An Investigation into Keystroke Latency Metrics as an Indicator of Programming Performance

Richard C. Thomas
School of Computer Science & Software Engineering, M002
The University of Western Australia
35 Stirling Hwy, Crawley 6009, Western Australia
richard@csse.uwa.edu.au

Amela Karahasanovic
Simula Research Laboratory
PO Box 134,
1325 Lysaker, Norway
amela@simula.no

Gregor E. Kennedy
Biomedical Multimedia Unit
The University of Melbourne
Parkville 3010, Vic, Australia
gek@unimelb.edu.au

Abstract

Typing has long been studied in psychology and HCI, and strong cognitive models for transcription typing exist. The goal of the present research was to test if there is any correlation between students' keystroking speed and performance while they are programming. We present the results from two studies with computer science students conducted in different contexts. Keystroke timings were recorded while they worked on Java and Ada source code. Quality of their programming work was measured mainly in terms of completeness. In the controlled experiment that lasted six hours, 39 students undertook three change tasks on a 6000 LOC Java application. In the field study, data was collected over 6 weeks from 141 students while they worked unsupervised on Ada programming in first year laboratories. In both cases there were highly significant (P=0.001), moderately strong, negative correlations between speed and coding performance. With additional development, these techniques may have promise for user modelling and assessment as well as in educational diagnostics.

Keywords: Digraph latencies, empirical methods, programming performance, keystroke model, chunking.

1 Introduction

One of the greatest challenges facing teachers of introductory computer science units is the high rate of attrition. Sometimes 25% or even 40% of enrolled students do not succeed. Associated with this is a desire to be able to spot the potential problem students as early as possible.

Timely warning can aid the teacher to provide remediation, at least in a well-resourced world. The student too would be able to make more informed choices when presented with evidence that progress is not encouraging, perhaps prompting a change of study methods or to review wider choices before it is too late.

In this paper we present work in progress on the use of low level, continuous keystroke monitoring as a means to identify potential problems early on. In particular we investigate whether the latency, or delay, between certain keystrokes correlates with objective measures of programming performance. We have checked this for students in two countries: one in a controlled experiment developing Java code for a few hours; the other using first year laboratories over a few weeks, where Ada was the programming language.

Keystrokes are well understood. There is a strong cognitive theory of typing, reviewed below. Furthermore keystroke latencies can be applied to user modelling. For instance their potential as a means for user authentication has been investigated (Joyce and Gupta, 1990; Monrose and Rubin 2000).

Latencies have shown promise elsewhere. In the LISTEN project to teach children to read aloud, Beck *et al* (2003) found that latency before saying a word had promise as a feature for assessment of reading achievement.

2 Models of Typing

Typing has been studied in psychology, cognitive science and human-computer interaction. Although not mainstream, there is a solid body of knowledge on which to base the present investigation.

Newell (1990) gives a very clear account of the process of transcription, or copy, typing. Typing is a pipeline process, basically: perceive a chunk, determine the spelling, obtain a letter, and execute a keystroke. The pipelining effect allows, for example, reading a word to happen in parallel with pressing the key for some previously identified letter, and one hand can operate in parallel with the other. It has been shown that SOAR cognitive models accurately reflect some observed typing phenomena. John and Newell (1989) give some detail of the perception of a chunk in these models. It could be a word or syllable or a single character depending upon circumstances, such as whether it contains random letters or is partially covered up.

From psychology there are several phenomena (Salthouse 1986) that inform the present investigations. Foremost for

us is the word initiation effect: the first keystroke of each word is typed about 20% slower than the others. This is probably a manifestation of the parsing and chunking process. Furthermore word order does not affect typing, so nonsense sentences are as fast as semantically meaningful ones. In contrast non-words, e.g. HDRN rather than HARD, are typed up to 40% slower. Meaningless material is typed more slowly than normal text. Conversely comprehension is not a factor in typing performance: one does not copy type faster for understanding the content, nor do those who have high understanding of the material type faster.

The keystroke-level model in human-computer interaction (Card, Moran and Newell 1983) presents a set of guidelines to determine how long a task will take. Times are given for mouse clicks, keystrokes, moving the hand from mouse to keyboard, and the important Mental Operator. The Mental Operator can be thought of as a mental chunking function. To estimate task execution time the procedure is: first determine the number of operators, insert Mental Operators at likely chunk boundaries and then compute task time from the standard values. Heuristics are provided as to where to insert Mental Operators, such as at the start of typing a word. So chunking is recognised as contributing to overall speed.

