The neglected battle fields of Syntax Errors

Sarah K Kummerfeld and Judy Kay

School of Information Technologies,

University of Sydney, 2006 Australia

{sarah, judy}@cs.usyd.edu.au

Abstract

Syntax error correction is an essential part of the debugging process. Yet there has been little research investigating how programmers approach syntax error correction and how to help beginner programmers learn to fix errors efficiently. This paper describes development and evaluation of a tool to support students learning how to correct syntax errors.

We collected both quantitative and qualitative data for a small but varied group of students as they corrected syntax errors. This showed that even the more experienced students took significant time to correct some syntax errors. It also indicated that general and language specific programming experience provides both strategic skill in correcting errors and greater depth of understanding of the error messages themselves. At the same time, we observed that beginners can be almost as efficient as more expert users when they have access to our tool for explanations of the less intuitive compile error messages.

1 Introduction

We have observed students struggling to correct syntax errors. They find the process sufficiently difficult that they focus debugging effort solely on syntax error correction, referring to their program as having one bug left, when they actually mean one compile error message. We also observed students taking little care to read the compile error messages, perhaps because they find them incomprehensible.

Debugging pedagogy has been an active area of research [3,4,13]. There has been much work done to develop methods and tools to help students perform thorough testing [7]. However, this work has been focused almost exclusively on runtime and logical errors rather than compile-time errors [11,14].

There is relatively little written about syntax error correction. The few studies that have been conducted focused on enumeration of different classes of syntax errors [2 and references therein]. Certainly, experienced programmers make relatively few syntax errors and can typically correct them far more easily than runtime or logical errors. However, syntax errors are significant for beginners.

Our own experience and observations of students indicates that students using an unfamiliar or new programming language waste considerable time correcting syntax errors. Studies have shown that excessive time spent on correcting syntax problems can be detrimental to long-term success as students become disheartened with programming [7,9]. So this battle to combat syntax errors is a significant and neglected aspect of teaching programming.

Reducing the cognitive load, by easing the burden of syntax error correction, will also enable students to learn more quickly and efficiently. This phenomenon is particularly noticeable in students learning a language like C for the first time. Competence in C requires an understanding of a broad range of concepts for even very simple programming tasks. The students in this study had first learnt Blue [5] (an Eiffel-like language) using a development environment with very good debugging facilities.

2 The development of a tool

We chose a simple approach to supporting students learning to deal with syntax errors. It is a web-based reference guide, which catalogues some common C/C++ compile errors. The errors were compiled from personal coding experience and also were provided by beginner C programmers as they wrote their first C programs.

Each error message is explained with examples highlighting the problem and at least one possible correction. Figure 1 shows a sample entry from the reference guide. The actual text (i.e. *using ANSII, gcc compiler*) of the error message is shown at the top of the entry: the user can search for this, using the text of the message from the compiler. An explanation is in the top left box of the table. The lower row shows a code example on the left. In the actual screen, this appears in red. Its correction is to the right in green (see [6] for a complete listing of the 16 errors). We emphasise that this type of tool

represents a very simple approach that could readily be created by any teacher. It would also be desirable to elicit additional examples from the student body.

'

