Which Aspects of Novice Programmers' Usage of an IDE Predict Learning Outcomes?

Gregory Dyke
Ecole des Mines de Saint-Etienne,
France
158, Cours Fauriel
42023 Saint-Etienne CEDEX
+33 6 50 56 30 99
gregdyke@gmail.com

ABSTRACT

We present the preliminary analysis of a study whose long term aim is to track IDE usage to identify novice-programmers in need of support. Our analysis focused on the activity of 24 dyads on a 3 week assignment. We correlated frequencies of events such as use of code generation and of the debugger with assignment grades, final exam grades, and the difference in rankings within dyad on the final exam. Our results show several significant correlations. In particular, code generation and debugging are correlated with the final grade, and running in non-debug mode is correlated with differences in ranking. These results are encouraging as they show that it is possible to predict learning outcomes with simple frequency data and suggest more complex indicators could achieve robust prediction.

Categories and Subject Descriptors

K.3.2 [**Computers and education**]: Computer and information Science Education

General Terms

Measurement, Experimentation, Human Factors.

Keywords

Log data, Novice programmers, IDE usage

1. INTRODUCTION

Within the field of the Learning Sciences and for Technology Enhanced Learning in particular, the role of the tutor and their ability to monitor student activity has been considered crucial [16]. Several approaches have looked at ways of constructing high-level interaction indicators from low-level log data in order to assist such tutor monitoring [12], [4]. For the teaching of programming, several studies have tracked programming activity at various levels in order to correlate it with learning outcomes and thus to suggest best practices to students or to identify "at-risk" students [7], [10], [11].

In these studies, however, metrics have been mostly limited to compiling activity, resulting errors and standard software metrics such as lines of code, code complexity, amount of time spent, etc. and not to other more fine-grained activities, particularly those afforded by IDEs such as code-completion and generation. Furthermore, while there is a general consensus as to the benefits of pair programming [3], [8], [13] studies investigating such practices have not looked in detail at the programming activity itself.

The field of Educational Data Mining has shown that it is possible to detect many behaviors from tracking data, such as gaming the system, off-task behavior and guessing in intelligent tutoring systems [2]. Such techniques have also been used to detect affect (boredom, frustration, etc.) through analysis of compilation behaviour. Other studies encourage us to think that low-level data [15] can be used in many instances to predict higher-level outcomes.

In this paper, we examine novice programmers usage of the Eclipse IDE whilst working in pairs, and the relationship between this usage and learning outcomes as measured by their grades. We first explain how our educational context led us to conduct such as study and describe the data we chose to collect and the hypotheses we had concerning how this data might help to predict learning outcomes. We then present and discuss the results of a preliminary analysis which indicate: that several indicators related to use of the full possibilities of the IDE correlate significantly with learning outcomes; that the frequency indicators we calculated are a better predictor of the final exam grade than of the grade of the assignment the students were working on during the data collection; and that similar indicators both positively correlate with the final exam grade and with the within-dyad difference between grades on the final exam in spite of these two outcomes having a very low degree of correlation with each other.

2. STUDY DESCRIPTION AND METHODOLOGY

2.1 Context

This study was conducted on a class of 124 first year students at a French engineering school (a 3-year graduate-level school whose entry exam is usually prepared during the two years after high-school). The salient features of this population are that they are extremely high achievers who are used to working hard but who are mostly not interested in computer science and programming. Unfortunately for them, a significant number of entry-level jobs obtained by graduates from this school feature programming responsibilities. Furthermore grades in Computer Science courses are not expected to be a discriminating factor in students' overall pass rate and, in contrast to the problems reported by many introductory programming courses for non CS majors, the dropout rate is very low (<1%). However, in our department, we are concerned over students' lack of willingness to seek out help during lab work (with many TAs present) – making it difficult for us to help out the students who are in difficulty – and our difficulty in fostering the habit of regularly running the written code, in order to make sure that it works. We are also curious as to whether there are situations where our forcing students to work in pairs (due to lack of available hardware) has a negative impact on their individual learning.

