# Evaluation of Two Textual Programming Notations for Children

**Tim Wright**[§] **Andy Cockburn**[†]

[§]School of Mathematics, Statistics and Computer Science
University of Victoria,
PO Box 600, Wellington, New Zealand,
tim@mcs.vuw.ac.nz

[‡]School of Computer Science and Software Engineering
University of Canterbury,
Ilam Road, Christchurch, New Zealand,
andy@cosc.canterbury.ac.nz

## Abstract

Many researchers have developed many programming environments for children. Typically each of these environments contains its own programming notation ranging from computer code to animated virtual 3D robots and in some case the notation consists of physical objects. While some of these notations were created by examining how children naturally describe computer programs, little research has examined how children understand programs written using these notations. Even less research has examined how children understand programs written using multiple notations.

This paper describes an evaluation that compares how children can understand computer programs written using different programming notations: conventional code, English, or a combination of the two. The children were about eleven years old and we measured speed in answering questions about computer programs and the accuracy of their answers. We found that children reading computer programs written in a conventional-style notation were more efficient (faster with no reliable difference in accuracy) than children reading programs written in English. Children with access to a combination of both notations performed between the two other conditions.

*Keywords:* Programming languages, programming notation evaluation, children, multiple notations.

## 1 Introduction

Programming computers is a remarkably useful skill. Computer programmers can instruct their computer to perform tasks including, but not limited to, adding new functions to existing applications, designing and building custom applications, and contributing to community-based open-source projects. While these tasks are far beyond the everyday tasks a non-programming user performs, he or she may benefit from programming experience while performing activities ranging from automating repetitive tasks to modifying programs that they receive from (for example) web pages (JavaScript is often embedded in webpages), word processors (generating macros using programming by demonstration mechanisms), and friends (through email).

Despite the usefulness of computer programming, only a small proportion of end-users have programming skills. To help end-users gain these skills, many researchers have created many varied programming environments (for example: Wright & Cockburn (2002), Cockburn & Bryant (1997), Repenning (2000), and Gilligan (1998)). These environments use a wide variety of different techniques ranging from programming by demonstration environments, where the computer attempts to infer a computer program based on user actions, to programming environments where users must specify exactly what the program is going to do at all times. The types of symbols users manipulate when using these environment can also vary from programming environments where users manipulate textual symbols to programming environments where users manipulate physical objects. Although this plethora of research includes many user-friendly programming environments (examples include Smith & Cypher (1999), Conway, Audia, Burnette, Durbin, Gossweiler, Koga, Long, Mallory, Miale, Monkaitis, Patten, Shochet, Staak, Stoakley, Viega, White, Williams, Cogrove, Christiansen, Deline, Pierce, Stearns, Sturgill & Pausch (2000), or Kahn (1999)), the primary programming environments that many users have access to are those that come with their office suite: a spreadsheet and a word processor's macro language. In these environments users program using textual symbols and a syntax close to what professional programmers use. This provides motivation to teach people about conventional programming syntax and semantics, and we believe the appropriate place to start teaching people about conventional notations is at school.

This paper describes an evaluation that examines how different notations help children aged about eleven years old understand and learn conventional-style syntax. It planned to test two hypotheses: that children understand computer programs faster and more accurately when presented with an English representation of a computer program and that children understand computer programs written in a conventional notation faster and more accurately when they have interacted with the conventional notation previously. Taken together, these hypotheses would show that users of multiple notations understand code faster and more accurately than users of conventional notations, with the advantage of learning the conventional notation.

The evaluation produced an surprising result: children could read and understand the conventional-style notation *faster* than they could understand code written in English. There was no difference in accuracy. This result rejected our hypothesis. Fortunately, the evaluation found that most children preferred the English-like notation. These results provide motivation for multiple notation programming environments: we can provide users with access to *both* a more efficient notation and a notation they prefer.

## 2 Motivation

Wright & Cockburn (2003) decomposed programming

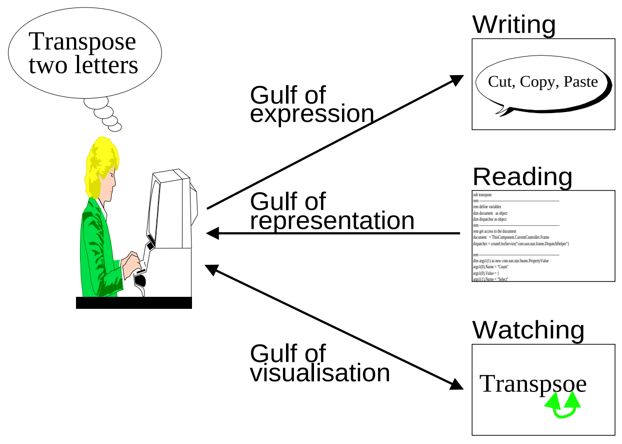


Figure 1: Three fundamental programming activities (reading, writing, and watching) and the three programming gulfs (gulfs of expression, representation, and visualisation).

into three fundamental activities: writing programs, reading programs, and watching programs run (shown in Figure 1). To recapitulate our previous work, writing is the process of converting an internal model of desired program behaviour into the symbols of a programming notation. Reading is the process of parsing a programming notation and creating understanding. Watching is the process of creating programming understanding by viewing either an animation of the notation or the behaviour specified by the notation. An analysis of how notations are used in programming environments found that many environments use different notations for different activities and some environments use different notations for the same activity (Wright & Cockburn 2003). The analysis also described ways that environments might use multiple notations to aid program understanding.