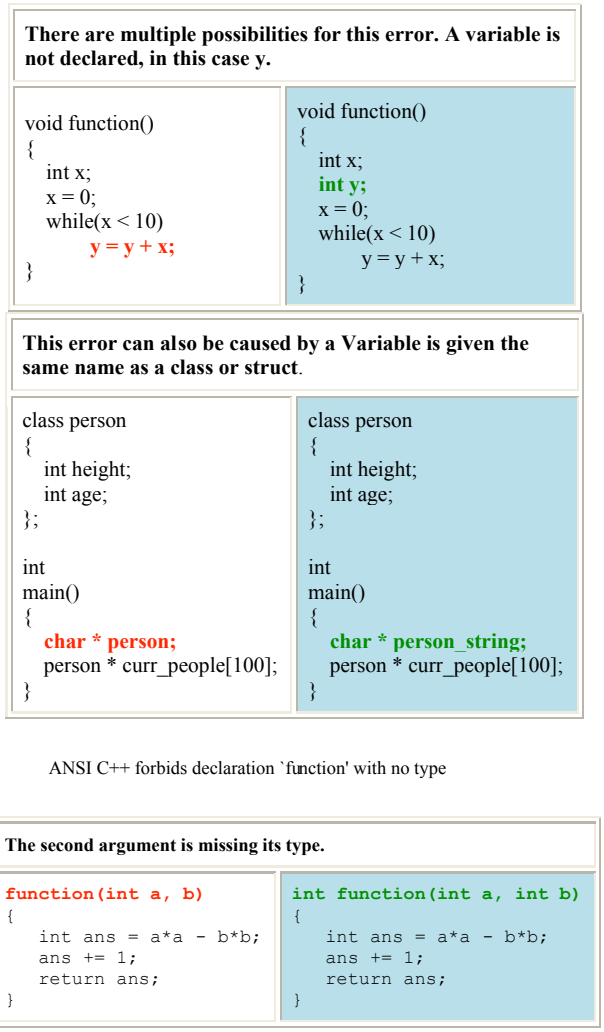

Figure 1: Sample entries from the reference guide. The first entry shown is an example where a single error message may have indicate one of a range of problems, show here are: missing variable declaration and the declaration of a class and variable with the same name. The second entry explains the error caused by failure to include the type of a variable in an argument list.

3 Evaluation of the tool and study of syntax error debugging approaches

We wanted to evaluate the effectiveness of this tool for both novice and more experienced programming students who needed to perform syntax error correction. We conducted a small scale study, qualitative study using a similar approach to that taken for runtime debugging (e.g.[2]). We chose to go beyond pure evaluation of the tool, opting for a broader exploration so that we might gain a deeper understanding of the way that it might fit into student's strategies and knowledge about syntax error correction. We explored three main broader issues which the literature suggests are relevant to understanding student strategies for debugging: how programming students, both novice and experienced approach syntax error correction; which were effective approaches to syntax error correction and; how syntax error correction might be taught or supported.

We selected a group of students who had varying degrees of programming experience, academic proficiency and familiarity with C/C++. At this point, it is critical to note that there is a large range of performance amongst professional programmers [8]: amongst programmers with similar experience, studies report differences by a factor of 10 to 20 in programming skill, where this includes time to complete a task as well as the quality of code produced and number of errors during programming and the final product. This means that it is difficult to compare efficiency in debugging between users. Our study includes a range of user experience and academic performance so that we could observe the ways that a range of different learners approach syntax error correction, both with and without the guide. Subjects were arbitrarily allocated to one of two groups: those with access to the guide and those without.

Figures 2 and 3 show the subjects' variation in programming experience (both generally and in C) and their academic proficiency based on results in computer programming subjects.

A set of 10 small programs, each with one or two syntax errors was developed. These were designed to include a cross-section of different problems in no particular order. They were chosen because the compile error messages were difficult for the author/contributor to correct and so, were judged as unintuitive or likely to cause difficulties in debugging. The experienced C/C++ programmer and teacher will recognize them as problems. For example, the error shown in Figure 1 indicates that a function is missing its 'type'. In fact, the function's second argument is missing its type. Making the link between the error and the real problem requires either specialized knowledge (having encountered it before) or a strategy for tracking down the problem using the error message as a starting hint. Another important case is where one error can be provoked by a number of different problems. Figure 4 provides a summary of all the errors.

