2006

Qualitative Evaluation of the Java Intelligent Tutoring System

Edward R. Sykes
Sheridan College, ed.sykes@sheridancollege.ca

Qualitative Evaluation of the Java Intelligent Tutoring System

Edward R. Sykes
School of Applied Computing and Engineering Sciences, Sheridan College
1430 Trafalgar Road, Oakville, Ont., Canada, L6H 2L1

ABSTRACT

In an effort to support the growing trend of the Java programming language and to promote web-based personalized education, the Java Intelligent Tutoring System (JITS) was designed and developed. This tutoring system is unique in a number of ways. Most Intelligent Tutoring Systems require the teacher to author problems with corresponding solutions. JITS, on the other hand, requires the teacher to only supply the problem and problem specification. JITS is designed to "intelligently" examine the student's submitted code and determines appropriate feedback based on a number of factors such as JITS' cognitive model of the student, the student's skill level, and problem details. JITS is intended to be used by beginner programming students in their first year of College or University. This paper discusses the important aspects of the design and development of JITS, the qualitative methods and procedures, and findings. Research was conducted at the Sheridan Institute of Technology and Advanced Learning, Ontario, Canada.

Keywords: Web-Based Education, Evaluation of Programming Tutors, Intelligent Tutoring Systems, e-learning systems, AI in Education.

1. INTRODUCTION

Online teaching tools such as WebCT and Blackboard are becoming extremely popular for distance education and mainstream in-class education. Entire colleges and universities have implemented online teaching tools as the central mechanism for delivering all of their courses [1]. The strength of these tools is their ability to provide the teacher and student with a great deal of versatility within the learning environment. Unfortunately, they do not provide a means by which a student may receive ongoing personalized instruction. Teaching students on a one-to-one basis significantly influences the degree of knowledge and skill retained by the student; Bloom suggests that one-to-one tutoring is the most effective strategy known, generally yielding two standard deviations better performance than traditional instruction. He suggests further that mastery learning approaches one-to-one instruction in terms of measured learner gains [2].

This highlights the crisis in the educational community. In order for students to reach their potential, they need individual tutoring. However, due to a plethora of factors such as the limitations of online teaching tools, financial considerations, and sheer logistics, each student cannot be granted access to a personalized human tutor for a consistent duration of time. After all, traditionally there is only one teacher in a classroom of students. So, what can be done to solve this problem? One solution is to design and implement Intelligent Tutoring Systems (ITS). A generally accepted definition for an ITS is a system that employs artificial intelligence methods to assist trainees to improve their problem solving skills by monitoring their reasoning, tracking errors to their source, and, based on the diagnosis, providing advice and assistance to strengthen problem solving skills [3]. ITS allows for more open-ended problems [3].

This paper presents an overview of the design and development of the Java Intelligent Tutoring System, the research methods used, and the findings. These findings include student and faculty perspectives of those who used this ITS.

2. JITS DESIGN THEORY

The design of JITS primarily followed the ACT-R Cognitive Theory for Developing Tutors. The first principle derived from ACT-R (Architecture of Cognitive Tutors) is that it is essential to define the target cognitive model as a set of production rules [4, 5]. Production rules are a set of IF–THEN–ELSE constructs which outline discrete knowledge components which collectively represent the steps required for a student to reach a solution for a problem. A typical ITS may have several hundred production rules to effectively cover the domain and various states a student may be in within a realm of feasibility and predictability. Heffernan and Koedinger (2001) reinforce this principle: "Without this [principle] one does not have a well-defined educational goal" [6]. In other words, in the context of ACT-R, tutoring is assuring students (a) construct the production rules, (b) practice the production rules, and (c) remediate the errors in the production rules. Additionally, it is a goal of the Intelligent Tutoring System to guide the student towards a solution. However, it is not mandatory that the solution be achieved by the student. In other words, the ITS recognizes that the student may become frustrated and not wish to continue. The ITS records the current state of the student's progress, noting the degree of learning that has taken place even though a solution may not have been achieved.

The second principle concerns how these production rules are to be communicated to the student [4]. According to ACT-R theory, one cannot directly tell students the underlying rules [4, 7]. The goal for ITS is to provide a vehicle by which students construct knowledge for themselves as opposed to having the information told to them [8]. ITS need to communicate the production rules to students by providing them with examples that illustrate the rules. As a result, the most effective way for students to construct knowledge is to acquire these rules as a by-product of problem-solving. This form of experiential learning is an effective way for students to construct knowledge and increase their cognitive abilities [9].

The third principle of ACT-R theory is that one wants to maximize the rate at which students have opportunities to form and practice these production rules [4]. Based on other research by ITS researchers, it was shown that what predicts students' final

achievement is how much practice they have had of these rules and not how that practice occurs [5, 10]. Associated with the concept that "practice makes perfect" is the corollary to minimize floundering which is incorporated into many leading-edge Intelligent Tutoring Systems. The basic idea is to reduce student frustration during the problem-solving session and select problems that offer practice on those production rules where students most need practice [10]. A production rule in the ACT-R theory is a statement of a particular contingency that controls behaviour in the Intelligent Tutoring System. The following are two examples of production rules:

Example 1:

```
IF the goal is to classify a person
    AND he is unmarried
THEN classify him as a bachelor
```

Example 2:

```
IF the goal is to add two digits d1 and d2 in
      a column
   AND d1 + d2 = d3
THEN
   write d3 in the column
```

A production rule is a condition-action pair. The condition specifies a pattern of input symbols that must be present for the production rule to execute. The action section specifies the action that is to take place. A typical ITS may have hundreds of production rules to encapsulate the knowledge of the domain of instruction. For the design of the Java Intelligent Tutoring System, the set of production rules is represented by the grammar of the Java language coupled with custom production rules augmented to the grammar.

The fourth principle of ACT-R cognitive theory for tutoring deals with how to treat errors in student problem solving [4]. Anderson et al. base this principle on an earlier work in 1990, which states, "people learn best when they generate the answer for themselves rather than are told" [11]. However, the consequence of letting people generate their own knowledge is that errors are inevitable. Fortunately, there are four considerations outlined in ACT-R theory that deal with error remediation [4]. First, many errors do not reflect misunderstandings or lack of knowledge; rather the errors are simply unintentional slips. The second consideration is that people learn best when they construct the knowledge themselves. This is analogous to hands-on training as opposed to lecture-based teaching. The third consideration is that a lot of time can be wasted when the student is floundering while trying to solve a problem. This state is called an ***error state*** and is not beneficial for learning. The fourth consideration is that when students have problems with their knowledge, it is more effective to provide another opportunity to learn the correct production. Since the student does not need a deep appreciation of their error, it is not effective for the ITS to expound on it [12].

The ACT-R Theory for the development of tutors has led to a standard framework for the design and construction of Intelligent Tutoring Systems. The goal of this framework is to ensure that Intelligent Tutoring Systems will provide rich learning environments for students that will support their cognitive development in the specific domain of study in as effective means as possible.

Many researchers in the area of ITS support the following steps to design and construct an Intelligent Tutoring System.

1. construct the interface;
2. define the production rules;
3. create the declarative instruction; and
4. set up the Instructional Agent to manage the curriculum and engage the student through rich-interaction [4, 11, 12].