This paper describes an empirical examination on the effects of different notations on reading programming notations. There are two motivations for our experimental design. First, to control contributing factors, we only evaluate one activity — the reading activity. Second, we want to examine how well children can gain an understanding of conventional-style syntax when using different (or multiple) notations.

### 2.1 Evaluating Reading

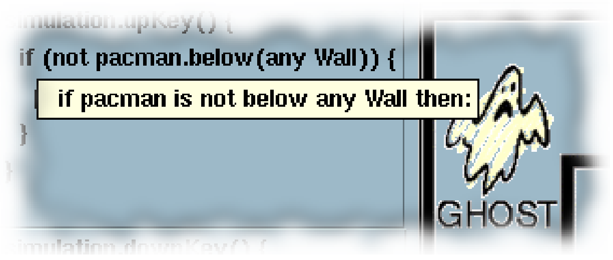
To control contributing factors and reduce experimental variance we evaluate one of our three fundamental programming activities: the reading activity. Our argument to evaluate reading rather than writing or watching follows:

**1. Modifying an existing program is a prerequisite to writing a new program.**

MacLean, Carter, Lövstrand & Moran (1990) performed a decomposition of the types of programming activities end-users perform. They ordered these activities from easiest to hardest and argued that users will start programming at the easiest level and gradually move up the scale. In their taxonomy, program modification occurs before program creation, meaning that users cannot create programs unless they can successfully modify a program.

**2. If users want to modify a program they must understand the programming notation.**

Modifying a program risks creating syntactical and semantic errors. If a user does not understand a programming notation, they will not have the skills to fix their



```
if (not pacman.below(any Wall)) {
  if pacman is not below any Wall then:
}
```

Figure 2: Multiple notation interface using tool-tips to show the English-like representation.

errors. In the reading, writing, and watching taxonomy, there are two activities that aid program understanding: reading and watching.

**3. There are few evaluations of how people interact with multiple representations of a computer program.**

Little research has examined the effects of users reading multiple notations, however much work has examined the effects on users of multiple watching notations: many program visualisation systems show program code and output and provide mechanisms to help users link the two (Ellershaw & Oudshoorn 1994, Myers 1990), and some work has examined the effects of letting people write using multiple notations (Cockburn & Bryant 1997).

**4. The writing and watching tasks contain elements of the reading task.**

When users are writing code they must continually read and revise their code to check for and fix syntactic and semantic errors. This means that we must evaluate reading first; after knowing the effects of multiple notations on reading we try to control the effects of reading and attempt an evaluation of the effects of multiple notations on writing. The watching task also contains elements of the reading task: when users are watching an animation of their program they must read the code that is being animated.

### 2.2 Teaching Conventional Code

People who to have an understanding of conventional-style code can achieve greater levels of efficiency because they are often confronted with notations designed for a computer rather than a user. They might encounter these notations after demonstrating a macro to their word processor, writing formula in spreadsheets, or even viewing error messages returned by an email system. If users understand what the notation means then users can perform tasks ranging from understanding what went wrong to modifying and debugging their program.

We hypothesise that using multiple notations increases user's understanding of conventional notation. This evaluation tests this hypothesis by dividing users into several groups and asking them several questions about static representations of computer programs. Each group is trained using one of three computer interfaces, and they are all tested on the same interface. The three training interfaces show users conventional-style versions of the computer program, English-like versions of the computer program, and both notations at the same time. In this way we can equitably compare the effect of different notations on understanding of conventional code.

## 3 Experimental Description

Each of the forty-six participants, aged about eleven years, was asked a total of twenty questions about four different

| Conventional-style | English-like |
|---|---|
| `any PacMan.contactWith(any PowerPill) {`<br>`  the PacMan.power = 10;`<br>`  the PowerPill.remove();`<br>`}` | `whenever any PacMan touches any PowerPill:`<br>`  set the PacMan’s power to 10`<br>`  remove the PowerPill`<br>`end whenever` |

Table 1: Example procedure written in conventional-style and English-like notations

simulations. The evaluation was split into two phases. Fifteen questions were in a *training phase*, and five questions in a *testing phase*. In the the training phase participants in different conditions used different computer interfaces, whereas in the testing phase all participants used the same interface. The interface used in the training phase in the first *Conventional* condition has the code is represented in a conventional-style syntax. In the *English* condition, participants answered questions about the same simulations, but the code in the training phase was represented using an English-like syntax. The third *Multiple* training interface combined the two notations in the training phase by placing English-like representation in tool-tips (see Figure 2). The Conventional and English conditions act as controls, ensuring that effects seen in the Multiple condition are due to users having access to both representations.

