of the compiled Java Class files, an example of bytecode can be seen in Figure 6. As with source-based static analysis tools, most bytecode-based tools employ the use of an external parser first, after which the rules are executed on the parsers output. The types of problems that can be found using the Java bytecode are similar to the ones that can be found using source code analysis, neither approach has significant advantages or disadvantages over the other; they are simply different approaches to the same task (Amit 2005).

```
public class BadClass {
   public void doBadStuff() {
      System.gc();
   }
}
```

Figure 6 Code that exhibits the bug patterntaken from (Goetz 2006)

```
public void doBadStuff();
 Code:
  0:     invokestatic       #2; //Method java/lang/System.gc:()V
  3:     return
```

Figure 7 Bytecode listing for code in Figure 6 taken from (Goetz 2006)

2.7.3. PMD

PMD (InfoEther 2006) is a static analysis tool which works on Java source code searching for potential bugs. PMD is able to detect a wide range of potential problems: unused local variables and parameters, wasteful String/StringBuffer usage, unnecessary if statements, for loops that could be while loops and finally copy/pasted code. Rutar, N., C. B. Almazan, et al. (2004) states that in addition to the traditional bugs detected, PMD also detects bugs that are stylistic conventions that may cause potential problems if used inappropriately.

Table 4 Bug finding tools and their basic properties (Rutar, Almazan et al. 2004)

Name	Version	Input	Interfaces	Technology
Bandera	0.3b2 (2003)	Source	Command Line, GUI	Model checking
ESC/Java	2.0a7 (2004)	Source	Command Line, GUI	Theorem proving
Findbugs	0.8.2 (2004)	Bytecode	Command Line, GUI, IDE, Ant	Syntax, dataflow
Jlint	3.0 (2004)	Bytecode	Command Line	Syntax, dataflow
PMD	1.9 (2004)	Source	Command Line, GUI, IDE, Ant	Syntax

2.7.4. Bandera

Bendera (Bandera 2006) is a tool framework for model checking Java programs. The Bandera framework is organized as a modular pipeline of tools, each tool communicating with the preceding and following (Dwyer, Hatcliff et al. 2006).

Bandera requires the programmer annotate their source code with the type of checks to be performed, or nothing specified if the programmer only wants to verify some standard synchronization properties. Without annotations Bandera verifies the absence of deadlocks (Rutar, Almazan et al. 2004).

2.7.5. FindBugs

FindBugs (FindBugs 2006) is an open source bug pattern matcher which works with Java bytecode. The source, binaries and documentation are available from http://findbugs.sourceforge.net/. It is a static analysis tool that is able to be extended by a user to find custom defined patterns (Hovemeyer and Pugh 2004; Rutar, Almazan et al. 2004; CheckStyle 2006).

Currently, FindBugs contains detectors for about 50 bug patterns. All of the bug pattern detectors are implemented using BCEL, an open source bytecode analysis and instrumentation library. The detectors are implemented using the Visitor design pattern; each detector visits each class of the analyzed library or application. (Hovemeyer and Pugh 2004)

The bug detectors that FindBugs has implemented can be put into 4 main categories:

- Class structure and inheritance hierarchy detectors that look at the structure of the classes.
- Linear code scan detectors that work on methods of classes to be analysed.
- Control sensitive detectors which use a control flow graph for analysed methods.
- Dataflow detectors are the most complex of the detectors; these use the control and data flow of the application to find bugs. An example of a programming error which dataflow detectors can find is the de-referencing of a null reference-variable.

(Hovemeyer and Pugh 2004)
A complete list of the bugs may be found in Appendix E.

1. Raising the awareness of developers of the usefulness of bug-finding tools
2. Incorporating bug-findings tools more seamlessly into the development process: for example, by providing them as part of Integrated Development Environments
3. Making it easier for developers to define their own (application-specific) bug patterns
4. Better ranking and prioritization of generated warnings
5. Identification and suppression of false warnings
6. Reducing the cost of bug-finding analysis, through incremental analysis techniques, and background or distributed processing

Table 5 Conclusion and future work (Hovemeyer and Pugh 2004)

The paper "Finding bugs is easy" (Hovemeyer and Pugh 2004) makes suggestions on future work that could be done with FindBugs; Table 5 shows these. The key points of the suggestions are the raising of awareness of bug-finding tools, the integration of these tools into a developer's workflow and the identification of ways to keep the detection at a high quality so as to not overwhelm the developer with false positives.

2.7.6. JLint

JLint (JLint 2006) is a pattern matching tool which works on Java bytecode. JLint is similar to Findbugs as it also is an open-source tool which performs syntactic checks and dataflow analysis. In addition to this, JLint also has an additional detector for finding multi threaded problems and checking for deadlocks on resources by building a lock graph and ensuring that there are never any cycles.

JLint is not easily extendable by the user to find additional patterns (Rutar, Almazan et al. 2004).

2.8. *Dynamic Analysis*

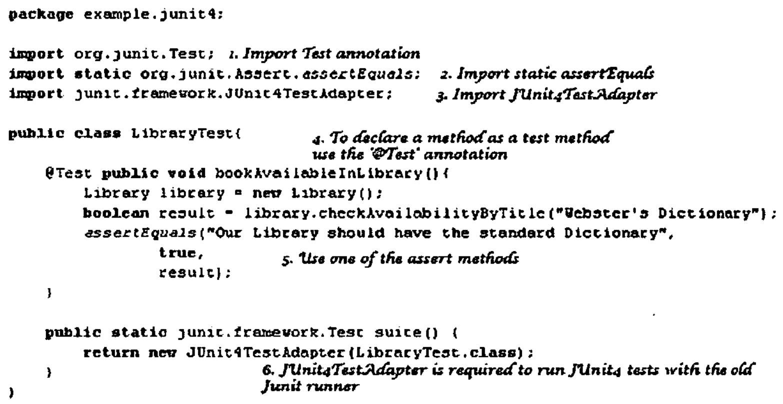
Dynamic analysis is the process of testing an application at runtime; this is predominately done by passing in various inputs and ensuring the expected output was returned. Dynamic analysis can be automated to execute against an application during the development lifecycle.

2.8.1. Test-driven Development

The idea behind test-driven development is to write a test case for a new piece of functionality before the actual code is written, the code is then written to pass the test. (Edwards 2003) discusses the benefits of students learning to program through this methodology. By learning to code an application in smaller pieces instead of the usual 'big bang' approach taken by students, they learn to be more confident in the piece they just wrote, thus allows them to make changes and additions with great confidence.

2.8.2. Junit

Junit is an open source unit testing framework for creating and running unit tests for the Java programming language. Junit was created by Kent Beck and Erich Gamma. When written, a unit test will test a piece of code to ensure the expected functionality of a piece of code works as expected. A Junit test case works by calling pieces of code, asserting what variables should be before and after a code section executes (Clark 2006). Figure 8 illustrates a unit test with assertions, a call is made checking the availabliltiy of a book after which an assertion is made that the result should be true. After execution the assertion will be verified and throw an error if the unit test failed.



```
package example.junit4;

import org.junit.Test;  1. Import Test annotation
import static org.junit.Assert.assertEquals;  2. Import static assertEquals
import junit.framework.JUnit4TestAdapter;     3. Import JUnit4TestAdapter

public class LibraryTest{            4. To declare a method as a test method
                                     use the '@Test' annotation
    @Test public void bookAvailableInLibrary(){
        Library library = new Library();
        boolean result = library.checkAvailabilityByTitle("Webster's Dictionary");
        assertEquals("Our Library should have the standard Dictionary",
                true,          5. Use one of the assert methods
                result);
    }

    public static junit.framework.Test suite() {
        return new JUnit4TestAdapter(LibraryTest.class);
    }                       6. JUnit4TestAdapter is required to run JUnit4 tests with the old
}                           Junit runner
```

Figure 8 A unit test written in Junit 4.0 taken from (Doshi 2005)

2.9. *Tools*

2.9.1. Hyper Text Mark-up Language (HTML)

Hyper Text Mark-up Language (HTML) (Raggett, Le Hors et al. 1999) is the standard language of the Internet. HTML was developed in the early 1990s by Tim Berners-Lee.

HTML files are simply text files consisting of text interspersed with standard tags used to 'mark-up' the formatting by specifying font type, where to place images, etc. A web browser will read in the formatting and render the page onto the screen based on the mark-up instructions. Links can be created between files to create a network of information. HTML files were used early on to help people share information (Raggett, Le Hors et al. 1999).

Table 6 Design Goals for XML taken from (Bray, Paoli et al. 1997)

1.	XML shall be straightforwardly usable over the Internet.
2.	XML shall support a wide variety of applications.
3.	XML shall be compatible with SGML.
4.	It shall be easy to write programs which process XML documents.
5.	The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6.	XML documents should be human-legible and reasonably clear.
7.	The XML design should be prepared quickly.
8.	The design of XML shall be formal and concise.
9.	XML documents shall be easy to create.
10.	Terseness in XML markup is of minimal importance.

2.9.2. Extensible Mark-up Language (XML)

Extensible Mark-up Language (XML) (Bray, Paoli et al. 1997) is a datastructure consisting of a set of tags to mark-up the data. The design goals of XML can be seen in Table 6. "*... XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.*

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form *character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.*" (Bray, Paoli et al. 1997)

2.9.3. ASP.NET 2.0

ASP.NET (Microsoft 2006) is a web programming technology created by Microsoft that allows developers to create dynamic web pages. ASP.NET web pages consist of both HTML mark-up and source code to enable the page to be dynamic. The source code of an ASP.NET web page can be written in any one of a number of different programming languages including C#, VB.NET and C++ (Mitchell 2006).

2.9.4. JavaServer Page (JSP)

"*JavaServer Pages (JSP) technology provides a simplified, fast way to create web pages that display dynamically-generated content. The JSP specification, developed through an industry-wide initiative led by Sun Microsystems, defines the interaction between the server and the JSP page, and describes the format and syntax of the page.*" (Sun 2005)

JavaServer Page (JSP) is a server side technology used for generating dynamic web pages. JSP uses a combination of XML tags and scriptlets to define the logic of the page, it also separates the formatting (HTML or XML) tags and returns these to the response page. By using this approach the page logic is separated from the design and display. JSP pages are compiled into servlets when it is called for the first time, subsequent requests for the page do not require further compilation unless the original JSP is modified (Sun 2005).

Although JSP pages are compiled into servlets, they have additional benefits over straight Java servlets in that they are simpler to develop, and allow a logical separation of code logic and page presention (Sun 2005).

3. Module selection

3.1. *Introduction*

In designing a solution with which to either prove or disprove the hypothesis presented above, a careful selection of components must be made to ensure that the final system is suitable in testing the hypothesis. First the choice of development platform is explored in Section 3.2 as the decision will affect all aspects of the system. Section 3.3 investigates the base framework upon which the rest of the system will be built. Section 3.4 details the importance of analysis tools in the system. Section 3.5 presents the 3 spheres of analysis and their role in the system, an investigation into the choices of each sphere is presented. The chapter concludes with Section 3.6 which explains which technologies were excluded from the system.

3.2. *Development Platform*

The choice of development platform for TEACH is an important component. The decision of development platform affects the ease at which the system can be developed; a wrong decision can be detrimental to the project. The longer term issue of support and further development must be taken into consideration when the decision is made.

The two leading web based platforms are Microsoft's ASP.NET (Microsoft 2006) and Sun's JSP (Sun 2005). An overview of both platforms reveals similar feature sets, both being well supported online, and both feature many tools to aid development.

A decision on development platform can't be done without looking at the overall environment in which it would be placed. Future students that take on the re-development or extension of the system may be at a disadvantage if ASP.NET were chosen as the School of Computing currently only teaches Java and as a result, they may not be able to easily build future work onto it. Additionally, the School of Computing has a significant investment in Tomcat installations in their server room. It would be of great advantage to utilise the existing facilities, rather than attempt to setup a new server environment.

It was decided to use JSP as the development platform due to the set-up of the School of Computing's web server infrastructure, policies, and teaching and learning curricula.

3.3. Base Framework

With the decision to go with JSP, the next issue was that of development. Should TEACH be started from scratch so that it can be crafted with its needs being met explicitly or should TEACH be built on top of an already existing framework?

Starting from scratch is always an attractive solution, as you get to build it to your vision of the system without compromising with another person's decisions. Unfortunately developing this way will drastically increase the development time of the system, and would require reinventing the wheel, as well as reinventing the same bugs which will require debugging.

Using a framework that has already been built will be advantageous. A framework completed by Cynthia Sim is AAPT; which was developed as a base framework for a web based programming tutor. AAPT contains many of the features required by TEACH: tutorial selection, a compilation system and a database backend allowing adding/updating of tutorials.