During the design of the Java Intelligent Tutoring System, these steps were performed but in a slightly different order than presented above. Due to the complexities involved with the way in which JITS is designed, a massive amount of effort was spent on step 2, that is, defining the production rules. This is because JITS was designed to recognize any small Java program and offer "intelligent" feedback when there is no authored solution available. In other words, unlike other Intelligent Tutoring Systems, there is no predetermined solution for each problem. As a result, the focus of this step in the project was on compiler error correction strategies which used extensive production rules in the form of Backus-Naur Form (BNF) for the grammar of the Java language. Once this was completed, the next step the researcher pursued was Step 3.

Once the production rules were in place and validated, the declarative instruction became the focus of the researcher. Declarative instruction was designed and implemented by a series of tutorial web pages with ease of navigation and quick reference of paramount design consideration. More information regarding this step is presented in the following sections.

In step 4 of the ACT-R theory recommendation, a prototype for the Instructional Agent including a hint generation module was designed and developed. Small curriculum modules were also created to test the interaction between user and the ITS prototype. After extensive testing of the prototype system, the last step for design and development was the construction of the User Interface (Step 1). Please see the corresponding section for more detail regarding the design and implementation of the Java Intelligent Tutoring System User Interface.

3. ERROR CORRECTION AS A DESIGN ASPECT

JITS is designed to provide extensive hands-on practice for students learning Java in the form of attempting to solve programming problems. All entry-level programming students make syntax mistakes and logic errors. Thus, a module that sophisticatedly determines the intent of the student and can identify various types of errors that students make is a necessary component for an ITS for the Java programming language.

While text correction is commonplace in word processors, mobile phones, etc., it is not commonplace in the area of compiling a computer program. When a person writes a program in any language, it must precisely follow the syntax and grammar rules of that language. Any mistake, even so minute as forgetting a ";" will cause the program to fail compilation. This research proposes an intriguing new use in teaching programming by autocorrecting typical mistakes that beginner programming students make. From a pedagogic/didactic perspective, support for the beginner programmer when these types of errors occur can be very helpful. Thus an error correction algorithm would be very helpful for students. Reviews from the learning and teaching science journals yields this to be true [9, 13]. As a result, the Java Error Correction Algorithm fits in this chosen theory. Furthermore, based on the principles of the ACT-R cognitive theory for developing tutors, the Java Error Correction Algorithm also coincides with this philosophy [11, 14].

4. JAVA ERROR CORRECTION ALGORITHM DESIGN

This section describes the design of the Java Error Correction Algorithm (JECA). The design arose from research involving decision trees, expert systems, and compiler tools [15]. It became clear after preliminary research that JavaCC provided the best features for the development of an error correction algorithm [16]. JECA is designed to consider three distinct cases:

CASE 1: student enters perfect code and it compiles and runs;

CASE 2: student enters code that needs modification but with JECA changes will compile and run; and

CASE 3: student enters code that needs modification but will not compile regardless of all corrections employed by JECA; however, suggestions are presented to the student to bring the code to a closer state for compilation.

The Java Intelligent Tutoring System's intelligence is accomplished by this embedded logic module (i.e., the Java Error Correction Algorithm). This module performs a number of operations behind the scenes. It implements a sophisticated scanner and parser that autocorrects the student's code when appropriate as well as generates a number of parse trees that have small permutations. This module then attempts to compile the best trees to ascertain the most likely path the student "intended" to follow. With this knowledge, JITS can efficiently and effectively tutor the student. The goals JECA are to:

1. analyze the student's code submission;
2. intelligently recognize the "intent" of the student;
3. "auto-correct" where appropriate (e.g., converting "While" into the keyword "while," "forr" into "for," etc.);
4. learn individual student's misconceptions, and categorize the types of errors s/he makes;
5. produce a "modified code" that will compile (or bring the code closer to a state of successful compilation); and
6. prompt the student programmer for information when necessary via well-defined hint support structures.

JECA, combined with a well-defined student modeling mechanism and dynamic hint generation capabilities, enables JITS to significantly improve the performance of beginner Java programmers. The algorithm used by JECA is presented below.

1. Create a copy of the student's submission (i.e., "modified_source").
2. The scanner examines the student's code and attempts to extract a token. Let *S* be the stream of characters to be validated as a token.
3. A validation process ensues in which comparisons are done using the reserved words and keywords of Java (Table 1), extended keywords (Table 2), and previously declared identifiers.
4. For a given identifier, if the scanner discovers, within a certain threshold, that *S* can undergo transformations to convert *S* into a valid token (i.e., a reserved word or keyword, an extended keyword, or as a previously defined identifier), then it will do so. However, if the scanner determines that *S* is sufficiently different from all of the items previously compared to, then it will be left unchanged (i.e., it will remain as a new identifier).
5. Update the modified_source code to reflect these changes and the newly constructed token is submitted to the parser.
6. Repeat 1 through 4 until all input from the student's source code has been processed and the parser has completed the construction of the parse tree representing the modified_source code.
7. Try to parse and compile the modified_source code. If the compilation succeeds, then relay the modifications performed to the student in order for them to correct their code and stop processing.
8. If the previous step fails, then extract information regarding why it failed and set up a competition of permutated parse trees containing insertions, deletions, and replacements at the problem area.
9. Run these permutated trees through the parser. The goal of this stage is to determine if the specific problem where the parse failed has been corrected.
10. Select the "best tree(s)" and compile these. The "best tree" is defined as the tree that allowed the parser to successfully consume the largest number of tokens compared to the other trees in the competition.
11. If one or more of these trees successfully compiles, then present this information to the user, indicating the changes made to the student's source code.
12. If none of the trees successfully compile then present the information to the student regarding the selection of the best tree.
13. Let the student respond/make corrections to the source code.
14. Repeat the process from 1 to 13.

The algorithm employed by JECA is presented in flowchart form in Figure 1 and Figure 2.

Table 1. Java Reserved Words and Keywords

abstract	else	interface	super
boolean	extends	long	switch
break	false [a]	native	synchronized
byte	final	new	this
case	finally	null [a]	throw
catch	float	package	throws
char	for	private	transient
class	goto [b]	protected	true [a]
const [b]	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while
double	int	strictfp [c]	

Note. [a] `true`, `false`, and `null` are reserved words. [b]indicates a keyword that is not currently used. [c]indicates a keyword that was added for Java 2

Table 2. Extended Java Reserved Words and Keywords

Boolean
Character
Number
Byte
Double
Float
Integer
Long
Short
String
StringBuffer