Monitoring software was developed to record the time of compilation, the error that occurred and the program code at the time. This was written in *Python* and designed to

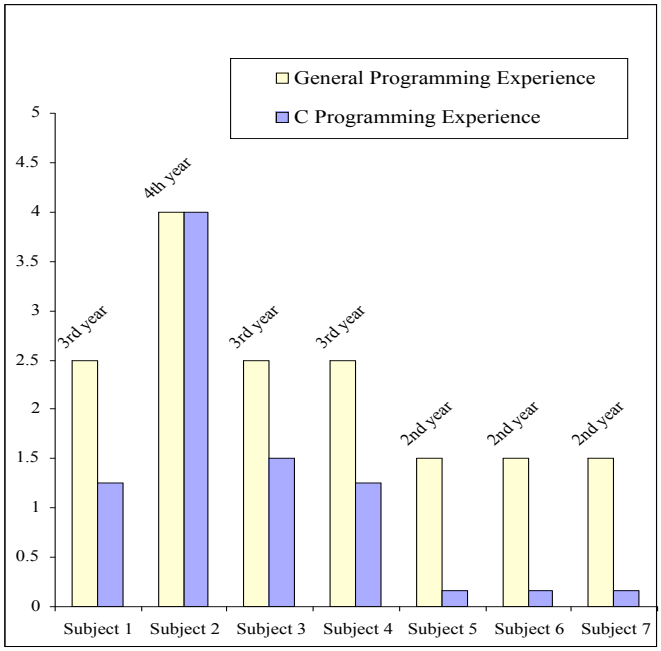

Figure 2: Summary of subject experience. *C programming experience was estimated by adding the number of years the subject had been programming in C and the number of major projects they had coded in C. Subjects 5, 6 and 7 were novice C programmers; they began learning C 8 weeks before the study. Subjects 1, 3 and 4 were more experienced and 2 was more experienced again. General programming experience included years of programming and languages learned. The year level of each subject is included above the bars.*

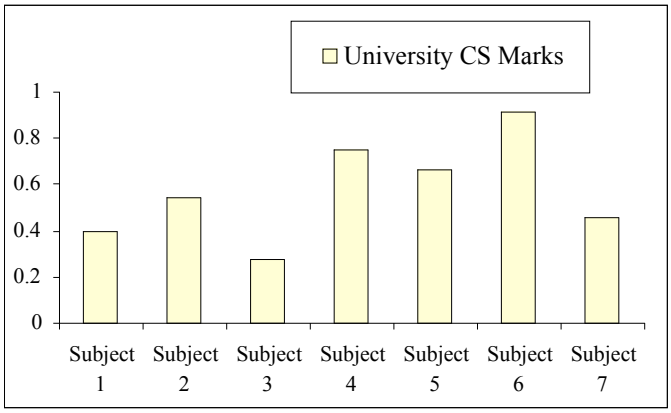

Figure 3: Summary of subject university *Computer Science marks. The marks and level of course (standard or advanced) for Computer Science programming courses was used as an estimate of academic proficiency. Subject 3 received mainly Pass grades (50-64%) while subject 6 received mainly High Distinctions (85-100%), placing them in top few percent of those passing.*

look like a standard shell, with a restricted set of commands. A separate window was made available for subjects to access other commands if they wished. The purpose of the restricted shell was merely to provide us with additional data: we did not want to limit the students or force them to alter the way they preferred to work. The students edited their coding using their choice of text editor. The code was compiled using gcc.

On completing the set tasks, the students were asked to complete a short questionnaire. This asked about their C and general programming experience and their university marks. Those who were given access to the reference were asked how useful they thought it was and whether they would be likely to use such a guide in the future.

Program	Error
A	- Semicolon is missing after class *video_obj* - *vector<*video_obj>* catalogue; * in wrong place
B	- missing *#include<stdio.h>* - missing /* on comment
C	- unused variable *'char * line'* - return statement for function *doit* is within an if statement (it may not be necessary to return anything even though it should).
D	- The declaration of function, *stratify*, is missing a return type.
E	- Semicolon missing after class *arrive* - The variable called *type* is declared twice, once as an *int* later as a *string*.
F	- Declaration of *main* function is missing the *char ** type in its argument list. - A variable called *the_char* not declared
G	- *Node* is used as both the name of a struct and the name of a variable.
H	- The function called *printReverse* is prototyped with the wrong arguments - There is a conditional statement with a single = instead of ==.