The decision was made to use AAPT as the base framework and to extend it. The advantage of taking this approach is a reduction in development time, meaning more time can be spent on the TEACH specific functionality, instead of the generic functions of a web based tutorial system. One potential limitation of using AAPT is being required to learn how the system is built and discovering where it can be extended potentially taking a significant length of time; however this is a small price to pay for a pre-developed framework.

3.4. Choice of Analysis Methods

The choice of analysis methodology took a significant portion of the planning time. A set of analysis tools would be required to run over a students work to detect problems, bugs and errors.

Picking a single style of analysis would pickup a specific span of problems, but may not cover a wide range leaving some errors unnoticed. After consultation with my supervisor, an agreed set of analysis was decided to form the boundaries of the project. These boundaries are discussed in the section 3.5.

3.5. The 3 Spheres of Analysis

The three spheres were designed to complement each other, each detecting a different range of issues the student may encounter, or that the lecturer may want to detect.
By focusing each sphere on a specific set, the best solution can be found for each. By combining the results of each, the union of the different test regimes can be provided. Figure 9 shows how the different spheres could complement the others to provide greater coverage.

With the wide range of solutions available for each of the spheres, it is important that the chosen solution is extensible. Future requirements from the School of Computing may require TEACH to be modified, by having the spheres able to be flexible and extensible it will allow certain aspects to be changed by the lecturers without requiring code changes to TEACH or the underlying spheres.

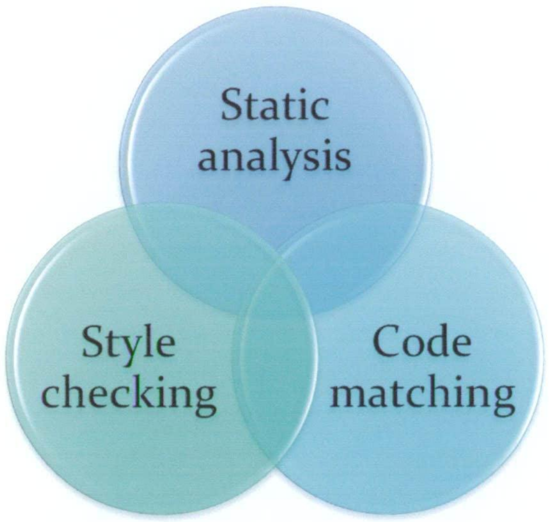


Figure 9 Graphical representation of the 3 Spheres of Analysis

3.5.1. Sphere 1: Static Analysis

Static code analysis is a technique for checking an application for potential bugs without actually executing it. Static analysis works at the source level, or the byte code-level, of an application.

This was seen to be an important module to TEACH as it would be the one to most directly affect the students' perception of the system. The types of errors found and the amount of feedback given was an important factor in deciding which tool to use in the solution.

Table 7 Bug finding tools and their basic properties (Rutar, Almazan et al. 2004)

Name	Version	Input	Interfaces	Technology
Bandera	0.3b2 (2003)	Source	Command Line, GUI	Model checking
ESC/Java	2.0a7 (2004)	Source	Command Line, GUI	Theorem proving
Findbugs	0.8.2 (2004)	Bytecode	Command Line, GUI, IDE, Ant	Syntax, dataflow
Jlint	3.0 (2004)	Bytecode	Command Line	Syntax, dataflow
PMD	1.9 (2004)	Source	Command Line, GUI, IDE, Ant	Syntax

The decision on which tool to use was not affected by whether the analysis was done on the source code, or the bytecode. In chapter 2 a number of tools were reviewed, Table 7 shows their some of their basic properties. Each will now be considered against the criteria for this project and against each other.

Bandera

Bandera was not practical for standard use. It requires the programmer to annotate their source code with the checks that should be performed. Without these assertions Bandera does not run on any useful piece of Java code (Rutar, Almazan et al. 2004) and it is difficult to see how students could easily produce these annotations without complicating the tutorial system. Due to this limitation Bandera was not considered further as a solution for the system.

JLint

JLint had many features that were desirable. For example it performs syntactic checks as well as dataflow checks. However JLint was not easily extensible which ruled it out as a viable choice when it was against similar tools which were extensible in an easier manner. As mentioned earlier extensibility is an important criteria due to future expansion.

PMD

PMD performs syntactic checks on a piece of source code, however it does not perform dataflow checks. The dataflow checks will be advantageous for TEACH as problems identified in the literature review would be detected.
Additionally many of the bugs detected in PMD depend on programming style "*For example, having a try statement with an empty catch block might indicate that the caught error is incorrectly discarded.*"(Rutar, Almazan et al. 2004), which may not be compatible with the lecturers teaching style.

FindBugs

Similar to JLint, FindBugs possessed many features that were desirable for TEACH. FindBugs performs dataflow checks on source code, which is advantageous due to the type of errors that students commonly perform. FindBugs provides a GUI interface in addition to the command line interface. The GUI assisted in learning how FindBugs performed its analysis and the options available which would be invaluable when trying to interface with TEACH. Reports are able to be exported to HTML or XML format, allowing the results to be easily obtained.

Finally FindBugs is flexible both in execution and in extensibility. When called from the command line parameters are passed to it to specify what checks to perform, allowing each execution of it to perform differing analyses if desired by the lecturer. Extensibility was previously mentioned as being an important feature as it will future proof the TEACH solution.

3.5.2. Sphere 2: Style Checking

By keeping a consistent style across a project such as coding style, indentations, methods, it will benefit the project by being easier to maintain (Hammurapi Group 2006).

A style checking module in TEACH ensures the students are learning to develop to a consistent theme. Teaching students to code in a consistent manner as they are starting to learn programming may help reinforce a good coding style.

Two different style checking applications were examined in the literature review. Both were open source applications that covered a range of style errors. Checkstyle was chosen, as it was better documented, possessed a solid list of the type of style errors that are found, was very easily modifiable. CheckStyle was able to be quickly integrated into TEACH as it had a simple command line interface and the XML reports were structured in a simple way.

3.5.3. Sphere3: Custom Matcher

Detecting logic and style problems is advantageous; however an application can successfully compile, be bug free, but still may not do what is required.
Although satisfied with the static analysis and style checking, the supervisor required a way to be able to detect if the student wrote the program to the specifications and didn't program a series of 'print' lines that would output the expected result instead of programming the required functionality.

It would be advantageous if more than one solution can be specified with a definition of "This is a good solution" or "This is a bad solution".

Literal Matching

A method was required which could match the code submitted by the student to what the lecturer expects from the student.

Literal matching of text was explored first. It had the positive aspect of the lecturer explicitly defining what the student should type for their solution to be correct for the tutorial. Unfortunately being this precise has the major disadvantage of being too strict on what the student could do. In order for the student to have been deemed to have coded a correct solution, they would have to have *exactly* the same text as the lecturer — the same variable names, the same white space, etc. This was too restrictive on the student, and so other options were explored.

Diffutils

Unix diff is a suitable way to match to different pieces of source code (the one submitted by the student, and the reference one from the lecturer) (Free Software Foundation 2000).

Diff was built for this kind of function, its sole purpose is to compare the differences between two different source files. Unfortunately it was not suitable for this system. It shows the differences between the two files, therefore if used to compare the differences between, for example, four possible solutions, the amount of data coming back would be overwhelming and would take much development to create any meaningful feedback from this system.

Regular Expressions

Without a suitable solution readily available, the decision was made to explore a custom solution. Regular expressions came up as a way to have softer matching of text. Instead of requiring an exact match of text, by using regular expressions it is more flexible. With exact text matching, a simple task such as matching a variable declaration is difficult, as there is a variable amount of white space that could be used eg.

```
int value1 = 5;
```

versus

```
int       value1=5;
```

By using regular expressions, the problem of white space can be easily overcome, as well as handling the choice of identifier. Figure 10 shows a regular expression used in TEACH to detect any String or int variable declaration.

```
(String|int)\s+\w+(\s+\=|\=)(\s+|)\w+;
```

Figure 10 Regular Expression to match a String or int declaration

Use of regular expressions also allows backtracking, meaning that a variable can be matched in one section, and then be used to match further in the expression.
An example is a variable declaration then an assignment to that variable later in the code.

3.6. *Other Analysis*

Dynamic analysis considered but was not included in TEACH. Although it would have complemented the solution, the added complexity would not have been worth the additional effort at this stage. Getting a solid focused analysis framework down was a better time investment rather than breadth in a shaky framework. The School of

Computing does not currently teach Unit testing, additionally it is an advanced topic that should not be introduced to first year students.

4. Architecture, Topology and Design

4.1. *Introduction*

TEACH was built as an extension to the original AAPT system created by Cynthia Sim. The development PC was an Athlon 3000+ with 512MB of RAM. Windows XP was installed, with TEACH running on Tomcat 4.1.31, with Java 5.0 Release 6.

The remainder of this chapter will explain the architecture of TEACH, it will first give an overview of the file system structure of the system, explain the database and finally the chapter will cover the application and the software solution.

4.2. *Directory Structure*

The directory structure of TEACH — which is essentially inherited from the original AAPT implementation — is split into two main components:

- The core components: the website, executable Java classes and databases
- External resources: tutorial questions, temporary submit directories, and other executables

Each of these will now be elaborated.

4.2.1. Core Components

The core components sit inside the `Tomcat\webapps\AAPT` folder. There are a total of three folders:

- `db`. This directory contains the Microsoft Access database that is used to store settings.
- `jsp`. This directory contains all of the website pages, images, and `jsp` files required for TEACH to run.
- `WEB-INF`. This directory contains all classes required to run the servlets.

The classes under the `WEB-INF` folder are further sub-divided into the following packages and sub packages:

- `Access`. This package contains all the classes required to retrieve information from the Access database.
- `Application`. This package contains the servlet classes used to submit code to be reviewed from the clients, and return the results to them.
    - `XML parser`. This sub package of the Application package is used to parse the XML outputted from FindBugs and Checkstyle and create objects to represent the information. This is explored further in sections 4.4.2 and 4.4.3
- `Entity`. This package contains all of the object classes: Questions, XML data, bugs, etc. It will be used by classes in the other packages.

(Sim 2005)

4.2.2. External Resources

The external resources folder is used to hold everything not directly related to the website, including:

- `Checkstyle`. Holds the Checkstyle 4.2 application
    - Check_settings. Stores the customisable style .XML files
- `Exercises`. The source files of the tutorials/exercises
- `FindBugs`. Holds the FindBugs 1.0.0 application.
- `SubmitWork`. When students submit their source code to be checked, it is temporarily stored here.

4.3. Database Design

TEACH uses a database to store information on the questions and solutions. The original AAPT database was used with many modifications.

There are 4 tables which are used to define all of the customisable aspects of TEACH, a diagram show the structure is shown in Figure 11.

- `aapt_question`. Stores the questions name, the path to the tutorial source code, the description displayed on the webpage, and whether it is enabled and shown on the webpage.
- `aapt_solutionweights`. This is used to define the search patterns and how much each successful match is worth. Also definable is the maximum number of times the pattern can be matched against the piece of source code.
- `aapt_solution`. This is used to allow the lecturer to override the default settings used in FindBugs and Checkstyle and allows them to specify their own settings. The naming of this table is due to its previous function with AAPT.
- `aapt_defaults`. The default settings for FindBugs and Checkststyle

Although TEACH expanded on the original work of AAPT, to simplify the implementation, all table names were kept to the original naming scheme of prefixing each table name with "aapt_".

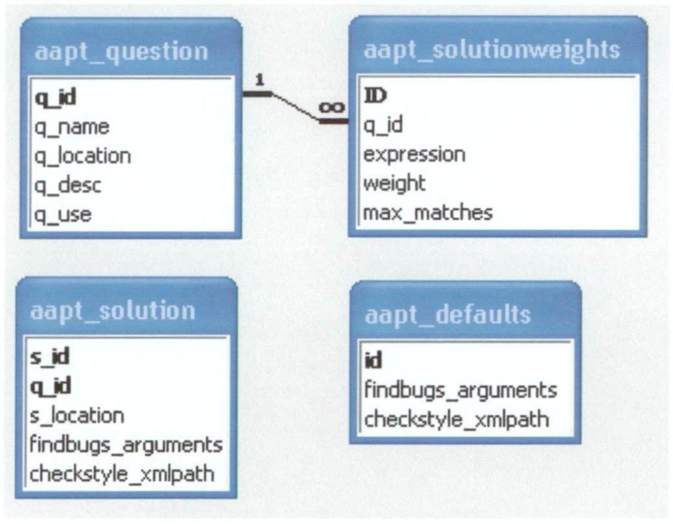


Figure 11 TEACH database structure

4.4. Application Structure

As indicated above, TEACH was built on top of the pre-existing web-based tutorial system, AAPT, which simplified development of the system. More time was able to be spent evaluating appropriate tools and implementing it well.