Fifteen participants were assigned to each condition, unfortunately one participant in the Conventional condition elected to stop her involvement in the evaluation. Her data were discarded and another participant was assigned to the Conventional condition.

While the semantics of the two notations are similar to each other (and are based on the notation described by Pane, Myers & Miller (2002)), the syntax of the two notations differ. An example program is shown in Table 1. To ensure the English-like and conventional-style notations were consistent we used Lex (a scanner generator) and Yacc (a parser generator) to produce a program that translates any program written in the conventional-style notation into the English-like notation. This transformation maps exactly one statement in the conventional-style notation to exactly one statement in the English-like notation.

After answering fifteen questions about three visual simulations in the training phase, all of the subjects, regardless of their training condition, moved to the testing phase where they answered five additional questions about a different visual simulation. The interface used in the testing phase was the same as that used in the Conventional condition of the training phase. The testing phase was shorter in order to control the amount that participants learned during the testing phase. Switching from the three interfaces used for training to the one interface used for testing allows us to equitably compare how successfully the participants learned conventional code based on the interface they used in the training phase.

## 3.1 Hypotheses

We have several hypotheses for this experiment. The first set of hypotheses (H1a and H1b) examine program understanding—we predict that participants in the English and Multiple conditions will complete tasks faster and more accurately than participants in the Conventional condition. This set of hypothesis only apply top data from the first (training) phase.

**H1a:** Children in the English condition will complete tasks faster and more accurately in the first phase than participants in the Conventional condition. We predict this hypothesis will hold as children have experience with English and not with conventional programming syntax. This hypothesis is backed up by research indicating that people naturally express programs in English (Bruckman & Edwards 1999, Miller 1981, Pane, Ratanamahatana & Myers 2001).

**H1b:** Children in the Multiple condition will complete tasks faster and more accurately in the first phase than participants in the Conventional condition. The participants will be faster because the interface is providing an English translation of the conventional code.

The second set of hypotheses (H2a and H2b) examine how well participants learn conventional syntax. They predict that participants in the Multiple and Conventional conditions will complete tasks faster and more accurately than their counterparts in the English condition. This set of hypotheses apply to data collected in the second (testing) phase.

**H2a:** Participants in the conventional condition will complete tasks faster and more accurately in the second phase than participants in the English condition. In the second phase participants in the English condition encounter conventional code for the first time. We predict that the new syntax will hinder participants’ speed at understanding computer programs.

**H2b:** Participants in multiple condition will complete tasks faster and more accurately in second phase than participants in the English condition. We predict that the participants will be faster and more accurate because they have interacted with conventional code before.

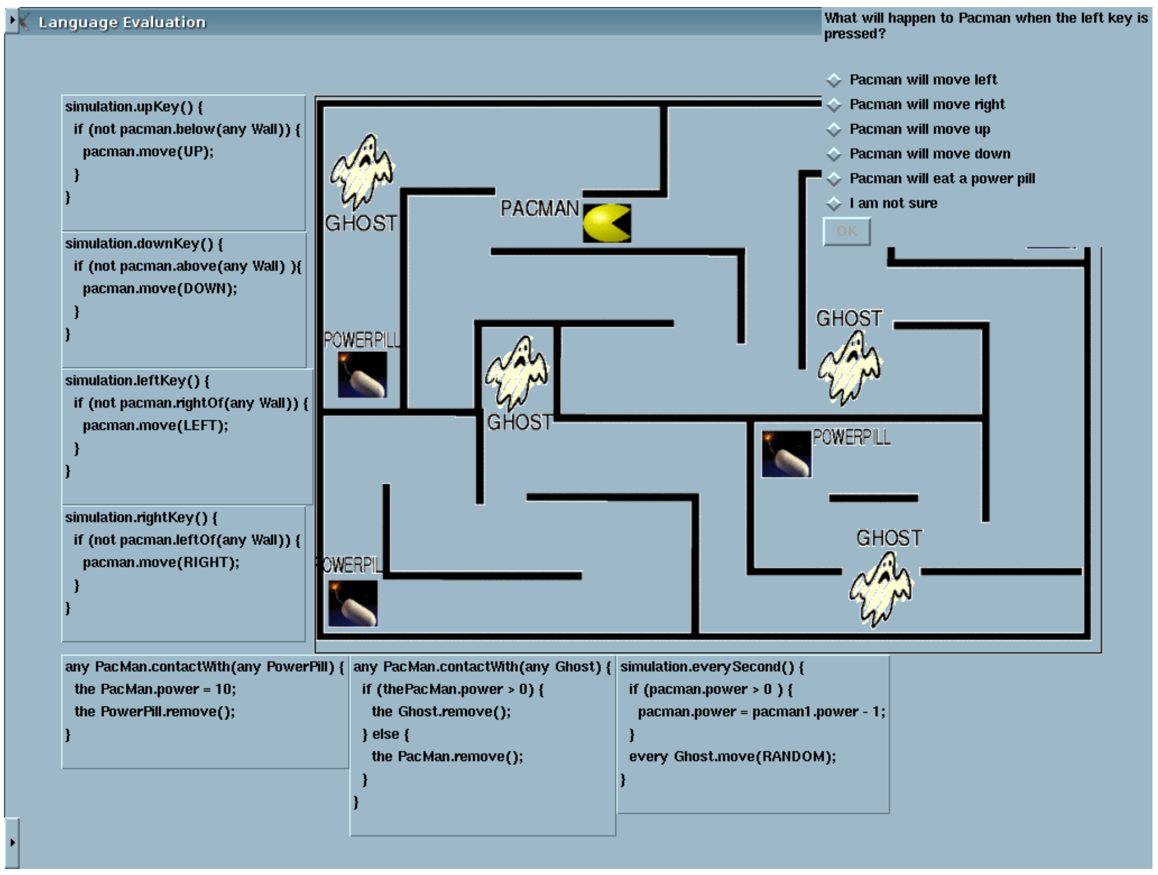
When examined together, these two sets of hypotheses (H1 and H2) predict that the multiple condition has the advantages of both the Conventional and English conditions.