A very common effect from psychology is the Power Law of Practice – the speed up that occurs when repeating a task. It has been shown that computer science students increase their overall typing rate during their first two years of university (Thomas 1998).

These results provide the theoretical basis to develop a set of metrics to investigate typing speeds of programmers. An important caveat must be given, though. The above work mainly relates to expert behaviour in copy typing tasks. When editing program source texts, a problem-solving mode is likely to be more dominant. This will affect thinking times and chunking. However one would expect that at the word level, such as typing the word *print*, students would revert to skilled behaviour.

3 Goals of the Research

The goal of this research was to test if there is any correlation between students' typing speed and performance while they are programming. Based on the anecdotal evidence from observation in the laboratory classes we expect that more senior or domain competent students tap and click faster. However, the previous empirical studies do not address this directly. Accordingly we build on them and present in Section 3.1 simple metrics for measuring subjects' typing characteristics followed by our hypotheses in Section 3.2.

3.1 Digraph Latency Metrics

The raw data has each keystroke listed, together with a timestamp in milliseconds when the key is pressed. In principle the latency of a keystroke can simply be the difference in the timestamps between successive keystrokes, the key down to key down time.

We allocate each keystroke into one of the following types

- **A** : alphabetic characters, e.g. *a, L*
- **N** : numerics, *0..9*
- **C** : control keys, e.g. *CONTROL* and then a *C*
- **O** : other keys, e.g. *()[]/*=;.* and *CR, SHIFT, SPACE, DELETE*
- **B** : all the browsing (positioning) keys, such as *HOME*, *left*

A digraph is a pair of sequential keys, such as *th* and *he* when typing the word *the*. We choose to give each digraph a type, according to its two keys:

- **A, N, C, O** or **B** if both are that type
- **H** when exactly one is type **B**
- **E** when they are different but neither is **B**

The rationale is that the typing of a series of **A, N** or **C** digraphs is more likely to occur in a single chunk, whereas the **E** digraphs are likely to be on the boundary of a chunk and therefore longer. This is illustrated in Figure 1.

```
TEXT:     typing words or 23476 [-]
Digraph:   AAAAAEEAAAAEEAEENNNNEOOO
```

Figure 1: The top row shows what is keyed on a US keyboard, and below is the corresponding digraph type. The E digraphs occur at or near likely chunk boundaries

O digraphs are also relatively likely to be on a chunk boundary. **H** digraphs are expected to be slow because the arrow keys are often some distance away from the main alphanumeric keypad. So the hand takes time to transfer to or from a **B** key, rather like the keystroke model (Card *et al* 1983), regardless of any additional thinking time.

The metrics themselves are the median latency value for each digraph type for each user while typing in files of program source text. Latencies in highly automated tasks, such as copy typing, are often not normally distributed: Joyce and Gupta (1990) observed log normal, for instance. Our observed data is even more right skewed due to the addition of thinking time and other delays, even lunch breaks. Accordingly we use the median, as have others (Genter, 1983; Salthouse 1986). For brevity, latency will henceforth be used to indicate median key down to key down time of a given digraph type, measured in milliseconds.

3.2 Hypotheses

We would expect that people who can program well would have more fluency than those who cannot. Such fluency would at least partially derive from thinking in terms of high level abstractions. The poor programmer would, at an extreme, perceive some code as almost

meaningless, for example the *:=* in assignment or the semicolon statement terminator. From typing theory the poor programmer would be expected to have smaller chunks and slower typing of the perceived non words. However there would be parts of source code that would be perceived as normal, such as *print*.

Given this background and the digraph types, our hypotheses were as follows:

1. High chunking times indicate low programming performance
2. Browsing/positioning (**B** digraphs) latency is independent of programming performance
3. Alpha or numeric string typing is also independent of programming performance
4. Alpha or numeric string typing (**A** or **N** digraphs) is faster than programming special syntax (**O** digraphs)
5. Chunking times (**E, O, H** digraphs) are slower than other digraphs

4 Study One

The results reported below were gained from the secondary analysis of data that had been collected for a controlled experiment into Think Aloud and related protocols (Karahasanovic *et al* 2004).