Note. This list is a subset of the objects defined in `java.lang.*`

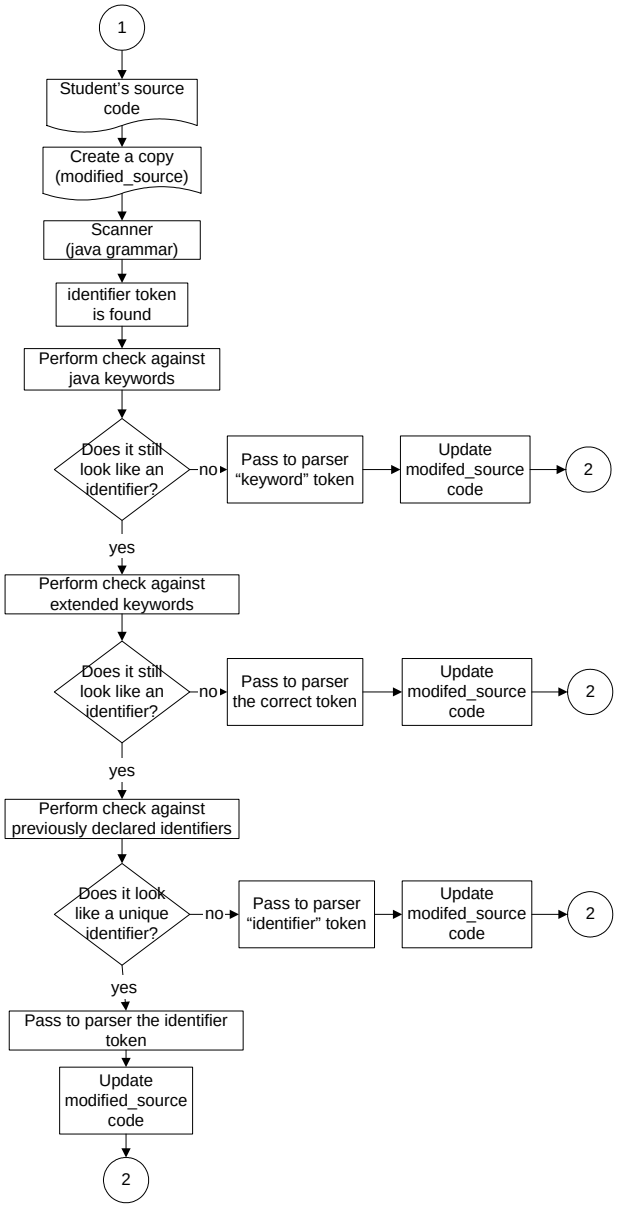

Figure 1. First Component of JECA – Scanner Correction Activities.

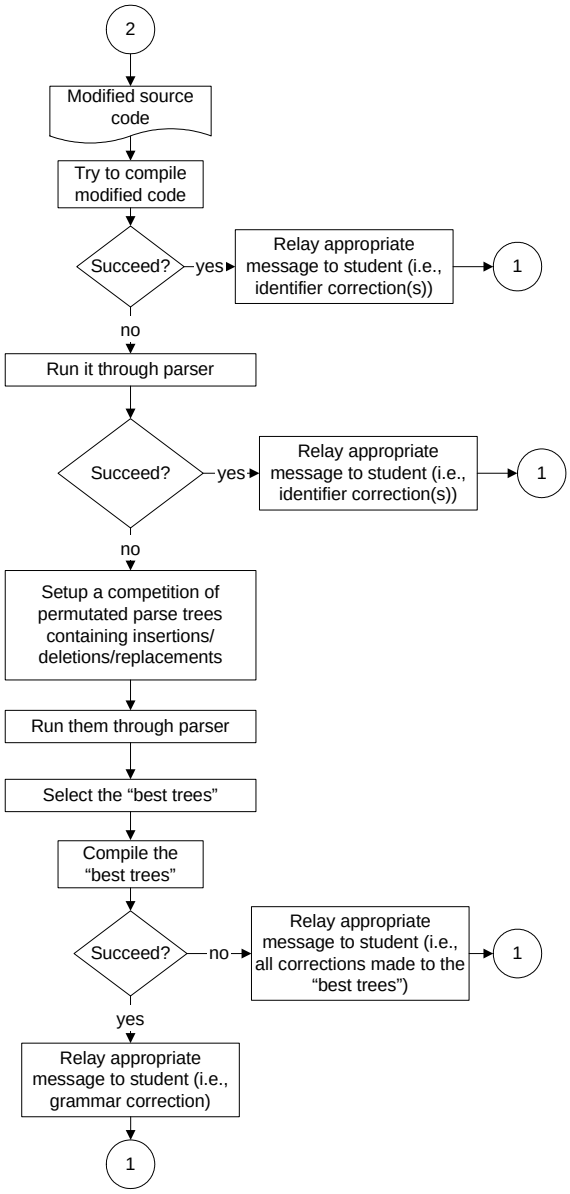

Figure 2. Second Component of JECA – Parser Correction Activities.

5. JAVA INTELLIGENT TUTORING SYSTEM DESIGN AND ARCHITECTURE

The design of the Java Intelligent Tutoring System heavily relied on JECA to provide the necessary information in order to offer suitable feedback to the student programmer. However, there were a number of factors that were considered in the design of JITS beyond what JECA offered. The two main perspectives that were considered in the design of JITS were both the student and the instructor perspectives. In order for an ITS to be successful in today's e-learning society, JITS was designed with the following qualities.

Student Perspective

The following qualities were deemed important in the design to satisfy students and were part of the desired list of criteria in the design of JITS:

1. provide an easily understood, student-friendly user interface that provides all the necessary features for effective ITS tutoring;
2. provide access via an ordinary browser;
3. will not need a high-speed internet connection (i.e., dial-up connection will work fine; thus, students in remote locations have full access to this resource);
4. process student's code submission and respond quickly to the student;
5. support many students concurrently working with the ITS;
6. engage the student by communicating in a clear and concise personalized fashion (e.g., unique hints and error messages for each student);
7. track student performance in a database (e.g., ORACLE); and
8. model the user as s/he works through a problem.

Instructor Perspective

The design of JITS also considered the instructor perspective. The following factors were important in meeting the needs of teachers using this ITS.

1. requires the author of the problem to provide minimal information (e.g., problem statement, program requirements, and required output);
2. the author of the problem does ***not*** specify any solutions (this is based on the premise that for a given programming problem there may in fact be numerous solutions);
3. JITS must be able to recognize a very large number of possible solutions for a particular programming problem;
4. student performance information should be easily accessible;
5. an instructor-friendly, web-based user interface to author problems (i.e., Authoring Tool).

This section includes the JITS User Interface, the JITS Authoring Tool, and a description of the web-based infrastructure architecture.

6. JITS USER INTERFACE

The JITS User Interface (UI) is comprised of a number of interrelated modules: the main programming IDE, the tutorial window, and the image viewer. JITS also includes support for professors to create and manage problems via the JITS authoring tool. Figure 3 depicts the current version of the JITS user interface.

The first section (i.e., label 1) presents a personalized welcome to the student logged in. Label 2 presents a note relative to the current state of solving the problem at hand. In this section, notes are dynamically created by JITS that are personalized to each student. Label 3 presents the problem template structure including the problem statement, the problem specifications, and the required output. This section also draws reference to the problem number out of the total number of problems available in this programming topic. At the end of Section 3, a link (i.e., label 4) is provided to a picture if the problem has a visual component (i.e., an equation or relevant drawing) to assist the student in more clearly understanding the problem (see Figure 4). If the student clicks the link, the picture is shown in a separate window to allow the student to refer to the picture while at the same time working with the main JITS user interface. Label 5 shows the template provided by JITS for each problem in the system. Label 6 presents the editing region where the student types his/her solution. Label 7 depicts the various buttons which the students use to interact with JITS. Buttons include "Submit" to submit a solution to a problem and to receive feedback. "View Top Hint" and "View All Hints" buttons are the means by which students can see the hints the JITS provides. The "View Solution" button provides potentially various solutions to the current problem. The "Previous Problem" and "Next Problem" buttons are used for navigating within a problem set. The "My Performance" button yields detailed information about the student's performance including problems solved, problems attempted, the number of attempts for each problem, and comparison information to the "average" JITS student.