A further hypothesis, H2c, predicts that participants in the Multiple condition will complete tasks in the second phase faster and more accurately than the other two conditions. If correct, this hypothesis provides strong motivation to add multiple language support to programming environments.

**H2c:** Participants in multiple condition will perform the second phase faster and more accurately than participants in the Conventional condition. Participants in both conditions have interacted with the conventional notation as much as each other, but participants in the multiple condition have had more help understanding the conventional notation.

### 3.1.1 Hypothesis Failure

If one (or more) of our hypotheses fail, we must adjust our conclusions. An interpretation of the H2 set of hypotheses failing and the H1 set succeeding is that different interface types affect performance, but not learning. An interpretation of the H1 set of hypotheses failing and the H2 set succeeding is that different interface types affect learning, but not performance. An interpretation of hypotheses failing is that the interfaces affect performance and learning equally. Our evaluation is still useful if both hypothesis fail as we can determine which interface children preferred.



```
simulation.upKey() {
  if (not pacman.below(any Wall)) {
    pacman.move(UP);
  }
}
```

```
simulation.downKey() {
  if (not pacman.above(any Wall) ){
    pacman.move(DOWN);
  }
}
```

```
simulation.leftKey() {
  if (not pacman.rightOf(any Wall)) {
    pacman.move(LEFT);
  }
}
```

```
simulation.rightKey() {
  if (not pacman.leftOf(any Wall)) {
    pacman.move(RIGHT);
  }
}
```

```
any PacMan.contactWith(any PowerPill) {
  the PacMan.power = 10;
  the PowerPill.remove();
}
```

```
any PacMan.contactWith(any Ghost) {
  if (thePacMan.power > 0) {
    the Ghost.remove();
  } else {
    the PacMan.remove();
  }
}
```

```
simulation.everySecond() {
  if (pacman.power > 0 ) {
    pacman.power = pacman1.power - 1;
  }
  every Ghost.move(RANDOM);
}
```

What will happen to Pacman when the left key is pressed?

- Pacman will move left
- Pacman will move right
- Pacman will move up
- Pacman will move down
- Pacman will eat a power pill
- I am not sure

Figure 3: Single Notation Interface

## 3.2 Subject Details

The experiment was conducted at three primary schools in New Zealand chosen to have students from a similar socio-economic background. We asked the teachers to select children who were in the middle of their class, based on math scores. The evaluation was run at the participant's schools in a quiet room. Participants were run sequentially. The forty-six participants were allocated to one of the three training conditions, giving fifteen children per group, with gender balanced between conditions. One child in the conventional condition ended her participation after answering seven of the twenty questions and her data were discarded. Approximately one third of participants from each school were allocated to each training condition.

Our implementation of the multiple notation interface used a transient notation that was shown in a tooltip. Each tooltip provided an English version of one line of conventional code. This approach is easy to integrate with other programming environments as it requires no additional screen real estate, and could also be used when browsing literate programs by adding tool-tips that contain the documentation associated with program regions (Knuth 1984). A snapshot of the tooltip is in Figure 2.

Approximately five minutes at the start of each evaluation was spent explaining the concepts behind visual simulations and the experimental interface to the participants. Each participant's involvement in the experiment lasted approximately twenty minutes.