Figure 4: Summary of syntax errors. Each program (A-H) contained one or two errors. This table summarizes the problems with each program.

4 Results and Discussion

Quantitative data was collected by analyzing the time and number of compilations necessary to solve each problem. These metrics were used as a measure of efficiency. The results are summarised in Figure 3. Subjects 1-6 solved all tasks in 1.7-4.5 minutes on average and in an average of 2.7-3.8 compilations. Note that the experienced programmers (1-4) performed quite similarly to 5 and 6. Subject 7 failed to correct 7 of the 8 errors despite spending over an hour on the exercise.

Qualitative data was collected using think-aloud protocols [10,12]. This made it possible to determine how the students approached syntax error correction. The results are described as 9 observations (a full summary of the think-aloud is available, [6]).

1. Experienced programmers used strategies to correct particular syntax errors. This was error-specific; meaning that different languages or even different compilers require a new set of strategies to be learned. An example of this was:

```
a.cc:14: semicolon missing after declaration of
`video_obj'
```

The more experienced C programmers moved firstly to the line indicated (14) and after looking briefly at this line, they confidently proceeded to look above for the declaration of the *video_obj* class. This allowed them to fix this error very quickly. In contrast, the less experienced C programmers did not know what to do once they had read line 14.

Another example of an error-specific strategy was in response to:

```
h.cc:81: warning: suggest parentheses around
assignment used as truth value
```

The experienced programmers asked themselves whether this was intended to be an assignment or a conditional statement missing its second '='. Their comments showed that they understood the implications of this error as well as knowing how it could be fixed.

The less experienced programmers simply added parentheses around the assignment statement. This did not affect the time taken to reach the successful compilation stage. However, if the wrong correction was made, the time to generate a logically correct program would be greater

2. Experienced programmers used a range of strategies to approach unfamiliar error messages. Some of the compile errors were unfamiliar to even the most experienced programmers. They used a set of generic strategies to find and correct the problem. The program B error due to a missing standard IO header file caused a large number of errors, beginning with:

```
b.cc:4: type specified omitted for parameter
b.cc:4: parse error before `*'
b.cc:5: `FILE' was not declared in this scope
b.cc:5: `stream' was not declared in this scope
b.cc:5: variable or field `readFiles' declared
void
```

The less experienced programmers indicated some panic at the large number of errors. They began reading the first error and once they established that they did not know what it meant, they immediately asked for assistance or used the reference guide.

The more experienced programmers also began with the first error. However, when this did not shed light on the problem, they began reading the later errors and the code around the lines specified in the error message. After reading a few more lines of the compiler output, it was clear that the standard IO library was needed. They appeared better able to read the message efficiently. They may have experienced this type of error.

3. When strategies failed, both novice and expert programmers tried erratic alterations to code in an attempt to correct syntax errors. This finding is consistent with previous studies, which showed that students would 'tinker' almost randomly with code as a final attempt at solving syntax errors ([7] and references therein). Some of the compilation errors proved difficult for the subjects. Once they had tried a logical/strategic approach to fixing the problem, they began making what seemed like erratic changes; compiling the code after each alteration, often introducing new problems. A striking demonstration of this observation occurred in program H:

```
h.cc:5: too many arguments to function
`printReverse(char *, int)'
h.cc:93: at this point in file
```

The initial response was to remove the third argument from the declaration of *printReverse*. This caused a linking error:

```
Undefined symbol    first referenced in file
printReverse(char *, int)/var/tmp/cca21sun.o
ld: fatal: Symbol referencing errors. No output
written to exec
collect2: ld returned 1 exit status
```

This panicked three of the subjects; they then proceeded to try making ad-hoc alterations to the code. These included: deleting the first or second arguments instead of the third and changing the variables' names in the function prototype!