The majority of the development effort of TEACH was spent on the 3 spheres of analysis: Static analysis, Style checking and Code matching. Figure 12 shows how the three spheres are integrated into the system.

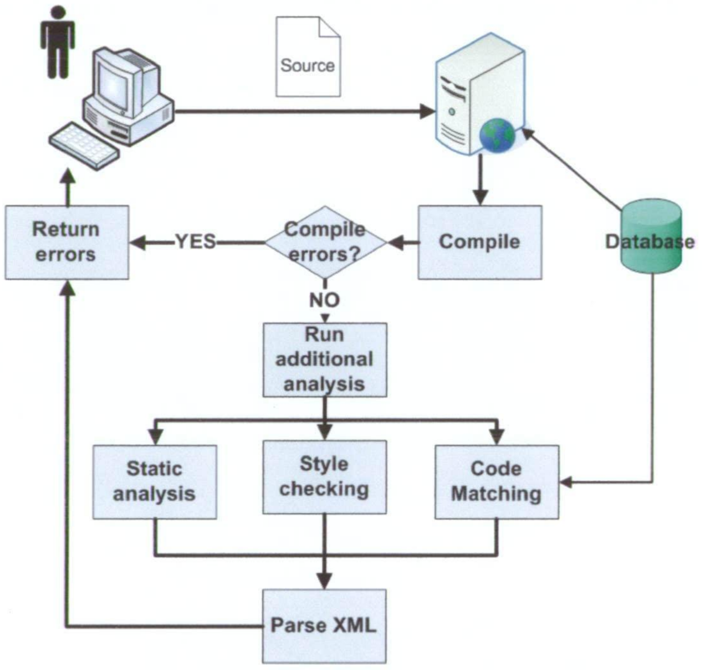


Figure 12 TEACH architecture

4.4.1. Compilation of Source Code

Compilation of the submitted code is the core required functionality of the system. AAPT provided the functionality for compilation, with some slight modifications made for TEACH. AAPT implemented the code compilation by creating a folder to store the submitted code, with the folder name based upon the IP address of the client machine. This implementation led to concurrency issues, for example if the same student was working within multiple browser windows, or if it appears that many clients are originating from the same IP. TEACH modified this to instead assign each submission a unique number, creating a folder with this number to work within, thus preventing concurrency issues.

Within the code of the tutorials, each may have its own class name, due to Java having a strict file naming convention for Java files, when code is submitted it is saved in its allocated folder named as the tutorial eg. *Division.Java.*

If there were any errors during compilation, the system records them, halts any further analysis, and returns the error messages to the user. If compilation of the student's code was successful, then further analysis is run. The code is compiled using the standard Java compile tool (javac)

4.4.2. Sphere 1: Static Analysis

As discussed in section 3.5.1, FindBugs was chosen as the analysis tool for static analysis of submitted code. It was integrated into the TEACH system to provide feedback on the students work. This is achieved by invoking FindBugs on the folder where the students work has been saved and compiled, Figure 13 shows the default command line call TEACH uses. The FindBugs application will then proceed to detect problems in the students code, the types of problems checked for is dependant upon the command line call, Figure 14 shows an example of how different checks can be added and removed.

```
findbugs -textui -low -xml:withMessages -outputFile bugs.xml -sourcepath filePath
```

Figure 13 TEACH default settings for FindBugs

Once analysis is complete FindBugs compiles a detailed report, exports it to an XML file, and saves it in the folder with the student's submitted code.

```
findbugs -textui -chooseVisitors +v1,-v2 -xml:withMessages -outputFile bugs.xml -sourcepath filePath
```

Figure 14 FindBugs call with selected visitors

A custom XML parser (`FindbugsParser.java`) processes the bug report and builds up a class object. This class object contains list of each bug's individual details: type, short description, line where it occurs, etc. These details are used to present the student with an understandable list of problems. The class object also contains the detailed information on each bug type, which will enable the student to resolve the bug. The final class object is passed onto the next servlet which handles the formatting of the webpage and how the list of bugs will be displayed to the student.

As discussed in section 3.5.1, one of the features of FindBugs is its flexibility and ease of being extendible. TEACH has embraced this and made it easy for the lecturer to define what checks should be performed. Each tutorial is able to have the command line call that will invoke FindBugs on student's code defined individually. This allows the lecturer to define which type of bugs should be looked for, and also to define which of the bugs found should be shown. Each bug has a priority level of how severe the bug is (High, Medium and Low). It may be decided that for beginning tutorials, only High priority bugs will be displayed, but for the more advanced (e.g. 2nd year students), all levels of bugs should be displayed.

Lecturers are able to create their own custom bug detectors and place them into the `plugin` subfolder. The lecturer may want to create custom detectors to detect of the student has made a call to a custom School of Computing class file. For example, it may be advantageous to detect that the student initialised, used, and then destroyed the custom classes.

If the tutorial has not defined its own FindBugs settings, then default settings will apply. TEACH will call FindBugs to perform the FindBugs default checks, and to report High, Medium and Low level bugs.

4.4.3. Sphere 2: Style Checking

As discussed in section 3.5.2, CheckStyle was chosen as the code style analysis tool. It was integrated into the TEACH system to provide feedback on the students' work. This is achieved by invoking CheckStyle on the folder where the student's code is held.

CheckStyle requires that when it is executed, the path of an XML file will be passed to it. This XML file specifies the standards that the code being checked should adhere to. TEACH allows the lecturer to define for each tutorial question, which coding style should be followed. The setting for which XML file to use for each tutorial question is retrieved from the database as required; if no file has been specified, then the default TEACH coding style XML file is used.

This flexibility will allow a lecturer to define their own standard for style checking over a set of tutorial questions, and allowing for individual questions to have their own style if necessary. Figure 15 shows the default invocation of Checkstyle and a call using a different XML file for that tutorial question.

```
checkstyle-all-4.2.jar -c TEACH_default_checks.xml -f xml -o checkstyle_errors.xml *.java
checkstyle-all-4.2.jar -c Q37_checks.xml -f xml -o checkstyle_errors.xml *.java
```

Figure 15 Example executions of CheckStyle

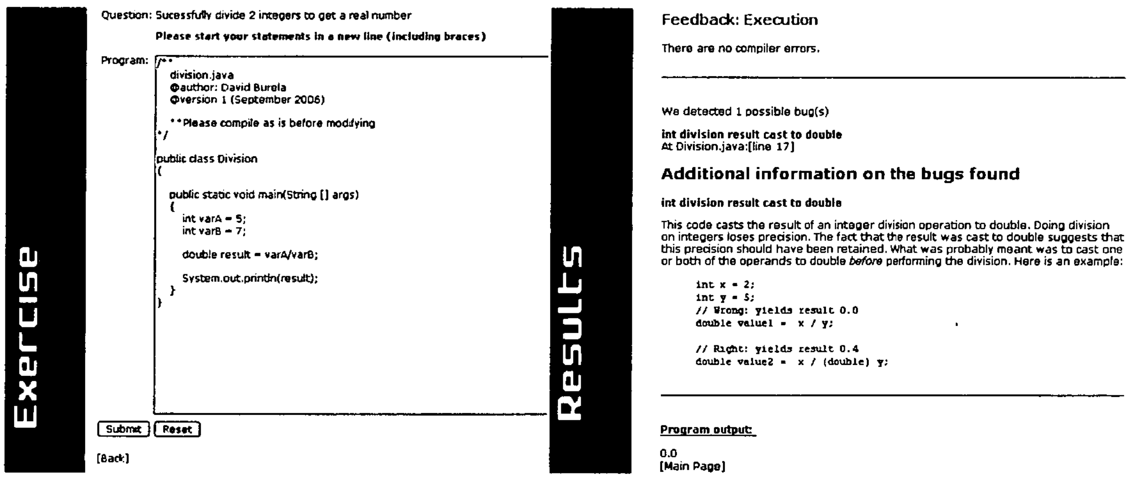
4.4.4. Sphere 3: Code Matching

Sphere three required a custom analysis tool to be created. The Java regular expression API was used to match the students' code against patterns defined by the lecturer. The lecturer may define multiple patterns in the database to be matched for a tutorial. Each pattern has other properties that can be defined: the point value for the pattern matching and the maximum number of times the pattern can accrue points for matching.

The code matching module, when called on a source file, will lookup the tutorial question in the TEACH database, and each pattern defined for the tutorial question will be retrieved. Each pattern will then be used in turn to try and make matches against the student's code, if a successful match is made the points value of the pattern is added to the student's score and an internal counter is incremented to hold the number of matches this pattern has made so far. After a match has been made, if the pattern allows any more matches the code is searched for a further match. If the pattern has reached the maximum number of matches, or there was not a successful match, the next pattern is evaluated. This cycle is continued until there are no longer any patterns left.to check. Once completed, the student's final score is passed onto the next servlet for feedback to the student.

4.4.5. Graphical Feedback

The analysis tools and the feedback generator were separated so that the analysis tools in the backend can be added, updated or changed without significant modifications needed elsewhere. The segregation also allows modifications of the layout of the results independently of the rest of the TEACH system. The analysis tools pass their results to the feedback module, thus allowing it to concentrate solely on the presentation of the results to the student. Each of the result objects is processed, with each error being written to the HTML output to be presented to the student, an example of the output can be seen in Figure 16.



Question: Sucessfully divide 2 integers to get a real number

Please start your statements in a new line (including braces)

Program:

```
/**
   division.java
   @author: David Burela
   @version 1 (September 2006)

   **Please compile as is before modifying
*/

public class Division
{

   public static void main(String [] args)
   {
      int varA = 5;
      int varB = 7;

      double result = varA/varB;

      System.out.println(result);
   }
}
```

Feedback: Execution

There are no compiler errors.

We detected 1 possible bug(s)

int division result cast to double
At Division.java:[line 17]

Additional information on the bugs found

int division result cast to double

This code casts the result of an integer division operation to double. Doing division on integers loses precision. The fact that the result was cast to double suggests that this precision should have been retained. What was probably meant was to cast one or both of the operands to double *before* performing the division. Here is an example:

```
int x = 2;
int y = 5;
// Wrong: yields result 0.0
double value1 =  x / y;

// Right: yields result 0.4
double value2 =  x / (double) y;
```

Program output:

```
0.0
```

Figure 16 Screenshot of TEACH: source code input(left) and the resulting feedback(right)

5. Evaluation

5.1. *What the Students Will be Testing*

5.1.1. Modules

The final system used for testing saw the students evaluating a subset of the entire system, specifically the `Checkstyle` and `FindBugs` module integrations. The expectation being that they will see the benefits of these individual modules and will respond to them.

The sample programs that have been created will be used to show the static analysis module in action. There is no specific sample program for the style checking — the rationale being that the students will encounter it under normal use of the program.

5.1.2. Code Matching

The students will not be testing the code matching and scoring module at this time. Although it was built and works satisfactorily, it is felt that there was not enough time to successfully build and test examples that would show this feature in full. The time investment required in creating one of these was better spent in creating higher quality sample questions for the static analysis module.

5.2. *Testing Session*

The students were required to spend 10–15 minutes with TEACH in order to familiarise themselves with the system. Upon completion of the session, the student would complete a survey form to provide feedback about the system and its utility.

More information about the tutorial questions is given in section 5.3, and information on the survey can be found in section 5.4.

Each testing session was done with 1–3 students at a time.The general format of the sessions followed a general outline:

- Introduce myself to the student and give a brief overview of what my aims are for TEACH.
- Provide students with the information sheet and consent form, explaining the role of each.
- Once the student had signed the consent form, TEACH was loaded in their web browser.
- The student is shown where the information webpage on how to use TEACH is located and told they may look at it after I leave them.
- The first tutorial exercise (`Division.java`) is opened, and compiled without modification to show the student the error message.
- The student is instructed to compile each tutorial without modification before attempting it.
- The student is left alone for 10–15 minutes to complete the tasks.
- A survey form is supplied to the student to complete.