4.1 Participants and Setting

The participants had completed second or third year computer science at either The University of Oslo or Oslo College, Norway. They were experienced in Java. They were asked to volunteer for a day during a vacation and were paid. The mean age was 24, range 20-38, one female. One person dropped out towards the end of testing, yielding 38 participants for the main Think Aloud experiment, but he was included below as his typing and quality scores exist.

These students attended an experimental session at Simula Research Laboratory, away from their institutes, starting either morning or afternoon and staying about 6 hours. They were given a meal at an appropriate hour, e.g. lunch for 30 minutes. There were 4-10 students per day.

4.2 Materials and Procedure

Each participant used a PC, with a standard Scandinavian keyboard, running Windows. A remote connection was established using Terminal Services Client to a Windows Server. Thus each person saw a normal desktop, and could access Borland JBuilder for Java development. Tasks were downloaded and uploaded to the server via a web based application called Simula Experiment Support Environment (SESE) (Arisholm *et al* 2002).

Keystroke and other data, such as mouse clicks and window focus events, were captured by the User Action Recorder (UAR), part of GRUMPS (GRUMPS 2004). Settings were such that a complete record of the keys actually pressed was obtained for each participant, together with millisecond timing. Keystrokes could usually be attributed back to an application or even Java class, but this was not always practical.

On arrival at the Lab there was a short welcome and introduction period. Subjects were informed about experiments goals, procedure and asked to sign a consent form. After doing that, each person was given a unique username and password to be used during the day. Then everyone undertook a training task, involving a small change to about 400 lines of Java code. This was to familiarise themselves with the compiler, systems and experimental procedures in SESE. The next stage was to undertake a standard calibration task for up to an hour. All participants in experiments at this laboratory take this.

A training session followed, depending upon the treatment of each person. Thus Think Aloud persons practiced that; people using retrospective think aloud were so trained and students using the Feedback Collection Method (Karahasanovic *et al* to appear) used that. The control group had no training. The first two groups worked in individual rooms accompanied by an observer, while the others were accommodated in a lab of up to 8 people plus an observer.

After training the main tasks began. There were three change tasks on an application of about 6000 lines of Java, each intended to be harder that its predecessor. The last task was given to ensure that none of the subjects finished before the end of the time allocated for the experiment and was not intended to be used in any analysis. We did not want subjects who worked faster to disturb other participants. The main session lasted about 3 to 4 hours. After this all participants answered a questionnaire and attended an interview.

During the experiment each student uploaded amended source code after they were satisfied with each task. Two independent, experienced programmers then assessed the submitted work. Marking was on the basis of completeness, scoring 4 for a perfect solution, down to zero for a non-attempt. Total scores for each person were the sum of the individual task scores. For the analysis below, only the calibration and first two main tasks were considered as the third had an uneven completion rate.

4.3 Results

An example of the raw keystroke data is shown in Figure 2. In post experiment data cleaning and transformation most keystrokes were recovered into digraphs corresponding to the types in Section 3.1. Repeating keys were excluded. Keystrokes attributed to the JBuilder application were analysed, as this was where program source was edited. Calculation of latencies for digraph type was then fairly straightforward.

Programming quality marks were summed over the three tasks, giving a maximum of 12. This is not an ordinal score, because it consists of three components. Accordingly a Spearman Rank Correlation Test was performed for each digraph type against programming score. The latency data are tabulated in Table 1: the mean count is digraphs per user; the mean value is for the set of

latencies derived from the participants. Table 2 gives the results of the correlations of latency with programming scores, and a scatter diagram for the E digraph is in Figure 3. We chose a significance factor of p=0.001 because we intended to perform many tests and wished to reduce the chance of a false positive. '**' indicates significant results in the tables below.

ActionID	Time	XML
1251079	1045143002268	<k>VK_T</k> <ch>t</ch> <v>Y</v> <s>N</s> <c>N</c> <a>N N <r>1</r>

Figure 2: Example of full keystroke data. The actual key pressed and displayed is determined from the XML. Shown is the letter 't', not a repeating key.