4. The compile-error reference guide provided enough help for the novices to perform favorably compared to the experts in terms of time efficiency. This was also true for more experienced programmers who encountered an error that they had not seen before or did not understand. This was established through the think-aloud results. (The complete log of the think-aloud is in [6].)

Program A included a variable declaration with a '*' in the wrong location, causing the error message:

```
a.cc:16: parse error before `>'
a.cc:22: `catalogue' undeclared (first use this
function)
a.cc:22: (Each undeclared identifier is reported
only once
a.cc:22: for each function it appears in.)
a.cc:27: confused by earlier errors, bailing out
```

The experienced students identified the error as soon as they looked at the problem line. By contrast, the less experienced students were not sure what the error message meant even after looking at the code. However, after looking at the reference guide they were able to correct the

problem immediately. This suggests that a syntax error reference may help beginners to correct errors more quickly.

The student with the fastest average solution time, Subject 4, also had the highest academic grades of the group and a very strong programming background. Interestingly, this was the only student who was given access to the guide, but did not refer to it. Even so, Subject 4 indicated in the survey that they would be 'very likely' to use such a reference.

5. The examples were an essential part of the reference guide explanation. The subjects who used the reference guide seemed not to understand the problem until they looked at the example code. This suggests that concrete examples are important for teaching syntax error correction.

6. The more experienced programmers were less inclined to use the reference guide, even when they did not understand an error message. However, when they did use the guide, they were able to correct problems very quickly. An example of this was the error in program A described above (observation 4).

For over 4 minutes, subject 3 inspected the error message and code before consulting the reference. In contrast, subject 1 read the error and looked briefly at the code (for less than 1 minute) before looking at the reference guide.

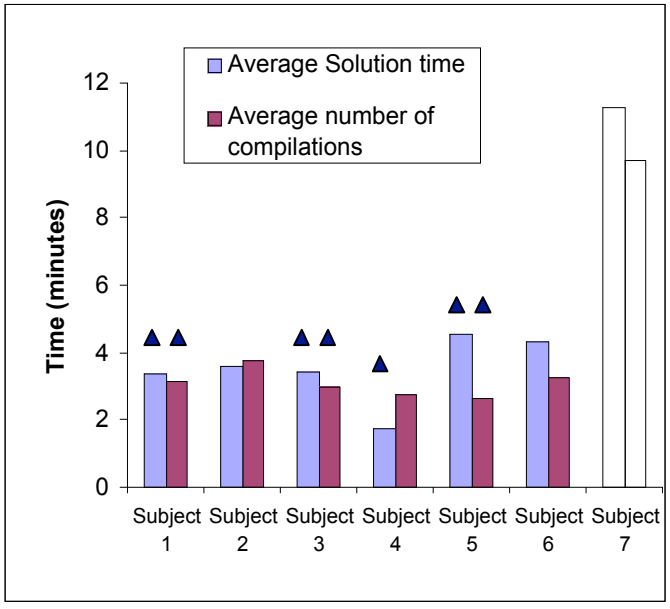

Figure 5: Average time and number of compilations to solution. Subjects 1, 3 and 5 were given the guide and used it. Subject 4 was given it but they did not use it. Subject 7 is greyed out to indicate they failed to complete 7 of the 8 exercises. All other subjects solved all problems.

As a result, the total time taken for this problem by Subject 3 was almost twice as long as that for Subject 1. These two programmers had a comparable level of programming experience and, as can be seen from Figure 5, had similar overall performance on all the tasks.

7. Error messages are often misleading. The examples above clearly illustrate this point. For example, line numbers indicating the location of the error are frequently wrong.