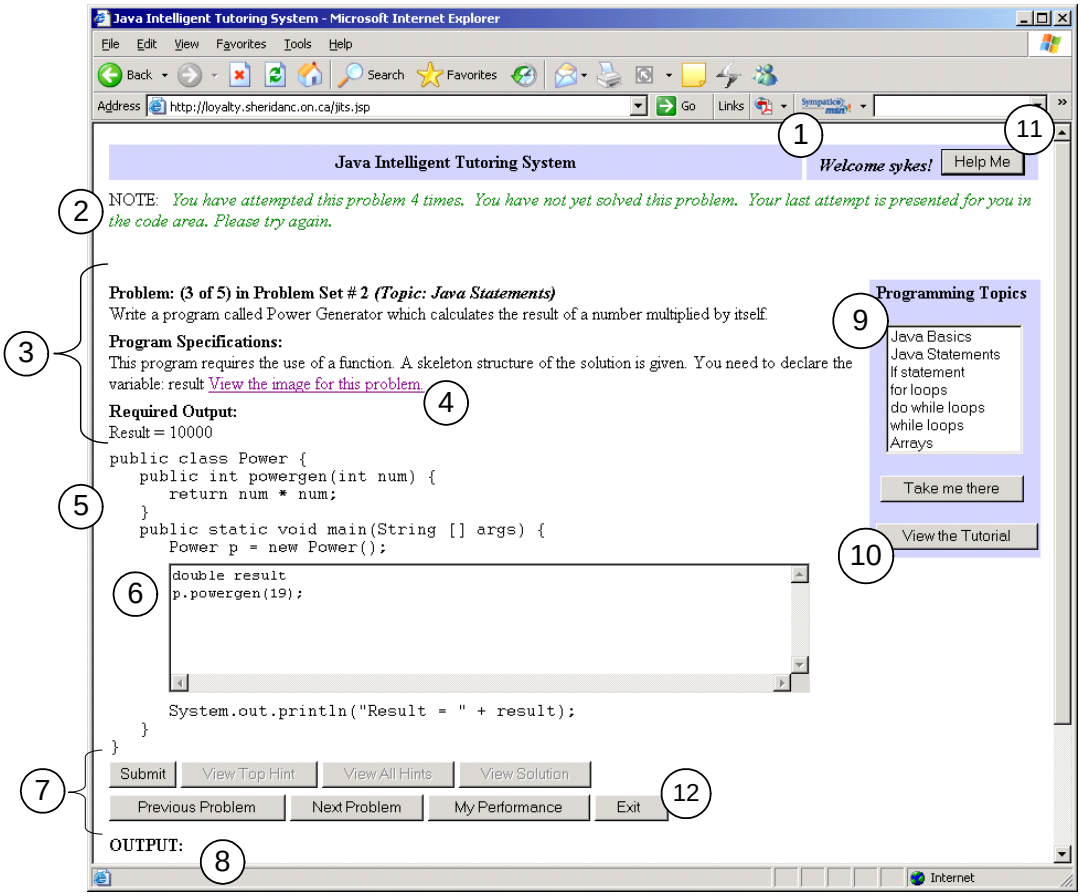

Figure 3. Main screen of the JITS User Interface.

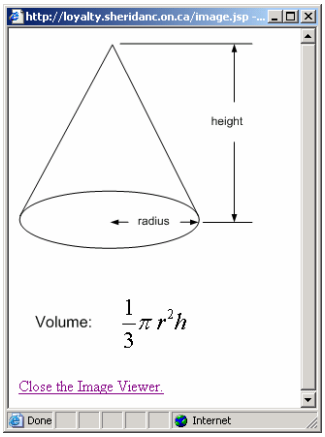

Figure 4. JITS Image Viewer depicting the image for the current problem (popup from the JITS User Interface).

The tutorial window may be viewed at the same time as the student is working with the main JITS user interface (i.e., the tutorial may be referenced while working on a problem in JITS). See Figure 5 for the Tutorial Window.

Links are provided in the "My Performance" output for rapid access to any problem the students wishes to retry (see Figure 6). Label 8 shows where the majority of the responses from JITS are presented. Information such as hints, solutions, performance scores, and errors are all shown in this area of JITS. Label 9 presents the choices of the various programming topics that the student may choose. The "Take Me There" button is used to bring the student to the selected programming topic.

Label 10 presents the "View the Tutorial" button, which launches the JITS Tutorial window.

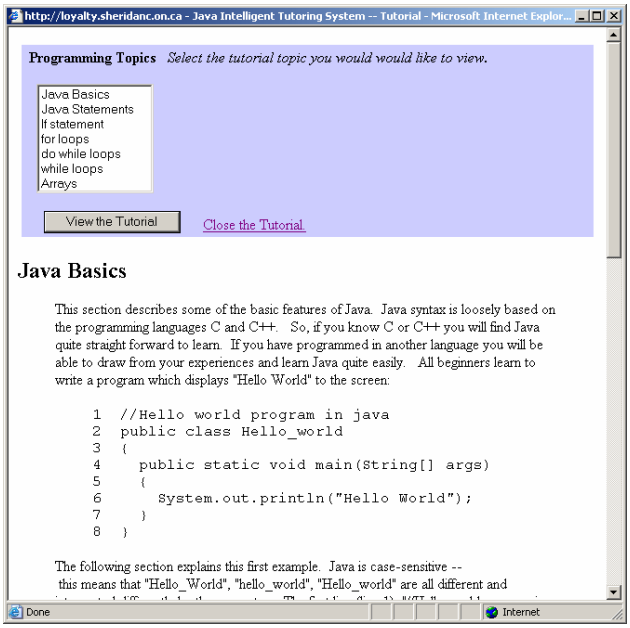

Figure 5. JITS Tutorial window displaying a sample tutorial from the list of Programming Topics.

Label 11 shows the "Help Me" button, which opens a separate window displaying the screenshot of JITS with labels to all of the components in JITS. The purpose of this window is to orient new users of JITS so that they feel supported and can more quickly become productive in this Intelligent Tutoring System. (See Figure 7 for the "Help Me" window.) Label 12 is the "Exit" button. This button brings up a screen which thanks the student for trying out the system and performs some system-wide cleanup procedures behind the scene.

Problem #	Problem Set	Solved?	Solution Viewed?	Average Student	
1	1	**No** -- 13 attempts so far.	***Yes.***	2 attempts to solve.	Review Problem: 1 of set: 1
2	3	**No** -- 1 attempt so far.	**No.**	2 attempts to solve.	Review Problem: 2 of set: 3
1	4	***Yes !*** It took 5 attempts.	**No.**	2 attempts to solve.	Review Problem: 1 of set: 4
1	6	**No** -- 3 attempts so far.	**No.**	2 attempts to solve.	Review Problem: 1 of set: 6
3	6	**No** -- 1 attempt so far.	**No.**	1 attempt to solve.	Review Problem: 3 of set: 6
1	7	***Yes !*** It took 2 attempts.	**No.**	1 attempt to solve.	Review Problem: 1 of set: 7
2	7	**No** -- 1 attempt so far.	**No.**	1 attempt to solve.	Review Problem: 2 of set: 7

Figure 6. "My Performance button" output showing performance and links to previously attempted problems. Different font styles, emphasis and the use of colour distinguishes solved problems from unsolved problems.