5.3. *Sample Programs*

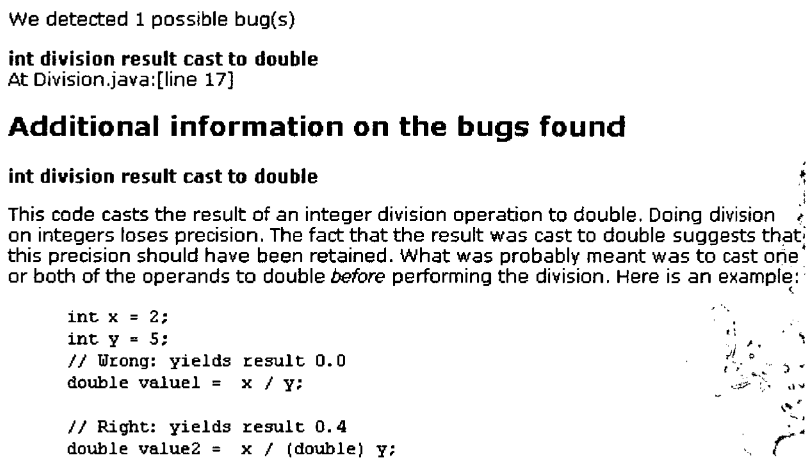
In order for the students to have a system to test, sample tutorial questions needed to be generated. These questions should successfully show the students different kinds of errors that can be detected, and also how the system can assist in their tutorial learning. The choice of sample questions is important as the students will only have limited time to interact with the system.

The final set of sample questions constructed for testing covered a range of common mistakes. The full source code for each tutorial program is included in Appendix C.

Some examples are presented in the following sub-sections.

5.3.1. Division

Tutorial question 1 was used to demonstrate a common division scenario. A beginning programmer may make the mistake of dividing 2 Integer numbers together, expecting a decimal answer. Not realising the effect of dividing 2 Integer numbers is an Integer, the student may spend a lot of wasted time trying to debug the problem. Figure 17 shows the error generated if the student tries to compile the tutorial without modification.



We detected 1 possible bug(s)

int division result cast to double
At Division.java:[line 17]

Additional information on the bugs found

int division result cast to double

This code casts the result of an integer division operation to double. Doing division on integers loses precision. The fact that the result was cast to double suggests that this precision should have been retained. What was probably meant was to cast one or both of the operands to double *before* performing the division. Here is an example:

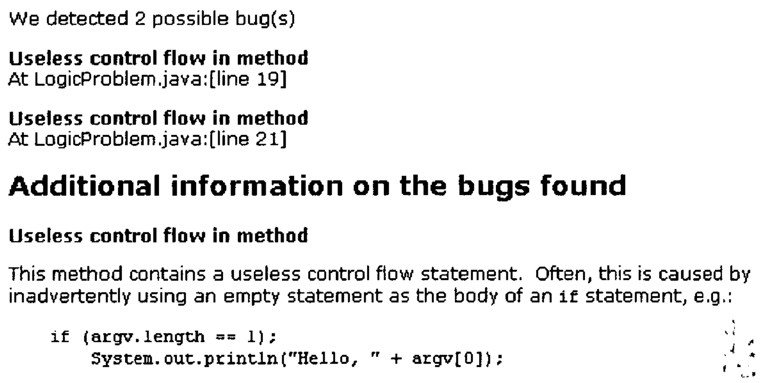
```
int x = 2;
int y = 5;
// Wrong: yields result 0.0
double value1 =  x / y;

// Right: yields result 0.4
double value2 =  x / (double) y;
```

Figure 17 Error output of tutorial question 1

5.3.2. Logic Error

Tutorial question 2 was used to demonstrate detection of a logic error. The program consists of two separate if statements with a semicolon after each. The effect of this is that the line after each if statement executes, regardless of how the conditional statement evaluated. This is a mistake that could stay hidden without being detected, TEACH highlights this error to the student.

Figure 18 shows the error generated if the student tries to compile the tutorial without modification.



We detected 2 possible bug(s)

Useless control flow in method
At LogicProblem.java:[line 19]

Useless control flow in method
At LogicProblem.java:[line 21]

Additional information on the bugs found

Useless control flow in method

This method contains a useless control flow statement. Often, this is caused by inadvertently using an empty statement as the body of an if statement, e.g.:

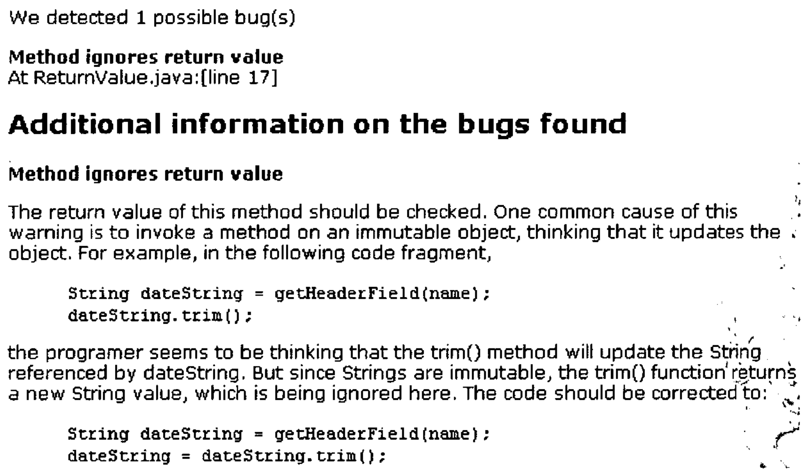
```
    if (argv.length == 1);
        System.out.println("Hello, " + argv[0]);
```

Figure 18 Error output of tutorial questsion 2

5.3.3. Ignoring Return Value

Tutorial question 3 was used to demonstrate how a student might call a method on an object, but then fail to store the results anywhere. This could be done either by simply forgetting to store the result of a call to `stringA.length()` like in the question (see Appendix C3), or it could be from not realising that the method call does not actually modify the underlying object, but returns a new object, eg. `stringA.trim()`.

Figure 19 shows the error generated if the student tries to compile the tutorial without modification.



We detected 1 possible bug(s)

Method ignores return value
At ReturnValue.java:[line 17]

Additional information on the bugs found

Method ignores return value

The return value of this method should be checked. One common cause of this warning is to invoke a method on an immutable object, thinking that it updates the object. For example, in the following code fragment,

```
    String dateString = getHeaderField(name);
    dateString.trim();
```

the programer seems to be thinking that the trim() method will update the String referenced by dateString. But since Strings are immutable, the trim() function returns a new String value, which is being ignored here. The code should be corrected to:

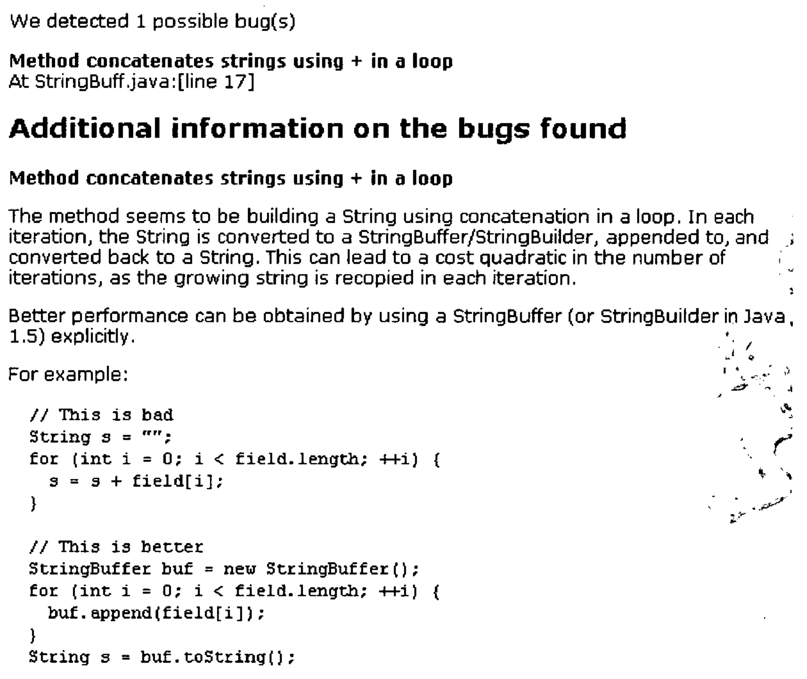
```
    String dateString = getHeaderField(name);
    dateString = dateString.trim();
```

Figure 19 Error output from tutorial question 3

5.3.4. String Buffer

Tutorial question 4 was an advanced tutorial not desigined for 1st year students but was included to demonstrate to advanced testers the depth of analysis the system can provide. The theory behind this is that it is inefficient to sequentially add elements to a `String`, as it needs to create a new `String` object each time. A more efficient approach would be to buffer up all the elements to add to the `String` and then create the final `String` object at the end.

Figure 20 shows the error generated if the student tries to compile the tutorial without modificaiton.


We detected 1 possible bug(s)

Method concatenates strings using + in a loop
At StringBuff.java:[line 17]

Additional information on the bugs found

Method concatenates strings using + in a loop

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

Better performance can be obtained by using a StringBuffer (or StringBuilder in Java 1.5) explicitly.

For example:

```
// This is bad
String s = "";
for (int i = 0; i < field.length; ++i) {
  s = s + field[i];
}

// This is better
StringBuffer buf = new StringBuffer();
for (int i = 0; i < field.length; ++i) {
  buf.append(field[i]);
}
String s = buf.toString();
```

Figure 20 Error output from question 4

5.4. Survey

After testing the system the students were required to complete a survey on the system. The survey will assist in capturing the students' reactions from their use of the system.

5.4.1. About the Questions

The questions were designed to obtain from the students testing the system, their feedback on how they would rate certain features of the system: navigation through the system, the type of feedback returned to the user, etc. The survey questions can be viewed in Appendix B.

Each question was able to be rated according to the following range

1. Strongly Disagree
2. Disagree
3. Neutral
4. Agree
5. Strongly Agree

N/A.

5.4.2. How the students were Contacted

An email message was sent out to all undergraduate and postgraduate students of the School of Computing, the contents of the email message can be seen in Appendix A. The email message contained the relevant information regarding TEACH and what would be required from students volunteering to test the system.

5.5. Ethics committee Approval

All research undertaken at the University of Tasmania that involves human participants requires approval from the University's Ethics Committee. The research that has been undertaken for this thesis was submitted to the Ethics Eommittee for approval, and was deemed to have met all requirements. It was granted approval and allocated number H9140.

6. Results and Observations

6.1. Survey Results

6.1.1. Question 1

Question 1 asked the student to evaluate the statement "*The system is intuitive to use*".

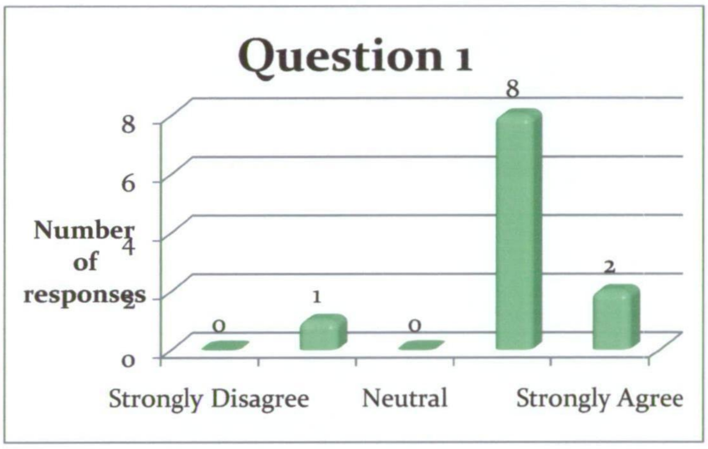


Figure 21 Responses to Question 1

From Figure 21, only a single tester of the 11 thought that the system was not intuitive to use. This is an outstanding result as the system will need to be easily picked up and used by students in a tutorial environment without much direction.

6.1.2. Question 2

Question 2 asked the student to evaluate the statement "Are the instructions clear". Pleasingly, the results obtained were synonymous with our hopes. From Figure 22, 82% of the testers felt that the instructions were clear and the remaining 18% had no strong opinion on it.