Digraph type	Mean count	Mean Latency (millisecond)	Standard deviation
B (browse)	3921	109.36	57.47
N (numeric)	144	170.74	41.57
A (alphabetic)	1225	174.23	44.68
O (other)	943	195.92	36.44
C (control)	127	317.10	109.60
E (edge not B)	1084	473.90	179.10
H (to/from B)	655	662.10	250.50

Table 1: Latency statistics by digraph type, Java students

4.4 Discussion

The mean (of median) latencies for each digraph type support hypotheses 4 and 5. It was surprising how fast the browsing, B, digraphs were; given that repeating keys were excluded, it shows how rapid the motor response can be.

The mean number of digraph occurrences for each type are plausible in that fewer numeric, N, and control, C, keystrokes would be expected. For the other types, it appears that a reasonably large sample has been obtained.

Turning to the correlations in Table 2, hypothesis 2 is supported by the result for browsing, B, digraphs. There is no case to reject the null hypothesis that they are independent.

Hypothesis 3 holds for numeric, N, digraphs but we have to reject it for alpha, A, digraphs even at the p=0.001 level. This is a surprise. It is possible there is some factor related to the auto-completion of identifiers in JBuilder, but this is still under investigation.

Digraph type	Spearman Rank Correlation	P-value
B (browse)	0.196	0.244
N (numeric)	-0.248	0.139
A (alphabetic)	-0.519	0.001 **
O (other)	-0.485	0.006
C (control)	-0.485	0.003
E (edge not B)	-0.516	0.001 **
H (to/from B)	-0.385	0.018

Table 2: Programming score correlations with digraph type, Java students

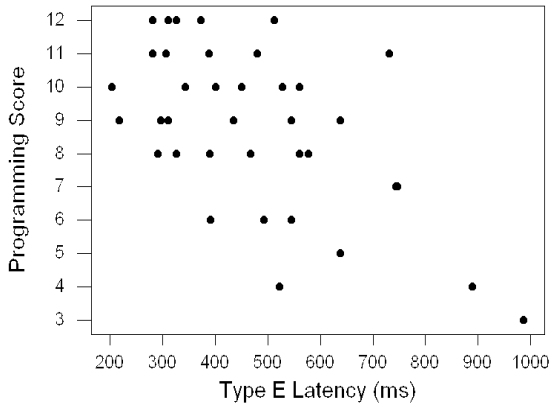

Figure 3: Scatter diagram of programming score against latency of type E digraphs, Java students

Hypothesis 1 is strongly supported by the very significant correlation of the E digraph with programming score. This digraph was always expected to occur on chunk boundaries. To a lesser extent so were O and H types, and these correlations stand out from B and N.

No predictions were made about C digraphs as their use is often a feature of keyboard type or interface preferences, for example control S and a menu command have identical effects.

The correlations are of medium strength, around –0.5, and there is substantial scatter as seen in Figure 3. These metrics are not sufficient to measure programming performance under experimental conditions, of course. The negative value supports hypothesis 1.

5 Study Two

The previous experiment took place in a controlled situation. In contrast we wanted to discover whether the correlations hold for students working in a natural setting, over a longer period. We used data that was collected a year earlier as part of the development of GRUMPS. We had specified that reliable collection over several weeks

was required and had some control over the actual items monitored. It was intended to use the data for more than one investigation. A fuller description of this background is available in Thomas *et al* (2003).

5.1 Participants and Setting

The participants were students at Glasgow University taking the compulsory CS1P unit in first year computing science. This was available to freshers as well as repeating students. The Ada programming language was taught, which was unfamiliar to most new students. Monitoring data was recorded on 141 people, with full details, especially marks, available on the 125 in the present analysis.

All students had a weekly scheduled laboratory session of two hours where Ada was the topic once per fortnight, the other week being used for database work. Students were free to come and go and use the computers as they pleased outside allocated laboratory sessions. During laboratory sessions, they were encouraged to work on study packs, but how strictly this was enforced depended on their individual tutors.

5.2 Materials and Procedure

The labs were equipped with PCs running Windows XP and had standard UK keyboards. At the end of January 2003 one lecture was partly devoted to telling the students about GRUMPS and its monitoring of activity on these machines. After an explanation of ethical issues everyone was invited to sign a consent form. About three quarters of those present did so. The final count of 141 unpaid volunteers covered about a third of the enrolled class. One person subsequently asked for his consent to be stopped as he felt his task bar was cluttered.