## 3.3 Procedure

We designed four static visual simulations of comparable complexity. Each simulation had seven rules and five questions. Each question could be answered using information from only one rule. To answer the questions, participants had to predict program behaviour based on a static representation and a visual arrangement of agents. One of the questions is displayed in Figure 3. We did not use a think aloud protocol as we theorised that the cognitive load of thinking aloud might interfere differently with the cognitive load of reading and understanding the computer programs in each condition.

After each phase, participants were asked to rate two statements on the Likert scale. The two statements were: *I was confident with my answers* and *it was easy to complete the tasks*.

| Training Phase | | | |
|---|---|---|---|
| Interface | Conventional | English-like | Multiple |
| Participants | s1–s15, s46 | s16–s30 | s31–s45 |

| Testing Phase | | | |
|---|---|---|---|
| Interface | Conventional | English-like | Multiple |
| Participants | s1–s15, s46 | s14–s30 | s31–s45 |

Table 2: Experimental Design. There were two factors: condition (interface used in the training phase) and phase (training or testing). Participant s46 ended her participation early and her data were discarded.

## 3.4 Apparatus

The experimental interfaces were implemented in Python and the Tkinter user interface toolkit (Grayson 2000). Participants used a IBM R50 laptop running Debian GNU/Linux with a 15" LCD screen and a USB optical mouse. The screen's resolution was 1024$\times$768. The interface logged which questions each user answered correctly or incorrectly and the time to answer each question.

| Speed | Accuracy | |
|---|---|---|
| | Low | High |
| Low | Over confidence; weak and incorrect mental model; guessing | Understand domain well; weak but correct mental model |
| High | Strong but incorrect mental model; participant not paying attention; lack of confidence | Strong and correct mental model |

Table 3: Relationships between the dependent variables: time to answer a question and the percentage of questions answered correctly.

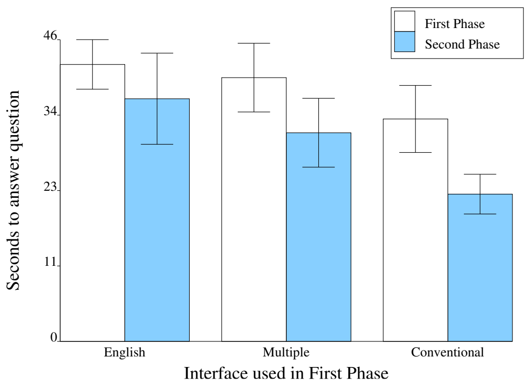


Figure 4: Average time taken to answer questions in the first and second phases. Error bars show standard error. There are reliable effects of both factors (phase: F(1,42)=21.5, p<0.01, condition: F(2,42)=3.60, p<0.05), but no interaction (F(2,42)=0.47, p=0.63).

## 3.5 Data Analysis

The experiment was a analysis of variance for factors training condition and phase. Training condition was a between-subjects factor with three levels: *Conventional*, *English*, and *Multiple*. Phase was a within-subjects factor with two levels: *training* and *testing*. Table 2 shows the design. This analysis was repeated for two dependent variables: time to answer a question and percentage of questions correct.

The dependent variables provide insight into how participants are answering the questions. For example, if a participant is taking a short time to answer the questions we can infer that they either have a strong mental model of what the code means, are just guessing, or are not paying attention. A long time to answer questions could imply that participants have a weak mental model of the code: participants are having to spend time reasoning about the code. It could also mean that participants have a lack of confidence in their skills and are continually changing their mind. We can further determine how participants are answering questions by examining their accuracy. The relationships are shown in Table 3

We log transformed the time data to stabilise the variance (Ericsson 1974). Unfortunately we could not log-transform the accuracy data: six participants spread among conditions answered all questions incorrectly in the testing phase (the log of zero is undefined). All means and standard deviations we report are from the raw data, whereas $F$ and $p$ values are taken from the log-transformed data.

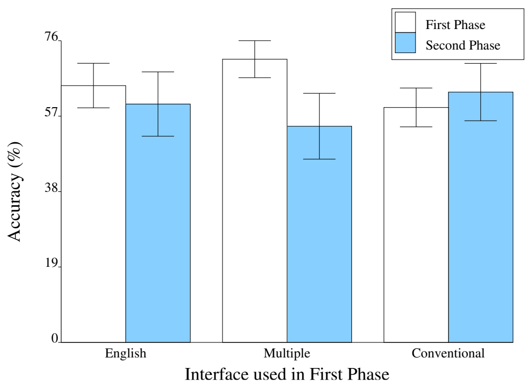


Figure 5: Average accuracy taken to answer questions in the first and second phases. Error bars show standard error. There are no reliable effects of either factor (phase: F(1,42)=1.608, p=0.212, condition: F(2,42)=0.030, p=0.971), but there is a marginal interaction (F(2,42)=1.682, p=0.198).

| | Training Phase | Testing Phase |
|---|---|---|
| English | 64 $\sigma$5.6 | 60 $\sigma$8.1 |
| Multiple | 71 $\sigma$4.6 | 54 $\sigma$8.2 |
| Conventional | 59 $\sigma$4.9 | 63 $\sigma$7.2 |

Table 4: Mean accuracies and standard errors of the three conditions in both phases. Neither factor produced a reliable effect in Anova but there was a marginal interaction. The values are graphed in Figure 5.