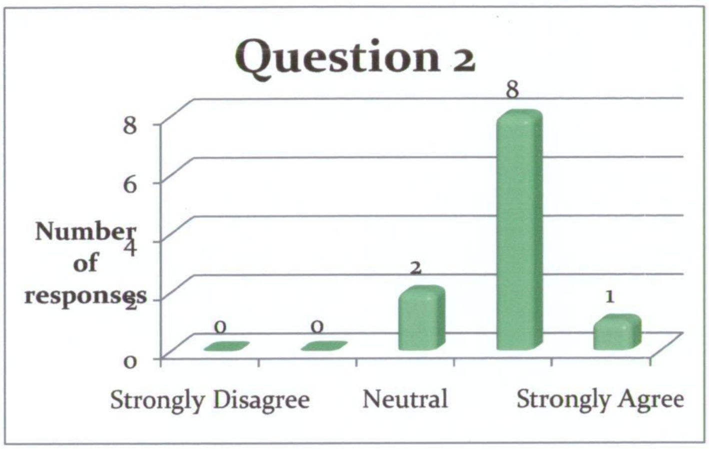


Figure 22 Responses to Question 2

6.1.3. Question 3

Question 3 asked the student to evaluate the statement *"Navigation through the different phases of the system is readily achieved."*

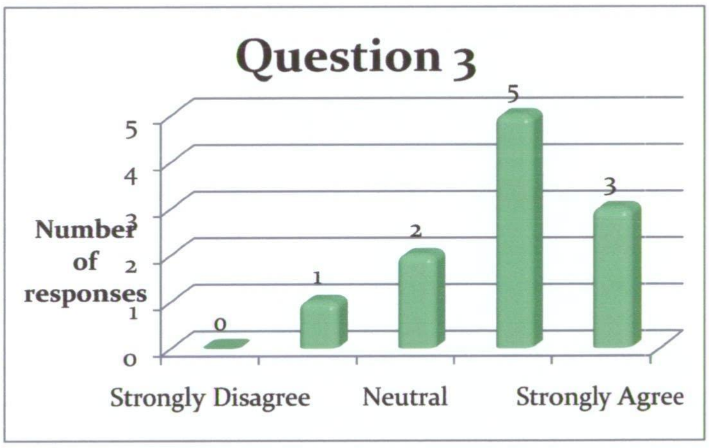


Figure 23 Responses to Question 3

From Figure 23, only a single tester felt that navigating the different phases was not readily achieved. With 72% of the testers recording favourable results it is a positive sign that TEACH is easily navigatable, but that there may be a slight room for improvement.

6.1.4. Question 4

Question 4 asked the student to evaluate the statement *"I like how compiler messages, execution output, and feedback on source code are separated on the screen"*

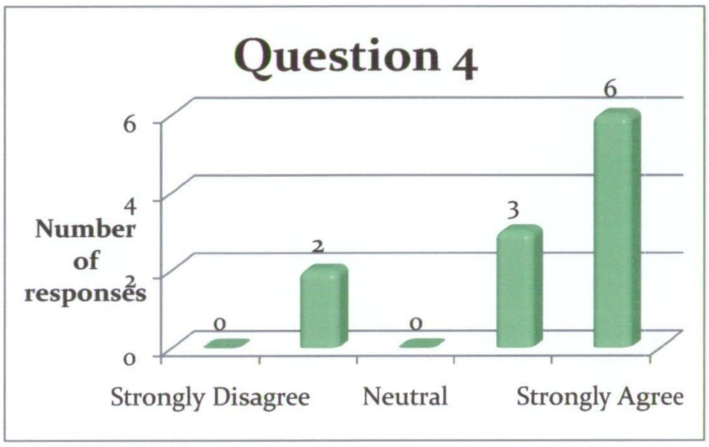


Figure 24 Responses to Question 4

From Figure 24, 82% of the students like how the compiler messages, execution output, and feedback were separated on the screen. This is an excellent response as feedback from TEACH is the most important aspect of the system.

6.1.5. Question 5

Question 5 asked the student to evaluate the statement *"Feedback from the system given to you was clear and easy to understand."*

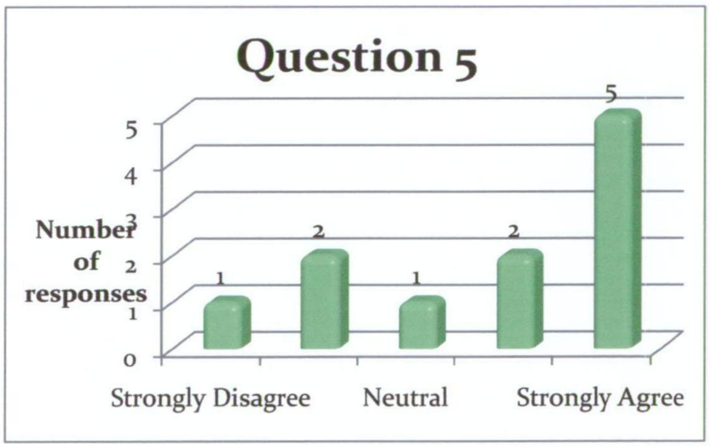


Figure 25 Responses to Question 5

From Figure 25, 64% of testers felt that the feedback from the system was clear and easy to understand, a figure which is not as high as would be liked. Clear feedback is a necessity for a code analysis tool and thus should focus strongly on refining this part of the system. The types of style checks specified to be checked for were taken from an example file included with CheckStyle, but modified to remove some of the more advanced checks. Further tweaking of the XML file will be needed to get feedback which has greater relevance to the student.

6.1.6. Question 6

Question 6 asked the student to evaluate the statement *"The integration of compiler and web browser is advantageous."*

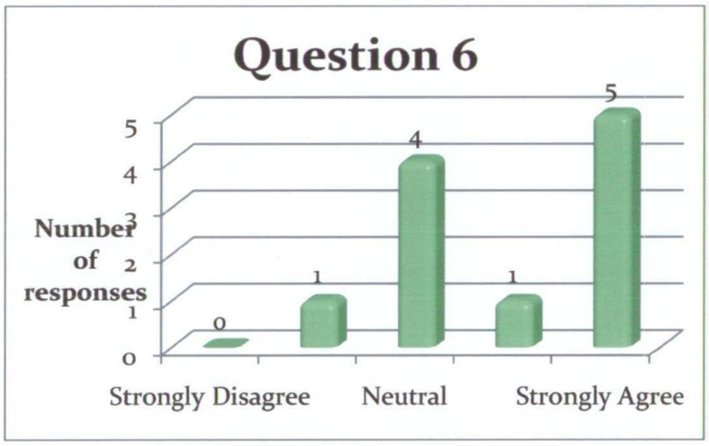


Figure 26 Responses to Question 6

From Figure 26, at 54% roughly half of the testers felt that the integration of the Java compiler with a web browser was advantageous. While this is not a favourable outcome, the compilation feature is there to simplify the workflow of the tutorials. Students indicated they did not think the course should be simplified; instead all students should be required to learn and understand how a compiler works as it is an integral part of being a programmer.

6.1.7. Question 7

Question 7 asked the student to evaluate the statement *"I think that this system would improve my learning"*. From Figure 27, 72% of testers felt that TEACH would improve their learning. This is an exceptional response, with TEACH being a learning system it is very encouraging that such a large percentage of students feel as if using TEACH would indeed improve their learning.

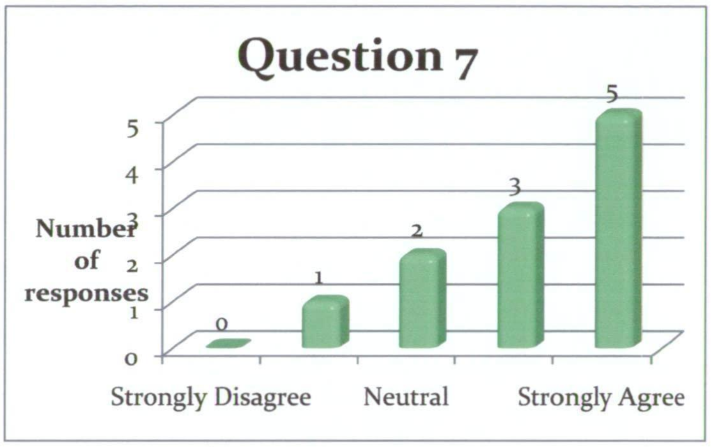


Figure 27 Responses to Question 7

6.1.8. Question 8

Question 8 asked the student to evaluate the statement *"I think that this project is useful."* From Figure 28, 91% of testers felt that this research project was useful. This gives standing to continue the project in subsequent years.

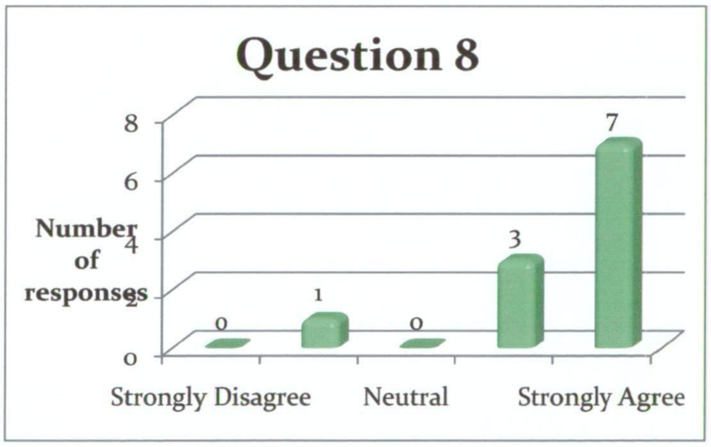


Figure 28 Responses to Question 8

6.1.9. Question 9

Question 9 asked the student to evaluate the statement *"I would like to use the system."*

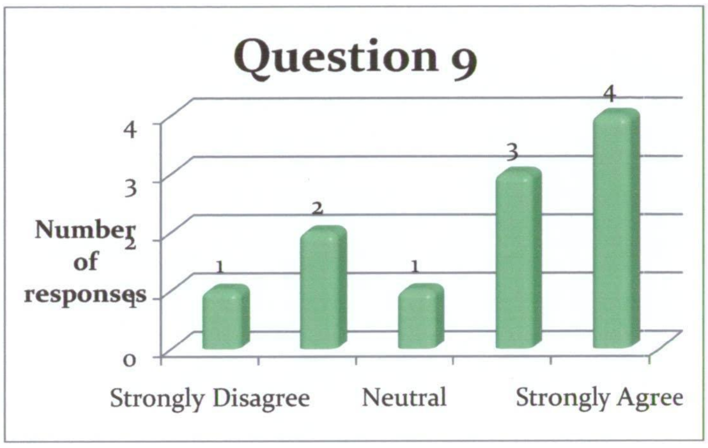


Figure 29 Responses to Question 9

From Figure 29, 64% of testers felt they would like to use the system, the results while being favourable are not as agreeable as hoped. This makes for an interesting observation as previously in Question 7 students had responded they think TEACH would help improve their learning. The question may have generated this response as a result of students thinking the system would be suitable for other students but not for themselves. Further work needs to be to uncover the reasoning behind this response.

6.1.10. Question 10

Question 8 asked the student to evaluate the statement *"You are happier with this system compared to the traditional tutorial system."* From Figure 30, with only 36% of testers responding that they are happier with this system compared to the traditional tutorial system, the response does look promising. However when the result is looked at in context of the freeform comments at the end of the survey the reasons become clearer. Students suggested that instead of just using this system in isolation, TEACH should instead be integrated with the current tutorial system to enhance them rather than replace them outright.

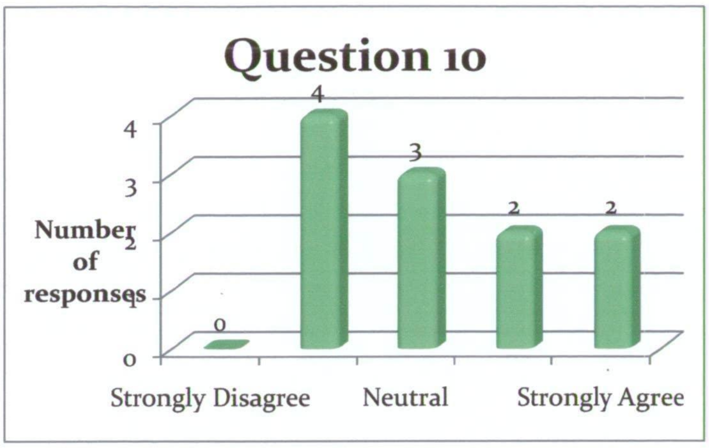


Figure 30 Responses to Question 10

6.1.11. Freeform comments

The last questions on the survey allowed the students to provide feedback on which parts of the system they liked, disliked or would like to change. Many of the suggestions were cosmetic changes to the system and these have been discussed under further work.

7. Conclusion

This work began by presenting the hypothesis stating students will prefer learning in the newly developed automated tutorial system over the current tutorial system with the aim of building and evaluating the effectiveness a web-based tutorial system. A review of the literature was then offered describing the context of each technology and tool considered in this thesis. An exploration of the tools available for use by the system was presented with justifications on why the each of the final tools was chosen. The "Three Spheres of Analysis" was introduced and the role explained in the context of the work. A detailed description was given on how the system was implemented. Finally the system was tested by students and feedback collected for analysis with the results exhibited and discussed.

The results show that the students were pleased with the system with positive feedback given on the intuitiveness of the system, clear instructions, navigation through the system and separation of feedback given by the system. Students indicated that they thought the system would improve their learning and also revealed that they felt the project was useful.