The same UAR as in Study 1 was automatically invoked at Windows login but only for people who had previously given consent. All of their sessions were monitored during a six-week period from 10 February.

Digraph type	Mean count	Mean Latency (millisecond)	Standard deviation
B (browse)	984	211.52	45.82
N (numeric)	67	181.90	51.53
A (alphabetic)	2574	184.86	40.08
O (other)	2087	253.93	49.80
C (control)	77	282.10	142.00
E (edge not B)	1902	398.20	132.0
H (to/from B)	501	651.70	257.6

Table 3: Latency statistics by digraph type, Ada students

At any time the user could switch off the UAR. For this study it collected Hidden Window Focus events, mouse clicks and keystrokes. Full keystroke data, as in Figure 2, were only gathered when the current process was Adagide.exe, the Ada programming environment. Further no window titles were stored, just process names, unlike in Study 1. These settings made it far easier than in Study 1 to identify programming keystrokes. Usernames were coded before transmission into the secure data repository. Hence the privacy of participants was respected.

There were two forms of assessment of programming ability. Both were closed book and were conducted shortly before the UAR started recording keystrokes. The scores contributed to the final grade of the unit, being marked as usual by a team of academics and demonstrator markers. None were involved in this research except for one person.

The first assessment was an exam in the lab. Students were given the question two weeks in advance. The actual exam was closed book except for a reminder sheet on Ada syntax. Marking rewarded completed functionality. The second assessment was a written test on Ada programming, taken during a lecture.

Digraph type	Written Test		Lab Exam	
	Corrl.	P	Corrl.	P
B (browse)	-0.093	0.314	-0.119	0.196
N (numeric)	-0.333	0.000 **	-0.220	0.016
A (alphabetic)	-0.183	0.042	-0.177	0.050
O (other)	-0.218	0.015	-0.283	0.002
C (control)	-0.083	0.404	-0.266	0.007
E (edge not B)	-0.276	0.002	-0.299	0.001 **
H (to/from B)	-0.312	0.000 **	-0.401	0.000 **

Table 4: Correlation of lab exam and written test scores with digraph type, Ada students

5.3 Results

2655 UAR sessions were recorded, comprising 4.7M actions over 1767 hours of interaction. This averages a total of about 19 hours per user, but much of this was not in Adagide.

Analysis was almost the same as for Study 1. However the UK and Scandinavian keyboards are not identical. To input some of the type O characters, such as square and curly brackets, the Alternate Graphics key must be depressed in the Scandinavian version. Slightly different tables were required to decode the keystroke XML data for the two studies, and the same program text does not generate exactly the same digraphs because of the keys pressed.

Another difference was that the Ada scores were on ordinal scales. Accordingly Pearson correlations were computed. Corresponding results for Experiment 2 are shown in Tables 3 and 4 and Figure 4.

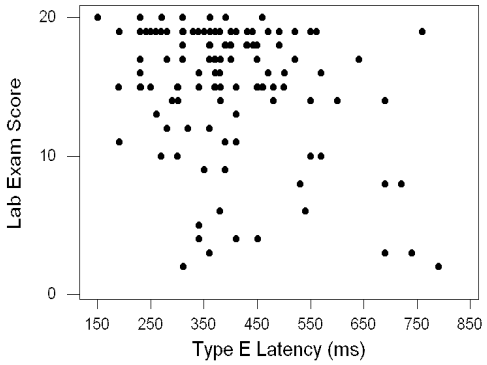

Figure 4: Scatter diagram of lab exam score against latency of type E digraphs, Ada students

5.4 Discussion

The digraphs counts are broadly in line with those in Study 1, with relatively few numeric and control types. Although the grand totals are strikingly similar at 8099 and 8192 for Studies 1 and 2 respectively, the big difference is that the Java students did far more browsing or positioning with the keyboard. Indeed all Java students used it, whereas 9 in Ada had less than 10 browsing digraphs, 4 having zero. So the Ada students used the browse keys less, but as beginners they had small program texts.

The Ada latencies also lend support to hypotheses 4 and 5. The alpha and numeric digraphs are faster than the E, H or O. Comparing with Sudy 1, the Ada students are marginally slower on A and N, substantially so on B. This is consistent with the notion that the first year Ada students were generally less practiced at keyboard skills than the second and third year Java students. Other explanations such as prior typing training could be a factor but are unknown.