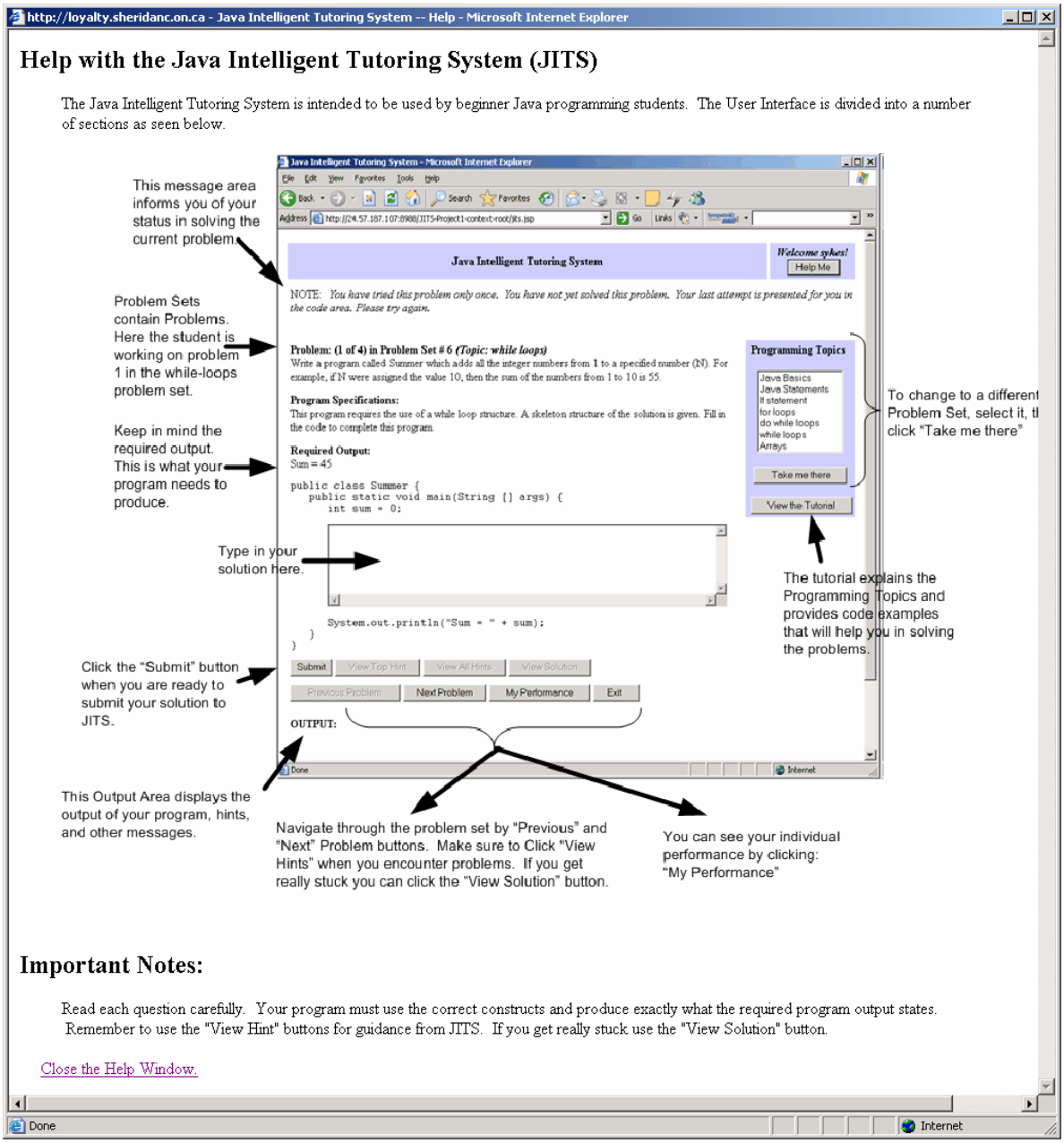

Figure 7. JITS' Help screen is used to assist new users to become oriented with this ITS.

7. JITS AUTHORING TOOL

An authoring tool is currently being developed which provides the teacher with a convenient means to add problems to the database for JITS to use. This is a very easy process because the teacher only needs to provide the following information:

a) the problem statement;
b) the problem description;
c) the required output; and
d) the skeleton structure of the program.

As a result, the JITS authoring tool is intended to be extremely user-friendly and easy to add many problems of various levels of difficulty.

Once the teacher has submitted the problems they are immediately available to JITS and thus students of the system.

The Authoring Tool provides a means to view the all the problems in the lesson set and edit selected problems [17]. In the Java Intelligent Tutoring System, the author of problems does not provide a solution.

JITS carefully scrutinizes the student's submission based on the problem description, specification, required output and template code is used by JITS uses to determine the appropriate feedback to the student. This ensures the greatest degree independent knowledge creation for each student [6, 18].

Once the professor is properly authenticated to the system via login screen, the JITS Authoring Tool User Interface is presented as shown in Figure 8.

8. INFRASTRUCTURE DESIGN

The infrastructure design for JITS draws from the area of leading-edge techniques and technologies for multithreaded distributed concurrent e-learning application designs. The Model-View-Controller (MVC) design pattern was used to ensure that concurrency and robustness would be provided by JITS. The MVC contains three main tiers: the client's browser, the middle-tier, and the database-tier.

By design, there were no restrictions placed on the browser. In other words, JITS was designed to work with any browser, and no custom installed client software of any sort was required. The middle-tier is a server running a TomCat web server, currently equipped with 4GB RAM and 2 Pentium-IV processors. The database-tier is a separate server running ORACLE. The initial JITS database schema was designed to support the core functionality of JITS consisting of 3 tables: student, problems, and student_problems. The student table contains information regarding each student in the system such as student name, password, current problem, etc.

The problems table contains details regarding programming problems used by JITS such as problem description, specifications, templates, etc. The student_problems table is an intersection relation representing details regarding each student's attempt at a problem.

The Model-View-Controller design pattern was a core component to the design of JITS. Figure 9 depicts the MVC design pattern. First the student makes a request (via HTTP in the browser). The Controller module receives the request and performs operations that include instantiating JavaBeans. These beans are used to model the student as s/he works with JITS. The collection of these beans represent the model of each student in JITS. During specific operations, beans may need to retrieve information from the JITS database schema (e.g., to select a new problem or retrieve solutions to a problem, etc.). These data are stored in the ORACLE JITS database schema represented in the figure as the Enterprise Information System (EIS). The information is gathered up and processed by the bean, which then forwards it to the View component (i.e., the Java ServerPage [JSP]), which then formats it appropriately for the student in the JITS user interface and returns it to the student's browser.