Despite being pleased with the system and stating the system would improve their learning, the numbers were divided over the decision that they would prefer this new style of tutorial learning over the traditional method. The results leave the hypothesis in an inconclusive proof or disproof, however from the results a new option presents itself: the students' suggestion that the new system be integrated with the current tutorial system rather than simply replacing it. This new option of integrating the new system with the current tutorial system should be explored in a future research project.

7.1. Future Work

7.1.1. Security Issues

TEACH was built with the aim of testing the hypothesis of this research; thought was not put into security concerns. Investigating ways to make TEACH more secure would be advantageous before being rolling out to students. One of the students in a testing session displayed how they were able to execute any application on the server, including "Format C:".

7.1.2. Infinite Loops

The student's code is compiled and then executed on the server, this causes an issue if the student has accidentally or maliciously put an infinite for loop into the code. The offending Java program will run on the server indefinitely consuming resources. An area of improvement would be to allow Java programs to run up to a determined length of time before being forcefully shutdown.

7.1.3. Code Input Enhancements

Feedback from the students indicated they would appreciate some graphical and functional enhancements to the code input area. Suggestions on enhancements included colouring of Java keywords similar to that provided by professional IDEs to assist in making the code readable. Placing line numbers was another suggestion which would assist in locating where a particular error had occurred.

7.1.4. Showing the Error in Context

Students requested that they be able to see the reported errors in their context of the surrounding code, instead of a line number being returned. Knowing where the error occurred in context of the surrounding code may be as important as the actually reported error.

7.1.5. Background Processing

An interesting area of research would involve modifying TEACH to continuously detect errors without the student having to submit the code manually. This could be implemented similar to automated spell checkers in word processors with erroneous sections of code highlighted after being written.

7.1.6. Distributed Processing

Enhancements could be applied to TEACH making it a distributed system. Different analysis spheres could be assigned to dedicated servers potentially reducing the analysis time of TEACH whilst making it scalable.

7.1.7. Dynamic Analysis

Integrating dynamic analysis tools into TEACH may be useful for teaching more advanced students, this would be dependant on the teaching structure of the School of Computing.

8. References

Amit, C. (2005). "Java & Static Analysis." Dr. Dobb's Journal; San Mateo **30**(7): 25,27.

Bandera. (2006). "Bandera." Retrieved 2006/08/20, from http://bandera.projects.cis.ksu.edu.

Bray, T., J. Paoli, et al. (1997). "Extensible Markup Language (XML)." World Wide Web Journal **2**(4): 27-66.

CheckStyle. (2006). "CheckStyle Available Checks." Retrieved 2006/09/06, from http://checkstyle.sourceforge.net/availablechecks.html.

CheckStyle. (2006). "CheckStyle Homepage." Retrieved 2006/07/18, from http://checkstyle.sourceforge.net/.

Clark, M. (2006, 2006/02/20). "JUnit FAQ." Retrieved 2006/08/03, from http://junit.sourceforge.net/doc/faq/faq.htm.

Doshi, G. (2005). "JUnit 4.0 in 10 minutes." Retrieved 2006/09/04, from http://www.instrumentalservices.com/index.php?option=com_content&task=view&id=45&Itemid=52.

Dwyer, M. B., J. Hatcliff, et al. (2006). "Evaluating the Effectiveness of Slicing for Model Reduction of Concurrent Object-Oriented Programs." LECTURE NOTES IN COMPUTER SCIENCE **3920**: 73.

Edwards, S. H. (2003). "Teaching software testing: automatic grading meets test-first coding." Conference on Object Oriented Programming Systems Languages and Applications: 318-319.

FindBugs. (2006, 2006/07/10). "FindBugs™ - Find Bugs in Java Programs." Retrieved 2006/07/20, from http://findbugs.sourceforge.net/.

Free Software Foundation, Inc. (2000). "Comparing and Merging Files." Retrieved 2006/08/15, from http://www.gnu.org/software/diffutils/manual/html_mono/diff.html.

Goetz, B. (2006, 2006/06/20). Retrieved 2006/07/25, from http://www-128.ibm.com/developerworks/library/j-jtp06206.html?ca=dgr-lnxw01BugDetectors.

Goyvaerts, J. (2006). Regular Expressions: The Complete Tutorial, Lulu Press, Inc.

Hammurapi Group (2004). Hammurapi User Manual.

Hammurapi Group. (2006, 2006/07/29). "Automated code review." Retrieved 2006/06/20, from http://wiki.hammurapi.biz/index.php?title=Automated_code_review.

Hovemeyer, D. and W. Pugh (2004). "Finding bugs is easy." ACM SIGPLAN Notices **39**(12): 92-106.

Hristova, M., A. Misra, et al. (2003). "Identifying and correcting Java programming errors for introductory computer science students." Proceedings of the 34th SIGCSE technical symposium on Computer science education: 153-156.

InfoEther. (2006). "PMD." Retrieved 2006/05/22, from http://pmd.sourceforge.net.

JLint. (2006). "JLint home page." Retrieved 2006/08/07, from http://sourceforge.net/projects/jlint/.

John Lewis, W. L. (2001). Java. Software Solutions, Addison-Wesley.

Microsoft. (2006). "ASP.NET Developer Center." Retrieved 2006/10/25, from http://msdn.microsoft.com/asp.net/.

Mitchell, S. (2006). Sams Teach Yourself ASP.NET 2.0 in 24 Hours, Sams Publishing.

Raggett, D., A. Le Hors, et al. (1999). "HTML 4.01 Specification." W3C Recommendation REC-html401-19991224, World Wide Web Consortium (W3C), Dec.

Rutar, N., C. B. Almazan, et al. (2004). "A Comparison of Bug Finding Tools for Java." Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on: 245-256.

Savitch, W. (1999). JAVA. An introduction to computer science & programming, Prentice-Hall.

Sim, C. (2005). Automatic Assesment and Programming Tutor, University of Tasmania. **Bachelor of Computing with Honours.**

Sun. (1998, March). "Sun Microsystems Accessibility Program - Overview of Java." Retrieved 2006/08/07.

Sun. (2005). "JavaServer Pages Technology - Frequently Asked Questions." Retrieved 2006/08/07.

Topor, R. W. (2002). "CIT1104 Programming II: Common (Java) programming errors." Retrieved 2006/04/06.

Truong, N., P. Bancroft, et al. (2003). "A web based environment for learning to program." Proceedings of the twenty-sixth Australasian computer science conference on Conference in research and practice in information technology-Volume 16: 255-264.

Truong, N., P. Bancroft, et al. (2005). "Learning to program through the web." Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education: 9-13.

Truong, N., P. Roe, et al. (2004). "Static analysis of students' Java programs." Proceedings of the sixth conference on Australian computing education-Volume 30: 317-325.

University of Maryland (2006). Findbugs Documentation.

UTAS. (2006). "KXT101 overview." Retrieved 2006/06/20, from http://www.comp.utas.edu.au/app/outlines/index.jsp?outlineID=144.

Appendix A: Information and Consent Forms

Tutorial Enhancement and Automated Code Helper

UTAS

INFORMATION SHEET

Dr Julian Dermoudy
Degree Coordinator
School of Computing
Private Bag 100
Hobart Tas 7001

Phone 6226 2933
Email Julian.Dermoudy@utas.edu.au

David Burela
Masters Student
School of Computing

Phone 6226 2922
Email dburela@utas.edu.au

What I am trying to do

As part of my Masters Degree in Computing, I am investigating whether students will be happier to use the web-based assistive learning environment or the traditional classroom tutorial learning. The project aims to assist with computing students learning, by means of advanced logic checking, as well as style checks.

What I would like you to do

What I would value now is assistance in determining on how the students feel about using the web-based assistive learning environment and suggestions for its improvements and possible use.

If you agree to participate, you will attend the School of Computing's PC laboratories and use the system for up to 30 minutes at a time of your choice between the 12th & 13th of October and the 19th & 20th (between 10am to 6pm), and then fill out a survey based on your experiences with the system (which should take 5 to 10 minutes). The system will request you to select and enter source code into the given text area. The source code will then be submitted to the server to check for logic, compilation and style errors, following which feedback will be given to you. The goal of the system is to provide useful feedback to students so that they can improve their programming.

How will the data be used?

Your survey responses form the data for my study. As the survey does not request any identifying information, your responses are anonymous. It follows that you will not be identifiable in the research output.

Voluntariness

Your participation is entirely voluntary, and not part of course requirements. You can withdraw from this research at any time, without any effect. If you would like to participate in this study, please complete and return a signed copy of the attached *Consent Form* to Mrs Andrea Kingston (the Secretary of the School of Computing). Please note that the consent forms will be stored separately from the survey responses, and will not be linked to them (in order to preserve anonymity). The research data will be securely stored at the School of Computing for 5 years before being destroyed.

Ethics approval

This project has received ethical approval from the Human Research Ethics Committee (Tasmania) Network (Approval No. H9140) and the School of Computing. If you have any concerns of an ethical nature or complaints about the manner in which the project is conducted, you may contact the Executive Officer of the Network, Marilyn Pugsley, Ph 6226 7479; email: Marilyn.Pugsley@utas.edu.au University students may also discuss any concerns regarding this project confidentially with the University Student Counsellor, Mark Hood, phone (03) 6324 3787.

Thank you for taking the time to read this information sheet. I hope you will be willing to participate in this study. If you would like to receive a summary of the findings of this study, please contact me or my supervisor, Dr Julian Dermoudy (contact details supplied at the top of this information sheet).

Tutorial Enhancement and Automated Code Helper

UTAS

CONSENT FORM

An investigation of whether students will be happier to use the web-based assistive learning environment or the traditional classroom tutorial learning.

1. I (the participant) have read the information sheet and any questions I have asked have been answered to my satisfaction.

2. The nature and possible effects of the study have been explained to me.

3. I understand that the study involves selecting a question, completing the sourcecode and trying to remove all errors, and completing a survey, which should take no more than 40 minutes in total.

4. I understand that all research data will be securely stored on University of Tasmania premises for 5 years before being destroyed.

5. I agree that research data gathered for the study may be published provided I am not identifiable as a participant.

6. I understand that the researchers will maintain my identity as a participant in this study confidential and that any information obtained by the researcher will be used only for the purpose of the research.

7. I agree to participate in this research and understand that I may withdraw at any time without any effects.

Name of participant ____
Signature of participant ____________________________ Date ______________

Statement by researcher:
I, the researcher, have explained this project and the implications of participation in it to this volunteer and I believe that the consent is informed and that he/she understands the implications of participation.

Name of researcher ____
Signature of researcher ____________________________ Date ______________

Appendix B: Survey Form

Tutorial Enhancement and Automated Code Helper

UTAS

SURVEY FORM

Please indicate your choice by circling the appropriate number:

1 – Strongly Disagree 2 – Disagree 3 – Neutral 4 – Agree 5 – Strongly Agree

01)	The system is intuitive to use.	**1**	**2**	**3**	**4**	**5**	**N/A**
02)	The instructions are clear.	**1**	**2**	**3**	**4**	**5**	**N/A**
03)	Navigation through the different phases of the system is readily achieved.	**1**	**2**	**3**	**4**	**5**	**N/A**
04)	I like how compiler messages, execution output, and feedback on source code are separated on the screen.	**1**	**2**	**3**	**4**	**5**	**N/A**
05)	Feedback from the system given to you was clear and easy to understand.	**1**	**2**	**3**	**4**	**5**	**N/A**
06)	The integration of compiler and web browser is advantageous.	**1**	**2**	**3**	**4**	**5**	**N/A**
07)	I think that this system would improve my learning	**1**	**2**	**3**	**4**	**5**	**N/A**
08)	I think that this project is useful.	**1**	**2**	**3**	**4**	**5**	**N/A**
09)	I would like to use the system.	**1**	**2**	**3**	**4**	**5**	**N/A**
10)	You are happier with this system compared to the traditional tutorial system.	**1**	**2**	**3**	**4**	**5**	**N/A**

Continued....

11) The best aspect of the system is...

12) The worst aspect of the system is...

13) If I could, I would add/change/remove...

14) Any other comments:

Thank you for your time and valuable feedback!

Have a great day!