The correlations support hypotheses 1 and 2. Chunking digraph times are important, E and H for the lab exam and H also for the written test. B, browsing, digraph times are not significant.

The surprise here is that N is highly significant for the written test. Outrageously, could the N digraph result suggest memory is also involved in the written test? People who remember well will be able to type long numbers more fluently.

The fact that both the written test and lab exam show a relation with digraph latency suggests that knowledge and performance in Ada are involved.

The general level of the probabilities for all the lab exam correlations does suggest there might be another factor for the Ada students. Perhaps they enter university with low typing skill and when they practice programming their tapping speed increases due to power law speed-ups. Perhaps the metrics merely reflect this rather than programming performance. If so, the A and B latencies would be faster when total digraph count is higher. In fact this can be rejected, r=-0.129, p=0.163 and r=-0.093, p=0.351 respectively. This could be investigated more deeply for individual typing speeds following Genter (1983).

An alternative possibility could be that these metrics are a proxy for learning in the lab: if they practice their Ada they will learn the concepts, not so much learn to type. Indeed written test score does increase with total digraph count, r=0.229, p=0.020. For the lab exam the hypothesis is rejected, r=0.174, p=0.081. A weakness in this argument is that the written test was taken before the digraphs were collected, though.

Thus this Study has provided support for the main hypotheses and also produced evidence that the metrics are related to a learning effect.

6 Summary of the Results

As these two studies were conducted in different contexts (different countries, different background of students) and under different conditions (controlled experiment, field study) it is more likely that our results would yield inconsistent results. On the contrary the two sets of results support each other.

The correlations are generally of a lower strength in Study 2. This is reasonable because the typing metrics have been computed for all Ada programming activity over a six week period, not the typing to complete the tasks in the assessed work. One would expect a controlled experiment to yield a stronger result than an extended field study.

We are still investigating why this result has occurred. We have background cognitive and other data on the Ada students for this. More detailed comparison of the weakest and strongest students may give insight on an explanation and also on how to refine the metrics.

Our hypotheses and indeed results are consistent with some aspects of cognitive load theory (Sweller 1999). For instance we expect those with good conceptual schema of programming to have some faster digraphs. It is possible that digraph metrics might reveal something of learning and problem solving preferences, such as adoption of means-ends search.

6.1 Validity

The results show one possible factor in a predictive model of programming performance. It is not sufficient in itself to be a substitute for other forms of assessment.

Our sample sizes are not small, but we cannot be sure that they are representative of their cohorts or the wider student population. In both countries the participants were volunteers. For Java one can imagine that the more confident students volunteered. For Ada it may be that

those attending the recruitment lecture were keener than their peers; it was the last before a break.

Although we have confidence in the UAR there could be some features in the timing of events that are unknown. Certainly UAR interval times are not normally distributed, showing a periodicity that may derive from process scheduler characteristics. There is also a small amount of evidence that the Java timestamps may have been distorted on occasions because of server congestion.

The measures of programming performance were not exhaustive. It is possible there is bias as an important factor was completeness, which might favour faster students.

6.2 Lessons Learned

The collection of keystroke data is controversial. There is always the danger of privacy violations. Although extensive efforts were made in this regard, both studies highlighted unexpected difficulties. In the Ada data, program source sometimes has author names typed in. String searches have revealed a list of them. For Java it was expected that special usernames for the study would protect identities. Indeed this was the case except when someone used the web browser to read their email on some third party provider.

An advantage of the metrics is that individual keystrokes need not be stored. These problems would then largely disappear.

The cost of cleaning and transforming the UAR data was rather high. Generic data has been commented upon as problematic in this regard (Reeves and Hedberg, 2003). It is expected that in due course our data will be amenable to more generic support functions.

7 Conclusions and Future Work

We have presented two studies in widely varied situations that present essentially the same result. Namely that digraphs associated with chunking boundaries appear to be a promising feature in the assessment of programming performance. We have shown this for two programming languages involving 39 and 125 participants taught in two countries.

There is also evidence from the first year students that performance in programming really was a factor rather than merely learning to type. First, the written test on Ada was just that, hand written. Second better programming scores were associated with higher practice in Ada as measured by total digraphs.

Work is required to enhance the metrics, both to focus on important features like chunking and also to take account of future understanding of the basis of these results.