8. Syntax error correction can be very time consuming. This trial was developed with a set of 10 small programs (28 to 459 lines) each with one or two syntax errors. During testing of the first subject, the number of programs was revised down to 8 to keep the exercise under 45 minutes in duration. On average the subjects took just over half an hour to correct all 8 programs. Even the experienced programmers occasionally took as long as 8 minutes to correct a single error (without the reference guide). Quantitative comparisons between those who did and did not use the guide are not meaningful with this sample size. However, the post-trial questionnaire showed that all of the subjects who used the reference thought it was helpful and said they would be very likely to use such a guide in future if it were available. One individual asked if the guide developed for this study could be released on the web.

9. Experience is not the only factor that affects syntax error correction efficiency. The time it took subjects to reach successful compilation may also be related to programming proficiency. The stronger students (in terms of university results) tended to solve the problems more quickly. This was particularly noticeable for the novice C programmers. Subject 4 was academically strong, with lots of experience. However, they performed similarly to the less able and less experienced subject 5. It may be that the weaker students benefit more than the more able students from using a reference guide. This is consistent with previous work into tailoring learning to cater variation in students' ability [1].

5 Conclusion

Syntax error correction is the first step in the debugging process. It is not possible to continue program development until the code compiles. This means it is an crucial part of the error correction process. It can also be very frustrating. Compilers for complex languages like C and C++ often produce unintuitive compile errors [7]. Inexperienced programmers can have considerable difficulty understanding error messages even if they are actually quite good.

This study found that experts tended to use error specific and general strategies to correct syntax errors. The more experienced C/C++ programmers were able to respond quickly to errors through what appeared to be learned strategies and a deep understanding of error messages and the language.

The experts also used generic strategies to approach errors that they were not familiar with. Programmers at all levels (without access to reference material) began making rather ad-hoc changes to the code once their language-specific knowledge and generic error correction strategies had been exhausted.

The beginners who did use the reference, tended to go directly to the reference guide when confronted with an error message that they had not seen before or did not understand. In contrast, the more experienced programmers tended to make an attempt at fixing the bug before referring to the guide.

The key to efficient syntax error correction was the use of strategies; specific strategies for familiar errors and generic heuristics for unfamiliar errors. Language-specific experience was important for development of both specific and generic strategies. The more experienced C programmers developed a deeper understanding of the error messages. General programming experience seemed less important, but did improve the subjects' ability to correct unfamiliar errors using generic strategies.

A reference could act as a primary aid for learning syntax error correction. It would save them time while they are developing the error specific skills. Reducing the burden of syntax error correction early is essential for preventing frustration [7].

Further development of teaching resources would be beneficial. Draper [2] suggests students should provoke errors in their own code to gain a deep understanding of syntax error messages. For example, students could be asked to introduce an error by removing a semicolon from working code and observe the outcome. However, we have found that it is difficult to motivate students to do this type of exercise.

One limitation of this study is that it separated the program writing from the error correction since the author provided the set of test programs. There may be differences in the programmers' approach when they are debugging their own code as opposed to foreign code. This would be even more beneficial if the students were to generate their own guide or submit them to a central repository.

Another limitation is that the same person wrote both the reference guide and the programs with errors. It seems likely that a reference guide like this should be able to assist students with some of the most common and troublesome syntax errors. The positive survey comments suggest that the students in this study felt this type of support would be useful.

One may argue that compilers should provide better error messages. While this may improve the situation, it would not be realistic to provide detailed explanations or concrete examples because this is a special need of beginners. In our think-aloud experiments, the examples seemed to be critical for error message comprehension.

Similarly, it may be suggested that use of Integrated Development Environments would improve syntax error correction. Some of the error presented here may be avoided/corrected through using templates and/or intelligent IDE. For example, in Program D, a function is missing a return type. This would be less likely to occur with the introduction of code templates. However, we argue that most of the errors presented would still be difficult to diagnose/prevent and that being able to understand widely used compilers is an important skill.

Clearly, the key to fast syntax error correction is being able to associate an error message with a problem or set of problems. It is unrealistic to expect students to be aware of every compile error. However, techniques or heuristics for approaching errors combined with reference material may improve efficiency in syntax error correction.