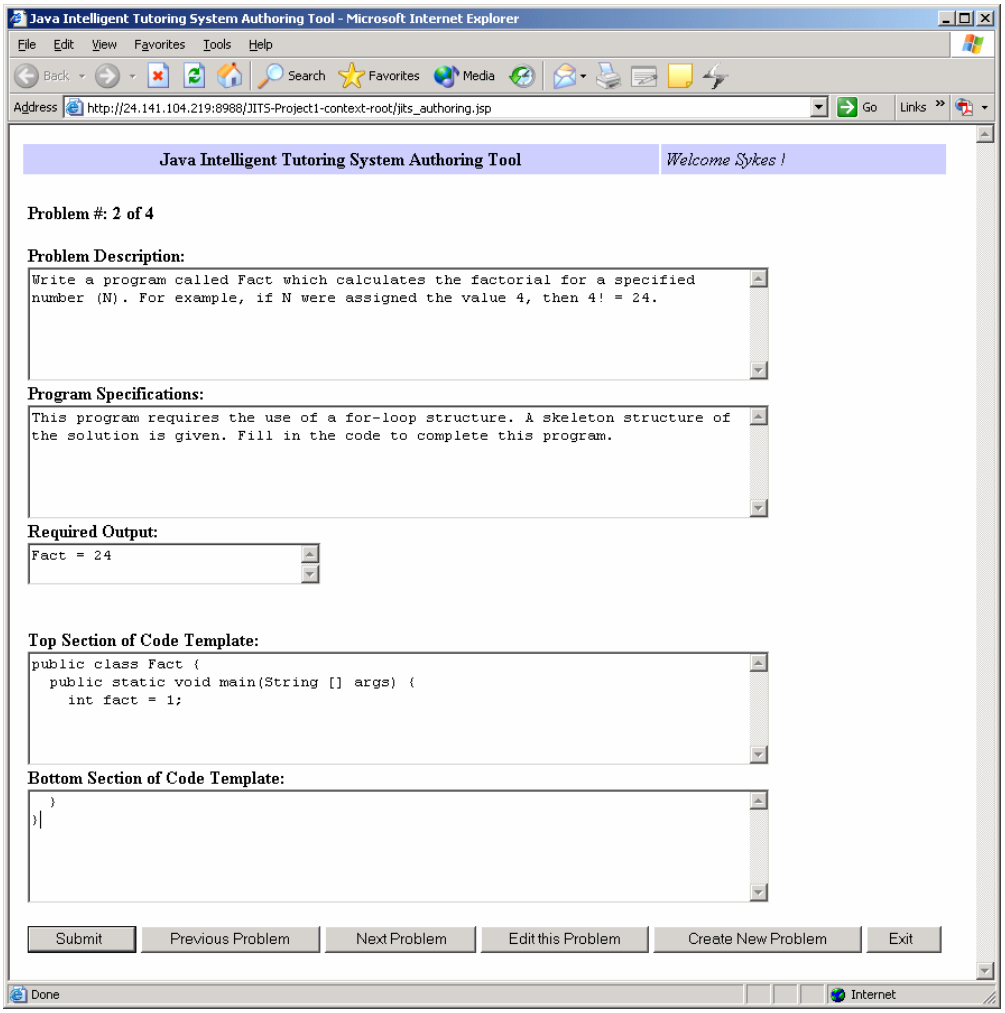

Figure 8. JITS Authoring Tool User Interface.

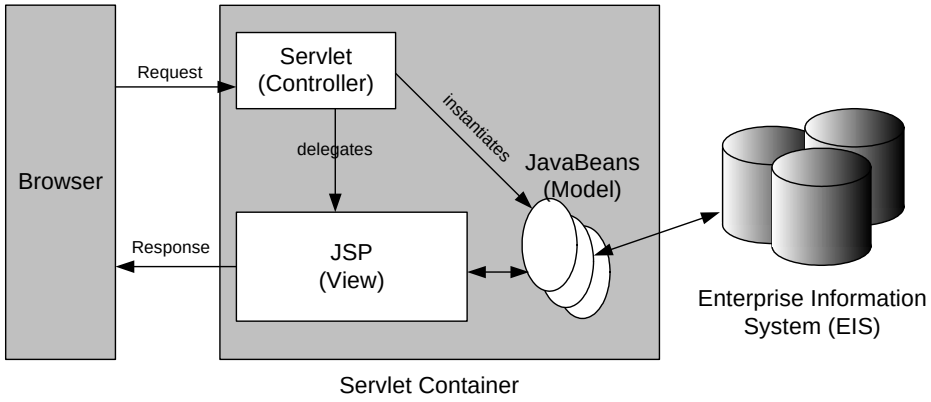

Figure 9. Model-View-Controller (MVC) design pattern implemented in JITS.

Hint Generation

An additional design consideration is the categories of hints that are generated by JECA for JITS. There are a number of different categories of hints that may be created as a result of the student's code submission. They are presented in Figure 10.

A `KEYWORD_REPLACEMENT_HINT` arises from a situation where the student typed in a suitably close representation to a Java keyword. For instance, if the student typed in "`Cases`," this would be interpreted as the keyword "`case`." An `EXTENDED_TYPE_REPLACEMENT_HINT` is when the student wrote "interger" which will interpreted as "Integer"–the `java.lang.Integer` data type. An `IDENTIFIER_REPLACEMENT_HINT` is used in the situation where a suitably close match to an existing identifier has been found.

```
KEYWORD_REPLACEMENT_HINT = 1;
EXTENDED_TYPE_REPLACEMENT_HINT = 2;
IDENTIFIER_REPLACEMENT_HINT = 3;
GRAMMATICAL_HINT = 4;
CLOSE_BUT_LOGIC_ERROR = 5;
SUCCESSFULLY_SOLVED_PROBLEM = 6;
GENERAL_HINT = 7;
OTHER_TYPE_OF_HINT = 8;
```

Figure 10. Hint categories.

For example, consider the following snippet of code:

```
float my_float = 3.1415;  // declaration
my_flot = my_floatt * 2;  // and use
```

There would be two `IDENTIFIER_REPLACEMENT_HINT`s generated for this section of code:

Identifier Replacement Hint:
Would you like me to replace "my_flot" with "my_float"?

Identifier Replacement Hint:
Would you like me to replace "my_floatt" with "my_float"?

A `GRAMMATICAL_HINT` is generated when the parser fails on a specific production in the modified Java grammar. Specific information regarding the error is recorded in a Hint object (e.g., `_offending_token`, and `corrected_line_of_code`). The last two types of hints are `GENERAL_HINT` and `OTHER_TYPE_OF_HINT`. `GENERAL_HINT` is used when the student is far from the solution path and needs to be realigned with the program statement and program specifications for the posed problem. If the student's code compiles but produces output that is not the same as the required output, as specified in the problem statement, the `CLOSE_BUT_LOGIC_ERROR` is used. The term "close" in this expression is intended to convey that the student is on the right track in terms of using the correct constructs, and code compiles and generates output that is reasonably "close" to the required output for this specific problem. When this type of error occurs, JITS, via an AI_Module, investigates what the logic error is and generates an appropriate hint. When the student solves the problem the `SUCCESSFULLY_SOLVED_PROBLEM` hint is used. Last, `OTHER_TYPE_OF_HINT` is reserved for future research.

There are several pieces of important information represented in a Hint object. (See Figure 11 for an illustration of a Hint object.) The `_type` member corresponds with one of the six types of categories of Hints currently supported in JECA. The `_col` and `_line` members specify where the error occurred. The `_line_of_code` and `_error_pointer` represent the source code and the exact location of where the error occurred. There are two tokens to assist in identifying where the error occurred in terms of the tokens. `_offending_token` represents the precise token the parser failed on, and `_previous_to_offending_token` represents the last successfully parsed token during parsing. The `_hint` member is a String summarizing the actual hint relying on the values of other data members in this object. It is intended to be used during the feedback process during student tutoring. The last member of the Hint class is the `_confidence`, which will be assigned an integer from 1 to 10. A confidence value of 1 indicates a high level of certainty, indicating the suggested hint is correct and will bring the student closer to a compiled program. On the other hand, a confidence value of 10 indicates uncertainty on behalf of the hint generated. In these situations, the student will have to use their own judgment based on the detailed information provided to them by the Hint objects, namely the data members, `_type`, `_col`, `_line`, `_line_of_code`, `_error_pointer`, `_offending_Token`, and `_previous_to_offending_Token`.