There are many potential applications for our techniques. One is to be able to do rapid assignment of people to groups based on an approximation of programming performance. Our measure, taken with other data, may be quite effective.

Another application is in plagiarism detection. If a student claims to be the author of a program, they could do the work again and latencies might detect the unwary. This would require extensive development.

There are strong grounds to investigate these ideas further. In one initial step, we are instrumenting BlueJ to collect some data on programming and then to display results on a web page in near real time.

8 Acknowledgements

The authors are grateful to the students of the Department of Informatics at the University of Oslo, Oslo University College and Department of Computing Science at the University of Glasgow who participated in our experiments. We thank Unni Nyhamar Hinkel and Annette Kristin Levine for contributions on the experiment on the think-aloud methods. We thank Per Thomas Jahr and Bent Østebø Johansen who tested and assesed the Java solutions delivered by the subjects of the think-aloud experiment; and Gunnar Carelius for his technical assistance. We also thank the GRUMPS and CS1P teams at Glasgow, especially Phil Gray, Iain McLeod and Rebecca Mancy, and gratefully acknowledge the funding provided for GRUMPS by the UK's EPSRC (GR/N381141). UWA PC307 students are thanked for data transformation tools: Chris Harris, Raymond Hon-man Yuen, Hayden Albrey, Christopher Tzehong Jee, Hernani Binti Nahrawi, Mohammad Yaqoob Siddiqui, Shen Zhang. Richard Thomas gratefully acknowledges support from the Simula Research Laboratory Guest Researcher programme.

9 References

Arisholm, E., Sjøberg, D.I.K., Carelius, G. and Lindsjørn, Y. (2002): A Web-based Support Environment for Software Engineering Experiments. *Nordic Journal of Computing*, **9**(4): 231–247.

Beck, J.E., Jia, P., Sison, J. and Mostow, J. (2003): Predicting Student Help-Request Behavior in an Intelligent Tutor for Reading. In *User Modeling 2003 Conference, Johnstown, PA, USA, June 22-26*. 303-312. Brusilovsky, P., Corbett, A. T. and de Rosis, F. (eds). Lecture Notes in Computer Science, **2702**, Springer Verlag.

Card, S.K., Moran, T.P. and Newell, A. (1983): *The psychology of human-computer interaction*. Hillsdale N.J., Lawrence Erlbaum.

Genter, D.A. (1983): The acquisition of typing skill. *Acta Psychologica*, **54**, 233-248.

GRUMPS: Generic Remote Usage Measurement Production System, Department of Computing Science, University of Glasgow. http://grumps.dcs.gla.ac.uk/. Accessed 16 August 2004.

John, B.E and Newell, A. (1989) Cumulating the science of HCI: From S-R compatibility to transcription typing. In *Proceedings of CHI, 1989*, 109-114, ACM.

Joyce and Gupta (1990): Identity authentication based on keystroke latencies. *Communications of the ACM*, **33**(2): 168-176

Karahasanovic, A., Anda, B., A., Arisholm, E., Hove, S.E., Jørgensen, M., Sjøberg, D.I.K., Welland, R. Collecting Feedback during Software Engineering. Experiments. *Journal of Empirical Software Engineering*, to appear 2005.

Karahasanovic, A., Fjuk, A., Sjøberg and Thomas, R.C. (2004): A controlled experiment to evaluate the reactivity and usefulness of the think-aloud tool. *Proc. Information Resources Management Association International Conference IRMA'04*, New Orleans La, USA, 1033-4, New Idea Group Publishing.

Monrose and Rubin (2000): Keystroke dynamics as a biometric authentication. *Future Generation Computer Systems*, **16**: 351-359.Newell, A. (1990): *Unified theories of cognition*. Harvard University Press.

Reeves, T.C. and Hedberg, J.G. (2003): *Interactive Learning Systems Evaluation*. Educational Technology Publications, Englewood Cliffs, NJ.

Salthouse, T.A. (1986): Perceptual, cognitive, and motoric aspects of transcription typing. *Psychological Bulletin,* **9**(3): 303-319.

Sweller, J. (1999): *Instructional design in technical areas*. ACER Press, Camberwell, Victoria, **43**.

Thomas, R.C. (1998): *Long term human-computer interaction*. Springer Verlag, London.