## 4 Results

### 4.1 Time

Anova showed reliable effects for both phase and condition:

**Phase:** Participants in the first phase were faster than participants in the second phase (a mean of 38.5 seconds for the training phase and 30.2 seconds in the testing phase: F(1,42)=21.5, p<0.01).

**Condition:** Participants needed a mean of 28.0 seconds to answer a question for the Conventional condition, 35.7 seconds in the Multiple condition, and 39.3 seconds in the English condition (F(2,42)=3.60, p<0.05).

There was no reliable interaction (F(2,42)=0.47, p=0.63). The values are shown in Figure 4.

### 4.2 Accuracy

Anova detected neither reliable effects of either factor for the dependent variable accuracy (phase: F(1,42)=1.608, p=0.212, condition: F(2,42)=0.030, p=0.971), nor any interaction (F(2,42)=1.682, p=0.198). Mean and standard error values can be found following and in Figure 5.

We are encouraged by the marginal interaction (F(2,42)=1.682, p=0.198). This appears to occur because participants in the Multiple condition are slightly more accurate than the other two conditions in the training phase but slightly less accurate than other conditions in the testing phase (see Figure 5). Further research will investigate this marginal effect.

### 4.3 Likert Questions

After each phase, participants were asked to rate two statements on the Likert scale. The two statements were: *I*

*I was confident with my answers*

| Condition | Training Phase | Testing Phase |
|---|---|---|
| English | 2 | 2 |
| Multiple | 2 | 3 |
| Conventional | 2 | 1 |

*It was easy to complete the tasks*

| Condition | Training Phase | Testing Phase |
|---|---|---|
| English | 3 | 2 |
| Multiple | 3 | 3 |
| Conventional | 3 | 2 |

Table 5: Medians for Likert questions. '1' indicates that the participant agrees with the statement, '5' indicates that they disagree. A Kruskal-Wallis test corrected for ties found only one of these differences is reliable: the first question in phase two: *I was confident with my answers* (H=6.55, df=2, N1=N2=N3=15, p<0.05).

*was confident with my answers* and *it was easy to complete the tasks*. Likert-scale ratings for the three interfaces were reliably different only for the first question in phase two (Kruskal-Wallis test corrected for ties, H=6.55, df=2, N1=N2=N3=15, p<0.05): participants in the Conventional condition were very confident with their answers (median: 1), participants in the Multiple condition were confident (median: 2) and participants in the English condition weren't sure if they were confident or not (median: 3). The other medians are shown in Table 5.

### 4.4 Comments from Participants

After taking part, participants were asked for comments about the evaluation.

**General Comments:** four participants (spread among conditions) commented the questions were hard.

**English Condition:** Ten of the fifteen participants in the English condition found the English code easier, one found the conventional code easier, and one found them about the same. However, one of the participants who found the English notation easier also said they preferred the Conventional notation: they found it more fun because they had to think more.

**Multiple Condition:** Eleven of the fifteen participants in the multiple condition liked or found the tooltips useful, however two participants said that it was more fun without the tooltips (the tasks were more challenging). One participant said having the tooltips in the first phase helped them understand the code in the second phase.

**Conventional Condition:** One of the fifteen participants in the Conventional condition commented that the notation was confusing: that the lines between words, brackets, and dots had no meaning. Another participant in the conventional condition mentioned they would like to see the programs run, and another mentioned it was easier than doing it on paper (we believe they were referring to answering the multi-choice questions). One participant in the conventional condition pulled out of the evaluation after answering seven questions as she found the questions too hard. We discarded her data.

### 4.5 Summary of Results

Participants in the Conventional condition were faster than participants in the English condition (F(2,42)=3.60, p<0.05). The speed of participants in the Multiple condition was between the Conventional and English conditions. There was no reliable difference in accuracy between any of the three experimental conditions (F(2,42)=0.030, p=0.971).

In the second phase, participants in the Conventional condition were more confident with their answers than other participants. Most participants who experienced different interfaces (those in the English or Multiple conditions) preferred the interface with an English representation. Some participants preferred having a Conventional interface as it made the questions "more fun".

## 5 Discussion

This evaluation was designed to examine two hypotheses. The first hypothesis predicted that children aged eleven years would answer questions about computer programs written using an English notation faster and more accurately than questions about computer programs written without an English notation. The second hypothesis predicted that children who interact with a conventional-style notation will be faster and more accurate at answering questions about computer programs than children who do not interact with a conventional-style notation. The analysis of the data rejects our first hypothesis and marginally supports our second hypothesis.