The correction of syntax errors is but one of many skills that need to mastered in learning to program. There is little work in the support of programmers, especially novices, in mastering this skill. This is probably due to the correct view that runtime and logical errors are much deeper and harder to deal with. Yet, a programmer must get past compiler errors. As befits the relatively modest problem of dealing with syntax errors, we have devised a modest approach to solving it. The approach simply consists of collating compile errors that students identify as troublesome, along with these, we take the code with the problem and use it to define a simple example of the problem and the way to correct it. Placing these on a searchable web page gives a simple reference too.

The literature and our experience indicated that different classes of learning are needed to fix compile errors:

- strategic knowledge of how compiler error messages typical errors, such as the knowledge that error messages commonly appear after the problem code;
- knowledge of the programming language since this is often critical to understanding the terms used in error messages;
- understanding of how to deal with particular error messages, a form of compiled knowledge that programmers develop over time, having seen and satisfactorily dealt with the same errors repeatedly.

The current study was designed to assess the usefulness of our online resource, in spite of the limitation that it only addresses this last aspect. The survey gives a strong indication that the users liked the resource and considered it would be useful. The qualitative study confirms the role of these different types of knowledge in correcting syntax errors. It also confirms that those using it were able to find

errors quickly with the aid of the resource. Importantly, it also indicates the importance of the examples showing how to correct the errors. Our study provides a strong foundation for continuing work on building a resource for assisting novice programmers to overcome syntax errors.

References

[1] Corbett, A.T., and Anderson, J.R Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge, *User Modeling and User-Adapted Interaction*, 4(1995)253-278.

[2] Draper, S.W. Discussion on error types, Seminar on classification of errors, 2000. Available WWW: http://www.psy.gla.ac.uk/~steve/talks).

[3] Johnson, W.L., Soloway, E. PROUST: Knowledge-Based Program Understanding, *IEEE Transactions on Software Engineering,* 11(1975):267-275.

[4] Joni, S.N., Soloway, E., Goldman, R., and Ehrlich, K. Just So Stories: How The Program Got That Bug, *Proceedings of the SIGCUE/SIGCAS Symposium on Computer Literacy*. (1983).

[5] Kolling, A.M., The Blue language Journal of Object-Oriented Programming 12(1999):10-17

[6] Kummerfeld, S.K., and Kay, J. How Best to Tackle and Teach Syntax Error Correction, Basser Tech Report 2001 Available WWW: http://www.cs.usyd.edu.au/~judy.

[7] Martin, J.L. Is Turing a better language for teaching programming than Pascal, Honours Dissertation, University of Stirling, Department of Computer Science. (1996) Available WWW: http://www.holtsoft.com/turing/essay.html

[8] McConnell, A.S. *Code Complete: a practical handbook of software construction*, (1993). Microsoft Press

[9] McKeown J. and Farrell, T. Why We Need to Develop Success in introductory programming courses, CCSCCPC. (1999) Available WWW: http://homepages.dsu.edu/mckeownj/CPCCCSCpaper.html

[10] Newman, W.M., and Lamming, M.G. *Interactive System Design*, (1995). Addison-Wesley.

[11] Ohlsson, S. Some Properties of Intelligent Tutoring Systems, in *Artificial intelligence and education*, eds Lawer, R.W., and Yazadani, M. (1997). Ablex publishing.

[12] Piaget, J., and Inhelder, B. *The child's construction of quantities*, (1974) Routledge and Kegan Paul.

[13] Spohrer, J.C., Soloway, E., and Pope, E. A Goal/Plan Analysis of Buggy Pascal Programs, *Human-Computer Interaction*. (1985).

[14] Yazdani, M. Intelligent tutoring systems and overview, in *Artificial intelligence and education*, eds Lawler, R.W., and Yazdani, M. (1987). Ablex publishing.