An example follows to illustrate these design aspects of the proposed error correction algorithm. Given the source program depicted in Figure 12, JECA would modify the program and generate the following three Hint objects.

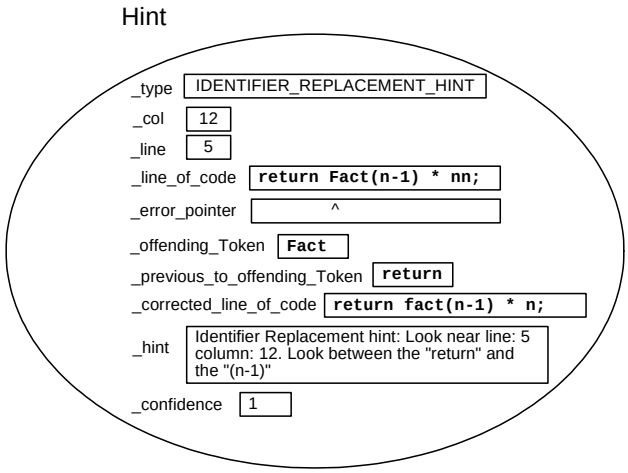

Figure 11. A JECA Hint object encapsulating identifier replacement error and remediation information.

```
public class Factorial {
  public static void main(String args[] ) {
     System.out.println("22! is " + fact (22) );
  }

  public static long fact (long n) {
     if (n=1)
        return 1;
     else
        return Fact(n-1) * nn;
    }
```

Figure 12. Factorial program with grammatical errors and syntax errors (emphasized as boldface).

JITS then takes these hints, and, in accordance with the AI_module, offers the most suitable feedback to the student. The Hint objects are displayed below for the Factorial problem (Figure 11) is presented below.

1. **Grammatical hint: In the line: "`if (n=1)...`" Look between the "n" and the "1". Suggestion: "`if (n==1) ...`"**
2. **Identifier replacement hint: In the line "`return Fact(n-1) ...`" Suggestion: replace "`Fact`" by "`fact`"**
3. **Identifier replacement hint: In the line "`return Fact(n-1) ...`" Suggestion: replace "nn" by "n"**

9. METHODOLOGY AND PROCEDURES

The methodology employed in this research is supported by two distinct research components. The first component is related to the manner in which JITS was designed and constructed. In this research section, students and professors using the prototype JITS offered suggestions and comments for the improvement of JITS. The new knowledge was fed back into the redesign and construction of JITS. Beyond the initial development of JITS, a cyclic process was used: design, develop, test, modify, redesign, redevelop, retest, etc. This research methodology involved qualitative instrumentation including observation, surveys, and personal interviews. The goal of this methodology was to improve JITS.

The second component of the methodology is related to the manner in which JITS was evaluated from a quantitative perspective. The research methodology for this section involved an experimental design with repeated measures. The results from this section of the research is not presented in this paper. However, the qualitative evaluation of JITS is presented from two distinct perspectives: students and professors.

Subjects

The population of this study was students across the province taking a comparable course in programming. The sample in this study was the students in their first year of college taking a beginner Java programming course at the Sheridan Institute of Technology and Advanced Learning. During the summer of June to August 2004, there were two such classes taking this course. One class was located at the Davis campus. This class was the experimental group (i.e., JITSC). The other class was located at the Trafalgar Road campus. This class was the control group, which consisted of 23 students. One professor taught both classes for the first 7 weeks. After a midterm break for week 8 in the term, another professor took over and taught both classes for the remainder of the term (i.e., for the last 7 weeks). Fourteen students consented to try the Java Intelligent Tutoring System (i.e., JITSC). Approximately every week, ½ to 1 hour long sessions were conducted by the researcher to elicit specific information about their experience with the Java Intelligent Tutoring System.

A similar study was conducted during the fall of September to December 2004. During this period there were two instructors teaching a first year Java programming course. Instructor "A" had two classes; the JITSC group consisted of fourteen students, and the C group consisted of 25 students. Instructor "B" had three classes; the JITSC group consisted of fourteen students, the C1 group consisted of eighteen students, and the C2 group consisted of 23 students. Both instructors taught for the entire semester (i.e., 14 consecutive weeks). Every week, ½ to 1 hour long sessions were conducted by the researcher to elicit specific information about their experience with the JITS.

During both time periods (Summer and Fall 2004) the JITSC group were talked to and observed during the ½ to 1 hour long sessions. Additionally, many JITSC students emailed the researcher with comments and suggestions for improvement. The manner in which students were interviewed was primarily individually based; however, there were some occasions when an issue was raised that were a shared concern among several students. The total number of students involved in this entire research project (i.e., all JITSC students) was 14*3 = 42. The kind of note taking procedures were observations recorded in a researcher's log book. Such observations included information regarding individual student's progress through a specific programming problem in JITS. For example, the programming topic, the problem number, types of mistakes and errors, and JITS' response to the student were all recorded in the researcher's log book.

Professors were also selected to participate in this study. The selection of professors was based on a number of factors including their knowledge of the Java programming language, level of course offerings, and interest in offering critical opinions on the Java Intelligent Tutoring System. A total of 4 professors were selected for this study.

Statement of Procedures

An interview-style survey sheet was constructed to aid in gathering input from students in the JITCS groups and professors. The survey included six open-ended questions to facilitate a great number of perspectives and opinions. One of the measurement instruments for this component of the study was this survey, depicted in Table 3. By presenting the survey to students and teachers who have used JITS, feedback representative of these two perspectives was gathered. Additionally, the researcher often visited the classroom to informally assess JITS. Between ½ hour and 1 hour per week was spent with students and professors, who offered important suggestions for improving JITS. This information was recorded in the researcher's logbook. This form of data gathering proved to be the most effective way of receiving feedback from students and instructors for the refinement and improvement of JITS.

10. FINDINGS

Overall, the students found the Java Intelligent Tutoring System enjoyable, beneficial, and useful. Table 4 depicts the summary statistics of the qualitative survey from the student's perspective. It can be seen that JITS performed above average in all categories and scored the highest in two categories: "Usefulness", and "Ease of Understanding Tutoring Style".

Table 4. JITS Qualitative Summary Results for Students

JITS Qualitative Summary Results – Students	
1. Usefulness	71%
2. Beneficial	64%
3. JITS is better than a traditional classroom	36%
4. Ease of JITS Tutoring Style	79%
5. Enjoyable	79%
6. Learn Better	71%

The Student's Perspective

All of the students enjoyed working with JITS. Many voiced they are pleased with the following:

1) Feedback mechanism – It provides hints quickly and to the point. The hints are also not overwhelmingly complicated – quite unlike traditional compilers.

Table 3. Qualitative interview sheet

Qualitative Project Interview

I am conducting a survey of those participants who were taught using the Java Intelligent Tutoring System at Sheridan. The information gathered from our interview will be used for my research. This involves determining the effectiveness of learning in this environment. For each question select the most appropriate response based on the following scale:

1 = strongly favorable to the concept, 2 = somewhat favorable to the concept, 3 = undecided, 4 = somewhat unfavorable to the concept, 5= strongly unfavorable to the concept. The following questions will be asked during the interview.