This rejection of the first hypothesis and the acceptance of the second hypothesis mean that children are more efficient (when measured as time and accuracy) when understanding programs written in conventional code. We believe children reading English code carefully parse the English to construct understanding, while children reading the conventional code scan over the code and gain enough understanding to answer the questions as accurately as children reading English code.

The increase in efficiency from conventional code draws doubt of the usefulness of English-like notations of computer programs. We believe that English-like notations are still useful: comments from the children indicated that they preferred using English-like notations, and other research indicates that English-like notations are useful for the writing activity (Pane et al. 2001, Miller 1981). However, English-like notations should only be provided in conjunction with a conventional-style notation.

We found the Likert data intriguing. In the testing phase, children who used the Conventional condition were most confident of their answers and children in the Multiple condition were least confident. This difference in confidence is reflected in the accuracy data: children in the Conventional condition (who were the most confident) answered a mean of 63% correct ($\sigma$7.2); children in the English condition answered a mean of 60% correct ($\sigma$8.1); and children in the Multiple condition (who were least confident in their answers) answered 54% correct ($\sigma$8.2). Unfortunately the difference in accuracy was not reliable (F(2,42)=0.030, p=0.971). One interpretation is that taking away a representation when a participant is used to multiple representations is more damaging to confidence than changing representations.

These results provide confidence in the value of multiple representations. Multiple representations provide access to a more efficient representation (conventional code) while providing access to a preferred representation (English). However, we note that once users have access to multiple notations the notations should not be taken away— although users should be able to view only one notation when they desire.

### 5.1 Limitations of Evaluation

We identified four limitations of this evaluation. The first is that we only evaluated the reading task. The second is that our results might not generalise to other populations and the third is that we used a reasonably small sample size (46 participants). The fourth is that the number of words in the English-like and conventional-style notations was different.

Only evaluating the reading task is a limitation because we can not be sure that our results transfer to the writing or watching tasks: indeed there is other research indicating children when unconstrained (they were describing programs on paper with a pen) write programs in English (Pane et al. 2001, Miller 1981). Because children naturally express programs in an English-like syntax we would expect them to be faster when using that syntax rather than using a conventional-style syntax. However, Wright (In Submission) argued that programming environments should use a syntax-directed editor. Syntax directed editors constrain programmers and remove syntactical issues. We are unsure how the research on writing notations would apply when children are constrained: when they writing programs using a syntax directed editor.

Another limitation is that our results might not generalise to other populations. In this evaluation our participants were children aged 10 or 11 from three decile nine primary schools in Wellington. Decile nine indicates that the children are from high socio-economic areas[1]. Further research needs to predict how our result generalises to children from different age groups, socio-economic backgrounds, and different language backgrounds. We are also unsure if these results generalise to adults: adults have higher reading skills than children aged eleven so the parse time of the English notation might not slow them down.

The third limitation is our small sample size: 46 participants. This is a limitation because we might be making a type-2 error (rejecting correct hypothesis due to large standard error). We expect that some of our marginal results might become reliable if the evaluation was repeated with more participants: in particular we expect that the marginal interaction between phase and condition for factor accuracy would become reliable (F(2,42)=1.687, p=0.198).

The final limitation is that the English-like and conventional-style notations contained different numbers of English words: the conventional notation had fewer words (a mean of 84 words for the conventional-style versus a mean of 148 words for the English-like notation per simulation). An evaluation to mitigate this problem is proposed in the future work section (Section 6.1).

## 6 Summary

This paper described an evaluation examining how quickly and accurately children understand computer programs using different notations. We expected that children using an English-like notation would be faster and more accurate than children using a conventional notation, and that children using a conventional notation would understand the conventional notation better. While we did observe the second effect, we observed the opposite of the first effect: children interacting with a conventional notation were faster to understand computer programs than children interacting with an English notation. This result has implications for researchers who believe programming using English-like notations is better than programming using conventional-style notations: these researchers must show that use of an English-like notation aids the writing or watching tasks—this evaluation proves that an English-like notation can slow the reading task.

Comments from children indicate they preferred the English notation, although some preferred just having the conventional notation. This provides more motivation for multiple notation programming environments: people can have access to both a more efficient notation (the conventional code) and a notation they prefer (the English code).

[1]New Zealand Schools are ranked from decile one to ten based on the population the school draws its students from. Decile one is the lowest and decile ten is the highest.

This result also influences the design of programming environments: because some users preferred just having the conventional notation, users of multiple notation programming environments should be able to turn one notation off. This design pointer is easily achieved when using tool-tips to show the additional notation. The main limitation of this evaluation was that it only examined the reading task.