Appendix C – Tutorial Questions

C1. Tutorial Question 1 – Division.java

```
/**

/**
    Division.java
    @author: David Burela
    @version 1 (September 2006)

    **Please compile as is before modifying
*/

public class Division
{

    public static void main(String [] args)
    {
        int varA = 5;
        int varB = 7;

        double result = varA/varB;

        System.out.println(result);
    }
}
```

C2. Tutorial Question 2 – LogicProblem.java

```
/**
    LogicProblem.java
    @author: David Burela
    @version 1 (September 2006)

    **Please compile as is before modifying
    This program should display if a number is greater than
    or less than 5.
    Bonus marks for displaying if it is equal to 5
*/

public class LogicProblem
{

    public static void main(String [] args)
    {
        int number = 7;

        if(number < 5);
            System.out.println(number + " is Less than 5");
        if(number > 5);
            System.out.println(number + " is Greater than
5");

    }
}
```

C3. Tutorial Question 3 – ReturnValue.java

```
/**
    ReturnValue.java
    @author: David Burela
    @version 1 (September 2006)

    **Please compile as is before modifying
    This program should return the length of your name.
*/

public class ReturnValue
{

    public static void main(String [] args)
    {
        String name = "Your name";        //Place your name
here
        int namelen = 0;
        name.length();

        System.out.println("your name is " + namelen +"
long!");

    }
}
```

C4. Tutorial Question 4 – StringBuff.java

```
/**
    StringBuff.java
    @author: David Burela
    @version 1 (September 2006)

    This program adds 10 ":"s to a string
*/

public class StringBuff
{

    public static void main(String [] args)
    {
        String buffer= "";
        for (int i=0; i < 10; i++)
        {
            buffer = buffer + ":";
        }

        System.out.println(buffer);
    }
}
```

C5. Tutorial Question 5 – W2Swap.java

```
/**
    W2Swap.java
    @author: Robyn Gibson
    @version 1 (February 2003)

    This program should swap the values of varA and varB
    and print out the newly swapped values
*/

public class W2Swap
{

    public static void main(String [] args)
    {
        int varA = 5;
        int varB = 7;
        // create int storage for temp variable

        //display varA and varB
        System.out.println("varA current value - " + varA);
        System.out.println("varB current value - " + varB);

        // temp becomes varA

        // varA becomes varB

        // varB becomes temp

        //display varA
        //display varB

    }
}
```

Appendix D: Reference of Available CheckStyle Checks

Checkstyle provides many checks that you can apply to your sourcecode, below is an alphabetical reference taken from (CheckStyle 2006).

AbstractClassName	Ensures that the names of abstract classes conforming to some regular expression.
AnonInnerLength	Checks for long anonymous inner classes.
ArrayTrailingComma	Checks if array initialization contains optional trailing comma.
ArrayTypeStyle	Checks the style of array type definitions.
AvoidInlineConditionals	Detects inline conditionals.
AvoidNestedBlocks	Finds nested blocks.
AvoidStarImport	Check that finds import statements that use the * notation.
BooleanExpressionComplexity	Restricts nested boolean operators (&&, || and ^) to a specified depth (default = 3).
ClassDataAbstractionCoupling	This metric measures the number of instantiations of other classes within the given class.
ClassFanOutComplexity	The number of other classes a given class relies on.
ConstantName	Checks that constant names conform to a format specified by the format property.
CovariantEquals	Checks that if a class defines a covariant method equals, then it defines method equals(java.lang.Object).
CyclomaticComplexity	Checks cyclomatic complexity against a specified limit.
DeclarationOrder	Checks that the parts of a class or interface declaration appear in the order suggested by the Code Conventions for the Java Programming Language.
DefaultComesLast	Check that the default is after all the cases in a switch statement.
DescendantToken	Checks for restricted tokens beneath other tokens.
DesignForExtension	Checks that classes are designed for inheritance.
DoubleCheckedLocking	Detect the double-checked locking idiom, a technique that tries to avoid synchronization overhead but is incorrect because of subtle artifacts of the java memory model.
EmptyBlock	Checks for empty blocks.
EmptyForInitializerPad	Checks the padding of an empty for initializer; that is whether a space is required at an empty for initializer, or such spaces are forbidden.

EmptyForIteratorPad	Checks the padding of an empty for iterator; that is whether a space is required at an empty for iterator, or such spaces are forbidden.
EmptyStatement	Detects empty statements (standalone ';').
EntityBean	Checks that an EntityBean implementation satisfies EntityBean requirements.
EqualsHashCode	Checks that classes that override equals() also override hashCode().
ExecutableStatementCount	Restricts the number of executable statements to a specified limit (default = 30).
ExplicitInitialization	Checks if any class or object member explicitly initialized to default for its type value (null for object references, zero for numeric types and char and false for boolean.
FallThrough	Checks for fall through in switch statements Finds locations where a case contains Java code - but lacks a break, return, throw or continue statement.
FileLength	Checks for long source files.
FinalClass	Checks that class which has only private ctors is declared as final.
FinalLocalVariable	Ensures that local variables that never get their values changed, must be declared final.
FinalParameters	Check that method/constructor/catch/foreach parameters are final.
FinalStatic	Checks that all static fields are declared final.
GenericIllegalRegexp	A generic check for code problems, the user can search for any pattern.
Header	Checks the header of the source against a fixed header file.
HiddenField	Checks that a local variable or a parameter does not shadow a field that is defined in the same class.
HideUtilityClassConstructor	Make sure that utility classes (classes that contain only static methods) do not have a public constructor.
IllegalCatch	Catching java.lang.Exception, java.lang.Error or java.lang.RuntimeException is almost never acceptable.
IllegalImport	Checks for imports from a set of illegal packages.
IllegalInstantiation	Checks for illegal instantiations where a factory method is preferred.
IllegalThrows	Throwing java.lang.Error or java.lang.RuntimeException is almost never acceptable.
IllegalToken	Checks for illegal tokens.

IllegalTokenText	Checks for illegal token text.
IllegalType	Checks that particular class are never used as types in variable declarations, return values or parameters.
ImportControl	Check that controls what packages can be imported in each package.
ImportOrder	Class to check the ordering/grouping of imports.
Indentation	Checks correct indentation of Java Code.
InnerAssignment	Checks for assignments in subexpressions, such as in String s = Integer.toString(i = 2);.
InterfaceIsType	Implements Bloch, Effective Java, Item 17 - Use Interfaces only to define types.
JUnitTestCase	Ensures that the setUp(), tearDown()methods are named correctly, have no arguments, return void and are either public or protected.
JavaNCSS	This check calculates the Non Commenting Source Statements (NCSS) metric for java source files and methods.
JavadocMethod	Checks the Javadoc of a method or constructor.
JavadocStyle	Custom Checkstyle Check to validate Javadoc.
JavadocType	Checks the Javadoc of a type.
JavadocVariable	Checks that a variable has Javadoc comment.
LeftCurly	Checks the placement of left curly braces on types, methods and other blocks:
LineLength	Checks for long lines.
LocalFinalVariableName	Checks that local final variable names conform to a format specified by the format property.
LocalHomeInterface	Checks the local home interface requirements: every method must not throw the java.rmi.RemoteException Reference: Enterprise JavaBeansTM Specification,Version 2.0, section 9.6.2.
LocalInterface	Checks the methods of a local interface.
LocalVariableName	Checks that local, non-final variable names conform to a format specified by the format property.
MagicNumber	Checks for magic numbers.
MemberName	Checks that instance variable names conform to a format specified by the format property.
MessageBean	Checks that a MessageBean implementation satisfies MessageBean requirements.
MethodLength	Checks for long methods.

MethodName	Checks that method names conform to a format specified by the format property.
MethodParamPad	Checks the padding between the identifier of a method definition, constructor definition, method call, or constructor invocation; and the left parenthesis of the parameter list.
MissingCtor	Checks that classes (except abstract one) define a ctor and don't rely on the default one.
MissingSwitchDefault	Checks that switch statement has "default" clause.
ModifiedControlVariable	Check for ensuring that for loop control variables are not modified inside the for block.
ModifierOrder	Checks that the order of modifiers conforms to the suggestions in the Java Language specification, sections 8.1.1, 8.3.1 and 8.4.3.
MultipleStringLiterals	Checks for multiple occurrences of the same string literal within a single file.
MultipleVariableDeclarations	Checks that each variable declaration is in its own statement and on its own line.
MutableException	Ensures that exceptions (defined as any class name conforming to some regular expression) are immutable.
NPathComplexity	Checks the npath complexity against a specified limt (default = 200).
NeedBraces	Checks for braces around code blocks.
NestedIfDepth	Restricts nested if-else blocks to a specified depth (default = 1).
NestedTryDepth	Restricts nested try-catch-finally blocks to a specified depth (default = 1).
NewlineAtEndOfFile	Checks that there is a newline at the end of each file.
NoWhitespaceAfter	Checks that there is no whitespace after a token.
NoWhitespaceBefore	Checks that there is no whitespace before a token.
OperatorWrap	Checks line wrapping for operators.
PackageDeclaration	Ensures there is a package declaration.
PackageHtml	Checks that all packages have a package documentation.
PackageName	Checks that package names conform to a format specified by the format property.
ParameterAssignment	Disallow assignment of parameters.
ParameterName	Checks that parameter names conform to a format specified by the format property.
ParameterNumber	Checks the number of parameters that a method or constructor has.

ParenPad	Checks the padding of parentheses; that is whether a space is required after a left parenthesis and before a right parenthesis, or such spaces are forbidden, with the exception that it does not check for padding of the right parenthesis at an empty for iterator.
RedundantImport	Checks for imports that are redundant.
RedundantModifier	Checks for redundant modifiers in interface and annotation definitions.
RedundantThrows	Checks for redundant exceptions declared in throws clause such as duplicates, unchecked exceptions or subclasses of another declared exception.
Regexp	A check that makes sure that a specified pattern exists (or not) in the file.
RegexpHeader	Checks the header of the source against a header file that contains a
RemoteHomeInterface	Checks the methods of a remote home interface.
RemoteInterface	Checks the methods of a remote interface.
RequireThis	Checks that code doesn't rely on the "this" default.
RequiredRegexp	A check that makes sure that a specified pattern exists in the code.
ReturnCount	Restricts return statements to a specified count (default = 2).
RightCurly	Checks the placement of right curly braces.
SessionBean	Checks that a SessionBean implementation satisfies SessionBean requirements.
SimplifyBooleanExpression	Checks for overly complicated boolean expressions.
SimplifyBooleanReturn	Checks for overly complicated boolean return statements.
StaticVariableName	Checks that static, non-final variable names conform to a format specified by the format property.
StrictDuplicateCode	Performs a line-by-line comparison of all code lines and reports duplicate code if a sequence of lines differs only in indentation.
StringLiteralEquality	Checks that string literals are not used with == or !=.
SuperClone	Checks that an overriding clone() method invokes super.clone().
SuperFinalize	Checks that an overriding finalize() method invokes super.finalize().
TabCharacter	Reports tab characters ('\t') in the source code.
ThisParameter	Checks that 'this' is not a parameter of any method calls or constructors for a bean.

ThisReturn	Checks that 'this' is not returned by a bean method.
ThrowsCount	Restricts throws statements to a specified count (default = 1).
TodoComment	A check for TODO comments.
TrailingComment	The check to ensure that requires that comments be the only thing on a line.
Translation	The TranslationCheck class helps to ensure the correct translation of code by checking property files for consistency regarding their keys.
TypeName	Checks that type names conform to a format specified by the format property.
TypecastParenPad	Checks the padding of parentheses for typecasts.
UncommentedMain	Detects uncommented main methods.
UnnecessaryParentheses	Checks if unnecessary parentheses are used in a statement or expression.
UnusedImports	Checks for unused import statements.
UpperEll	Checks that long constants are defined with an upper ell.
VisibilityModifier	Checks visibility of class members.
WhitespaceAfter	Checks that a token is followed by whitespace, with the exception that it does not check for whitespace after the semicolon of an empty for iterator.
WhitespaceAround	Checks that a token is surrounded by whitespace.
WriteTag	Outputs a JavaDoc tag as information.

Appendix E: FindBugs Bug Descriptions

The standard bugs found by FindBugs version 1.0.0 taken from (University of Maryland 2006).