1. How do you rate the Java Intelligent Tutoring Systems usefulness?

Very Useful **Not Useful**

1 2 3 4 5

Comments: _____

2. Do you feel the Java Intelligent Tutoring System is beneficial to your studies? List and explain the advantages/disadvantages of this learning environment.

Very Beneficial **No Benefits**

1 2 3 4 5

Comments: _____

3. Compare JITS with a traditional classroom. Do you feel JITS is better or worse than an ordinary classroom teaching environment? Identify any similarities and differences between a traditional classroom experience and the JITS learning experience.

JITS is much better than traditional classroom **JITS is much worse than traditional classroom**

1 2 3 4 5

Comments: _____

4. How do you rate the ease with which you use and understand the tutoring style of the JITS?

Very easy to use & understand **Very difficult to use & understand**

1 2 3 4 5

Comments: _____

5. Have you enjoyed JITS? Explain why or why not.

Very Enjoyable **Not enjoyable**

1 2 3 4 5

Comments: _____

6. Do you feel you learn more detailed information or about the same as a regular classroom when using JITS? Explain why or why not.

Learn Better **Learn the same**

1 2 3 4 5

Comments: _____

2) One student stated, "[JITS] tells me the exact spot in the code where I need make my correction – I like that. I wish other systems would do that."
3) JITS helps students solve syntax and logic errors while developing a solution to a problem. One student stated, "I am definitely learning better in this environment than in a traditional environment."
4) Integrated Development Environment – similar to professional programming environments.
5) Many students stated that they felt JITS was very useful since it is available 24/7 and all a student needs is a browser.
6) One student said, "Can we have this system in our course from now on?"

The Professor's Perspective

The section summarizes the views of Professors involved in this study. Table 5 shows the statistical results of the interviews.

Table 5. JITS Qualitative Summary Results: Professors

JITS Qualitative Summary Results – Professors	
1. Usefulness	100%
2. Beneficial	75%
3. JITS is better than a traditional classroom	25%
4. Ease of JITS Tutoring Style	75%
5. Enjoyable	75%
6. Learn Better	75%

Many Professors said they are pleased with JITS in the following ways:

1) One Professor stated, "The embedded logic unit called JECA is a sound tool – it picks out the most significant error the student need to focus on. I feel the student is developing core programming debugging skills with JITS."
2) Integrated Development Environment – similar to professional programming environments.
3) Many Professors said that they would like to use JITS to augment their existing Java courses. They felt that JITS provides a means for students to receive extra tutoring when the Professor is not available.
4) One Professor said, "The quality of tutoring that JITS performs is comparable to a human tutor."
5) All of the Professors said that they liked the fact that there was no client installation required for them or their students.
6) Many Professors were happy that JITS was available 24/7. This makes it easier for students to work on problems at their own time and at their own pace.

One Professor suggested that JITS could produce a report representing the student's performance over a period of time. This would also be helpful to identify students who need additional assistance. It could also be used to identify those students who are doing extremely well and may be interested in more challenging problems.

All of the Professors enjoyed using the JITS Authoring tool. Although still under development, the prototype made Professors aware that they can easily create, edit, and review problems. Once the problems have been added they are immediately available to their students. A second benefit Professors stated was the fact that they needed only a browser to access the Authoring tool and JITS. Custom client installations are not required to use the Java Intelligent Tutoring System and the Authoring Tool. The majority of Professors in this study felt that this 24/7 access from any Internet connection was a very good feature.

11. CONCLUSIONS

The Java Intelligent Tutoring System prototype has been a success. JITS was met with interest by students and Professors alike. After trying numerous problem sets with several groups of students and Professors, they were happy with the performance of JITS. The various issues and suggestions raised by students and Professors are being reviewed. For instance, the researcher is currently investigating video-streaming as an instructional aid and enhanced logic support for students while working on solving a problem. Integration of some of these requested features will be available in the future releases of the Java Intelligent Tutoring System available soon.

12. REFERENCES

[1] M. Boyd, "Center for instructional technologies," vol. 2003, 2003.
[2] S. B. Bloom, "The 2-sigma problem: The search for methods of group instruction as effective as one-to one- tutoring," *Educational Researcher*, vol. 13, pp. 4-16, 1984.
[3] W. Tracey, R., *The Human Resources Glossary: The Complete Desk Reference for HR Executives, Managers and Practitioners*, Third ed: CRC Press, 2003.
[4] J. R. Anderson, "Production Systems and the ACT-R Theory," in *Mind readings: Introductory selections on cognitive science*, P. Thagard, Ed. Cambridge, MA: MIT Press, 1998, pp. 59-76.
[5] J. R. Anderson and R. Pelletier, "A Development System for Model-Tracing Tutors," presented at The International Conference on the Learning Sciences, Northwester University, Evanson, Illinois, USA, 1991.
[6] K. R. Koedinger, "Cognitive tutors," in *Smart machines in education*, K. D. Forbus and P. J. Feltovich, Eds. Cambridge, MA: MIT Press, 2001, pp. 145-167.
[7] A. C. Graesser, N. K. Person, and D. Harter, "Teaching tactics and dialog in autotutor," *International Journal of Artificial Intelligence in Education*, vol. 12, pp. 12-23, 2001.
[8] Woolf, J. Beck, C. Eliot, and M. Stern, "Growth and maturity of intelligent tutoring systems: A status report," in *Smart machines in education*, K. D. Forbus and P. J. Feltovich, Eds. Cambridge, MA: MIT Press, 2001, pp. 100-144.
[9] R. C. O'Reilly and Y. Munakata, *Computational Explorations in Cognitive Neuroscience*. London, England: MIT Press, 2000.
[10] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier, "Cognitive Tutors: Lessons learned," *The Journal of the Learning Sciences*, vol. 4, pp. 167-207, 1995.
[11] J. R. Anderson, C. F. Boyle, A. T. Corbett, and M. W. Lewis, "Cognitive Modelling and Intelligent Tutoring," *Artificial Intelligence*, vol. 42, pp. 7-49, 1990.
[12] N. T. Heffernan and K. R. Koedinger, "The Design and Formative Analysis of a Dialog-Based Tutor.," presented at AI in Education 2000 Workshop on Building Dialogue Systems, 2001.
[13] M. Hristova, A. Misra, M. Rutter, and R. Mercuri, "Identifying and correcting Java programming errors for introductory computer science students," presented at 34th SIGCSE technical symposium on Computer Science Education, Reno, Navada, USA, 2003.
[14] E. R. Sykes and F. Franek, "Presenting JECA: A Java Error Correcting Algorithm for the Java Intelligent Tutoring System," presented at IASTED International Conference on Advances in Computer Science and Technology, St. Thomas, Virgin Islands, USA, 2004.
[15] E. R. Sykes and F. Franek, "A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java," *International Journal of Computers and Applications*, vol. 1, pp. 35-44, 2004.
[16] V. Sreenivasa, "JavaCC User Manual," 2006.
[17] C. N. Rowe and P. T. Galvin, "An authoring system for intelligent procedural-skill tutors.," *IEEE: Intelligent Systems*, vol. 14, pp. 61-69, 1998.
[18] A. C. Graesser and N. K. Person, "Question asking during tutoring," *American Educational Research Journal*, vol. 31, pp. 103-137, 1994.