### 6.1 Future Work

There is much scope for future work. In particular, we need to determine whether the children were faster because there is less information to parse or because they have a stronger mental model. We also need to determine whether this difference in efficiency is present in our other two programming activities: writing and watching. Particular research questions include:

- Is the difference in efficiency due to a longer parse time of English code, or is it due to a stronger mental model? There are several ways to answer this question. The first is to modify the English-like and conventional-style notations so that they contain the same number of English words and re-run the evaluation. The second is to re-run the evaluation using adults. Adults can read English faster than children, so the difference in parse-time should be less noticeable. Re-running the evaluation with adults would also tell us if the result transfers to adults.
- Does the difference in efficiency transfer to the writing and watching activities? To answer this question we need to run evaluations concentrating on the effects of multiple notations on the writing and watching activities. Unfortunately when people are watching or writing a program they are also reading the program: to figure out what the program is doing and to track down bugs. This evaluation gives us a baseline for each notation and we can use this baseline to examine the effect of a notation on just the writing or watching tasks.

## 7 Acknowledgements

We thank Brooklyn, Wadestown, and Wilton primary schools for access to their students. This evaluation benefited from discussions with Fiona Smith, a primary school teacher.

## References

Bruckman, A. & Edwards, E. (1999), Should We Leverage Natural-Language Knowledge? An Analysis of User Errors in a Natural-Language-Style Programming Language, *in* ‘Human Factors in Computing Systems: CHI ’99 Conference Proceedings (USA)’, pp. 207–214.

Cockburn, A. & Bryant, A. (1997), ‘Leogo: An equal opportunity user interface for programming’, *Journal of Visual Languages & Computing* **8**(5–6), 601–619.

Conway, M., Audia, S., Burnette, T., Durbin, J., Gossweiler, R., Koga, S., Long, C., Mallory, B., Miale, S., Monkaitis, K., Patten, J., Shochet, J., Staak, D., Stoakley, R., Viega, J., White, J., Williams, G., Cogrove, D., Christiansen, K., Deline, R., Pierce, J., Stearns, B., Sturgill, C. & Pausch, R. (2000), Alice: Lessons Learned from Building a 3D System for Novices, *in* ‘Human Factors in Computing Systems: CHI 2000 Conference Proceedings (USA)’, pp. 486–493.

Druin, A., ed. (1999), *The Design of Children’s Technology*, Morgan Kaufmann Publishers.

Ellershaw, S. & Oudshoorn, M. (1994), Program visualization - the state of the art, Technical Report TR 94-19, Department of Computer Science, University of Adelaide.

Ericsson, K. A. (1974), Problem-Solving Behaviour with the 8-puzzle II: Distribution of Latencies, Technical Report 432, Department of Psychology, University of Stockholm.

Gilligan, D. (1998), An Exploration of Programming by Demonstration in the Domain of Novice Programming, Master's thesis, Victoria University of Wellington.

Grayson, J. E. (2000), *Python and Tkinter Programming*, Manning Publications Co.

Kahn, K. (1999), A Computer Game To Teach Programming, *in* 'Proc. of the National Educational Computing Conference'.

Knuth, D. (1984), 'Literate programming', *The Computer Journal* **27**(2), 91–111.

MacLean, A., Carter, K., Lövstrand, L. & Moran, T. (1990), User-Tailorable Systems: Pressing the Issues with Buttons, *in* 'Human Factors in Computing Systems: CHI '90 Conference Proceedings (Seattle, WA, USA)', pp. 175–182.

Miller, L. A. (1981), 'Natural language programming: Styles, strategies, and contrasts', *IBM Systems Journal* **20**(2), 184–215.

Myers, B. A. (1990), 'Taxonomies of Visual Programming and Program Visualization', *Journal of Visual Languages & Computing* **1**, 97–123.

Pane, J. F., Ratanamahatana, C. A. & Myers, B. A. (2001), 'Studying the language and structure in non-programmers' solutions to programming problems', *International Journal of Human-Computer Studies* **54**, 237–264.

Pane, J., Myers, B. & Miller, L. (2002), Using HCI Techniques to Design a More Usable Programming System, *in* 'IEEE Symposia on Human-Centric Languages and Environments', Stresa, Italy, pp. 198–206.

Repenning, A. (2000), Agentsheets : an Interactive Simulation Environment with End-User Programmable Agents, *in* 'Proceedings of the IFIP Conference on Human Computer Interaction (INTERACT '2000, Tokyo, Japan)'.

Smith, D. C. & Cypher, A. (1999), Making Programming Easier for Children, *in* '(Druin 1999)', Morgan Kaufmann Publishers, chapter 9.

Wright, T. (In Submission), Collaborative and Multiple-Notation Programming Environments for Children, PhD thesis, University of Canterbury.

Wright, T. & Cockburn, A. (2002), Mulspren: a MUltiple Language Simulation PRogramming ENvironment, *in* 'IEEE Symposia on Human-Centric Languages and Environments', Arlington, Virginia, pp. 101–103.

Wright, T. & Cockburn, A. (2003), A Language and Task-based Taxonomy of Programming Environments, *in* 'IEEE Symposia on Human-Centric Languages and Environments', Auckland, New Zealand, pp. 192–194.