Description	Category
AM: Creates an empty jar file entry	Correctness
AM: Creates an empty zip file entry	Correctness
BC: Impossible cast	Correctness
BC: instanceof will always return false	Correctness
BIT: Incompatible bit masks	Correctness
BIT: Incompatible bit masks	Correctness
BIT: Incompatible bit masks	Correctness
BIT: Bitwise OR of signed byte value	Correctness
BOA: Class overrides a method implemented in super class Adapter wrongly	Correctness
CN: Class implements Cloneable but does not define or use clone method	Correctness
CN: clone method does not call super.clone()	Correctness
Co: Abstract class defines covariant compareTo() method	Correctness
Co: Covariant compareTo() method defined	Correctness
DE: Method might drop exception	Correctness
DE: Method might ignore exception	Correctness
DLS: Overwritten increment	Correctness
DMI: Passes a constant value for a month outside of the expected range of 0..11	Correctness
DMI: hasNext method invokes next	Correctness
DMI: Code contains a hard coded reference to an absolute pathname	Correctness
DMI: Non serializable object written to ObjectOutput	Correctness
DMI: Invocation of substring(0), which returns the original value	Correctness
DP: Classloaders should only be created inside doPrivileged block	Correctness
DP: Method invoked that should be only be invoked inside a doPrivileged block	Correctness
Dm: Can't use reflection to check for presense of annotation with default retention	Correctness
Dm: Method invokes System.exit(...)	Correctness
Dm: Method invokes runFinalizersOnExit, one of the most dangerous methods in the Java libraries.	Correctness

EC: equals() used to compare array and nonarray	Correctness
EC: Invocation of equals() on an array, which is equivalent to ==	Correctness
EC: Call to equals() with null argument	Correctness
EC: Call to equals() comparing unrelated class and interface	Correctness
EC: Call to equals() comparing different interface types	Correctness
EC: Call to equals() comparing different types	Correctness
ES: Comparison of String objects using == or !=	Correctness
Eq: Abstract class defines covariant equals() method	Correctness
Eq: Covariant equals() method defined	Correctness
Eq: Covariant equals() method defined, Object.equals(Object) inherited	Correctness
FE: Test for floating point equality.	Correctness
FI: Explicit invocation of finalizer	Correctness
FI: Finalizer does not call superclass finalizer	Correctness
FI: Finalizer nullifies superclass finalizer	Correctness
HE: Class defines equals() but not hashCode()	Correctness
HE: Class defines equals() and uses Object.hashCode()	Correctness
HE: Class defines hashCode() but not equals()	Correctness
HE: Class defines hashCode() and uses Object.equals()	Correctness
HE: Class inherits equals() and uses Object.hashCode()	Correctness
IA: Ambiguous invocation of either an inherited or outer method	Correctness
IC: Initialization circularity	Correctness
ICAST: Integer shift by an amount not in the range 0..31	Correctness
ICAST: int division result cast to double	Correctness
ICAST: int value cast to double and then passed to Math.ceil	Correctness
IJU: TestCase has no tests	Correctness
IJU: TestCase implements setUp but doesn't call super.setUp()	Correctness
IJU: TestCase implements a suite method, but this method is not static and should be	Correctness
IJU: TestCase implements tearDown but doesn't call super.tearDown()	Correctness
IL: A container is added to itself.	Correctness
IL: An apparent infinite recursive loop.	Correctness
IM: Integer multiply of result of integer remainder	Correctness
IMSE: Dubious catching of IllegalMonitorStateException	Correctness
INT: Integer remainder modulo 1	Correctness
INT: Vacuous comparison of integer value	Correctness
IP: A parameter is dead upon entry to a method but overwritten	Correctness

ISC: Needless instantiation of class that only supplies static methods	Correctness
It: Iterator next() method can't throw NoSuchElement exception	Correctness
J2EE: Store of non serializable object into HttpSession	Correctness
JCIP: Fields of immutable classes should be final	Correctness
MF: Class defines field that obscures a superclass field	Correctness
MF: Method defines a variable that obscures a field	Correctness
NP: Null pointer dereference in method	Correctness
NP: Null pointer dereference in method on exception path	Correctness
NP: Immediate dereference of the result of readLine()	Correctness
NP: Method call passes null to a parameter declared @NonNull	Correctness
NP: Method may return null, but is declared @NonNull	Correctness
NP: A known null value is checked to see if it is an instance of a type	Correctness
NP: Possible null pointer dereference in method	Correctness
NP: Possible null pointer dereference in method on exception path	Correctness
NP: Possible null pointer dereference due to return value of called method	Correctness
NP: Method call passes null for unconditionally dereferenced parameter	Correctness
NP: Method call passes null for unconditionally dereferenced parameter	Correctness
NP: Non-virtual method call passes null for unconditionally dereferenced parameter	Correctness
NP: Store of null value into field annotated NonNull	Correctness
NP: Read of unwritten field	Correctness
NS: Questionable use of non-short-circuit logic	Correctness
Nm: Class defines equal(); should it be equals()?	Correctness
Nm: Confusing method names	Correctness
Nm: Class defines hashcode(); should it be hashCode()?	Correctness
Nm: Class defines tostring(); should it be toString()?	Correctness
Nm: Apparent method/constructor confusion	Correctness
Nm: Very confusing method names	Correctness
ODR: Method may fail to close database resource	Correctness
ODR: Method may fail to close database resource on exception	Correctness
OS: Method may fail to close stream	Correctness
OS: Method may fail to close stream on exception	Correctness
QBA: Method assigns boolean literal in boolean expression	Correctness

QF: Complicated, subtle or wrong increment in for-loop	Correctness
RC: Suspicious reference comparison	Correctness
RCN: Redundant comparison of non-null value to null	Correctness
RCN: Redundant comparison of two null values	Correctness
RCN: Redundant nullcheck of value known to be non-null	Correctness
RCN: Redundant nullcheck of value known to be null	Correctness
RCN: Nullcheck of value previously dereferenced	Correctness
RE: Invalid syntax for regular expression	Correctness
RE: "." used for regular expression	Correctness
RR: Method ignores results of InputStream.read()	Correctness
RR: Method ignores results of InputStream.skip()	Correctness
RV: Random value from 0 to 1 is coerced to the integer 0	Correctness
RV: Method checks to see if result of String.indexOf is positive	Correctness
RV: Method discards result of readLine after checking if it is nonnull	Correctness
RV: Remainder of 32-bit signed random integer	Correctness
RV: Method ignores return value	Correctness
SA: Self assignment of field	Correctness
SI: Static initializer for class creates instance before all static final fields assigned	Correctness
SIO: Unnecessary type check done using instanceof operator	Correctness
SQL: Method attempts to access a prepared statement parameter with index 0	Correctness
SQL: Method attempts to access a result set field with index 0	Correctness
SQL: Nonconstant string passed to execute method on an SQL statement	Correctness
SQL: A prepared statement is generated from a nonconstant String	Correctness
STI: Unneeded use of currentThread() call, to call interrupted()	Correctness
STI: Static Thread.interrupted() method is mistakenly attempted to be called on an arbitrary Thread object	Correctness
SW: Certain swing methods should only be invoked from the Swing event thread	Correctness
Se: Non-transient non-serializable instance field in serializable class	Correctness
Se: Non-serializable value stored into instance field of a serializable class	Correctness
Se: Method must be private in order for serialization to work	Correctness
Se: serialVersionUID isn't final	Correctness

Se: serialVersionUID isn't long	Correctness
Se: serialVersionUID isn't static	Correctness
Se: Class is Serializable but its superclass doesn't define a void constructor	Correctness
Se: Class is Externalizable but doesn't define a void constructor	Correctness
SnVI: Class is Serializable, but doesn't define serialVersionUID	Correctness
UCF: Useless control flow in method	Correctness
UI: Usage of GetResource may be unsafe if class is extended	Correctness
UMAC: Uncallable method defined in anonymous class	Correctness
UR: Uninitialized read of field in constructor	Correctness
UwF: Field only ever set to null	Correctness
UwF: Unwritten field	Correctness
VA: Primitive array passed to function expecting a variable number of object arguments	Correctness
Dm: Method invokes dubious String.toUpperCase() or String.toLowerCase; use the Locale parameterized version instead	Internationalization
EI: Method may expose internal representation by returning reference to mutable object	Malicious code vulnerability
EI2: Method may expose internal representation by incorporating reference to mutable object	Malicious code vulnerability
FI: Finalizer should be protected, not public	Malicious code vulnerability
MS: Method may expose internal static state by storing a mutable object into a static field	Malicious code vulnerability
MS: Field isn't final and can't be protected from malicious code	Malicious code vulnerability
MS: Public static method may expose internal representation by returning array	Malicious code vulnerability
MS: Field should be both final and package protected	Malicious code vulnerability
MS: Field is a mutable array	Malicious code vulnerability
MS: Field is a mutable Hashtable	Malicious code vulnerability
MS: Field should be moved out of an interface and made package protected	Malicious code vulnerability
MS: Field should be package protected	Malicious code vulnerability
MS: Field isn't final but should be	Malicious code

	vulnerability
DC: Possible double check of field	Multithreaded correctness
Dm: Monitor wait() called on Condition	Multithreaded correctness
Dm: A thread was created using the default empty run method	Multithreaded correctness
ESync: Empty synchronized block	Multithreaded correctness
IS: Inconsistent synchronization	Multithreaded correctness
IS: Field not guarded against conconcurrent access	Multithreaded correctness
JLM: Synchronization performed on java.util.concurrent Lock in method	Multithreaded correctness
LI: Incorrect lazy initialization of static field	Multithreaded correctness
ML: Method synchronizes on an updated field	Multithreaded correctness
MWN: Mismatched notify()	Multithreaded correctness
MWN: Mismatched wait()	Multithreaded correctness
NN: Naked notify in method	Multithreaded correctness
No: Using notify() rather than notifyAll() in method	Multithreaded correctness
RS: Class's readObject() method is synchronized	Multithreaded correctness
Ru: Invokes run on a thread (did you mean to start it instead?)	Multithreaded correctness
SC: Constructor invokes Thread.start()	Multithreaded correctness
SP: Method spins on field	Multithreaded correctness
SWL: Method calls Thread.sleep() with a lock held	Multithreaded correctness
TLW: Wait with two locks held	Multithreaded correctness
UG: Unsynchronized get method, synchronized set method	Multithreaded correctness

UL: Method does not release lock on all paths	Multithreaded correctness
UL: Method does not release lock on all exception paths	Multithreaded correctness
UW: Unconditional wait in method	Multithreaded correctness
VO: A volatile reference to an array doesn't treat the array elements as volatile	Multithreaded correctness
WS: Class's writeObject() method is synchronized but nothing else is	Multithreaded correctness
Wa: Condition.await() not in loop in method	Multithreaded correctness
Wa: Wait not in loop in method	Multithreaded correctness
Dm: Method invokes dubious Boolean constructor; use Boolean.valueOf(...) instead	Performance
Dm: Method allocates a boxed primitive just to call toString	Performance
Dm: Explicit garbage collection; extremely dubious except in benchmarking code	Performance
Dm: Method allocates an object, only to get the class object	Performance
Dm: Use the nextInt method of Random rather than nextDouble to generate a random integer	Performance
Dm: Method invokes dubious new String(String) constructor; just use the argument	Performance
Dm: Method invokes dubious String.equals(""); use String.length() == 0 instead	Performance
Dm: Method invokes toString() method on a String; just use the String	Performance
Dm: Method invokes dubious new String() constructor; just use ""	Performance
FI: Empty finalizer should be deleted	Performance
FI: Finalizer does nothing but call superclass finalizer	Performance
ITA: Method uses toArray() with zero-length array argument	Performance
SBSC: Method concatenates strings using + in a loop	Performance
SIC: Should be a static inner class	Performance
SIC: Could be refactored into a named static inner class	Performance
SIC: Could be refactored into a static inner class	Performance
SS: Unread field: should this field be static?	Performance
UPM: Private method is never called	Performance
UrF: Unread field	Performance

UuF: Unused field	Performance
WMI: Inefficient use of keySet iterator instead of entrySet iterator	Performance
BC: Questionable cast to abstract collection	Style
BC: Questionable cast to concrete collection	Style
BC: Unchecked/unconfirmed cast	Style
BC: instanceof will always return true	Style
CI: Class is final but declares protected field	Style
DB: Method uses the same code for two branches	Style
DB: Method uses the same code for two switch clauses	Style
DLS: Dead store to local variable	Style
DLS: Dead store of null to local variable	Style
DMI: Invocation of toString on an array	Style
ICAST: Unsigned right shift cast to short/byte	Style
IM: Check for oddness that won't work for negative numbers	Style
MTIA: Class extends Servlet class and uses instance variables.	Style
MTIA: Class extends Struts Action class and uses instance variables.	Style
NP: Load of known null value	Style
Nm: Class names should start with an upper case letter	Style
Nm: Class is not derived from an Exception, even though it is named as such	Style
Nm: Field names should start with an lower case letter	Style
Nm: Method names should start with an lower case letter	Style
PS: Class exposes synchronization and semaphores in its public interface.	Style
PZLA: Consider returning a zero length array rather than null	Style
REC: java.lang.Exception is caught when Exception is not thrown	Style
RI: Class implements same interface as superclass.	Style
SA: Self assignment of local variable	Style
SF: Switch statement found where one case falls thru to the next case	Style
ST: Write to static field from instance method	Style
Se: Comparator doesn't implement Serializable	Style
UM: Method calls static Math class method on a constant value	Style
UwF: Field not initialized in constructor	Style
XFB: Method directly allocates a specific implementation of xml interfaces	Style