During the *Object-Oriented Programming in Java and UML modeling* course, students have four 90mn lab sessions (small assignments) and three 3h lab sessions (during which they work on a larger assignment) over the course of seven weeks. In all these sessions, students work in pairs. They are encouraged to continue working on their assignment outside of the lab. This is the students' second programming course and their first contact with the Java programming language. It is also their first experience with using the Eclipse IDE (as opposed to the previous course where they edited with a text editor and compiled on the command line).

Students are evaluated via a grade on the assignment, which they must turn in a week after the final lab session (same grade for both members of the pair), and via a grade on a final exam which tests knowledge about object-oriented programming in general, UML modeling and the java language. These grades are given out of 20 with 20 being the best grade and 10 being the pass grade. Half points may be awarded. Although grades on this scale may follow a normal distribution, it is customary to make the first 5-8 points relatively easy to acquire, ramping up the difficulty slightly to reach a pass grade. The last 5 points are particularly hard to achieve (i.e. anything above 15 is considered an excellent grade).

2.2 Indicators of Success in Programming

Use of best practices such as PSP (Personal Software Process [9]) during programming courses has been shown to drastically improve the quality of code produced [11]. The metrics used to evaluate these practices are based on low-level indicators which describe the amount of effort invested in a program, its size and its shortcomings. Effort can be evaluated through the amount of time spent on each file coding and/or browsing.

The prescriptive approach of PSP can be contrasted with various descriptive approaches, in particular in the observation of novice programmers (which may be more appropriate in this case as we can hardly expect novice programmers to better understand programming simply by encouraging them to copy the best practices of expert programmers). Several studies have shown that the persistence of compilation errors from one compilation instance to the next can be positively correlated with failure [10] and with poor scores on midterms [18].

Further work has confirmed these results and indicates that working in short multiple sessions is also beneficial to the final grade [7]. It also adds that time spent on a task is not correlated with the grade: students can spend little time through boredom or because the task is too easy; students can spend a lot of time because of perfectionism or because of multiple successive failures. They further note that students overestimate their compilation frequency, the quality of testing which their code has undergone and the proper breaking up of their code into modules. Other studies have confirmed these findings and show that beginning work early is also a strong predictor of assignment grade [6].

2.3 Further hypothetical indicators

We can complete these indicators by further hypothetical indicators from our own teaching experience which we expect to be correlated with grades : respect of naming conventions, using the capabilities afforded by the IDE (browsing, automatic indentation, refactoring, suggestions for correcting compilation errors), and frequency of execution (i.e. testing). As compiling is automatic in the Eclipse IDE, the EQ (Error Quotient) metric described by [10] cannot be reproduced but may be simulated by the amount of time taken to fix a compilation error.

As one of our long-term goals would be to automatically detect "at risk" dyads, we chose to forego collecting the kinds of verbal and gesture data used in other studies (e.g, [17], [14]). However, some studies [15] have shown encouraging results with regard to analyzing quality of collaboration based on low-level data (eye-tracking and presence/absence of speech over time). These results suggest that we might get some indication of the nature of the within-dyad collaboration through mouse activity (which can be used for deictic gestures) along with moving from one tab to another (changing source file), which could indicate potential losses of alignment and agreement [1] within the dyad (in spite of having no indication as to which member of the dyad is the author of this activity).

2.4 Data Collection and Preparation

Several tools have been proposed to collect data on programming activity [7], [11]. We chose HackyStat [11] because of its integration with the Eclipse IDE and the possibility of extending it with new captors. We added functionality to capture some mouse activity, testing activity (adding capture of run mode to the existing debug mode), use of code generation functionalities (auto-complete, error correction), and use of the information about errors (quick fix, which gives additional info about compile errors and suggests ways to correct them).

This resulted in the following event types and subtypes being captured:

- File edition
  - Open
  - Close
  - Save
  - Edit (triggered by a pause in editing)
  - Code generation (auto-complete, etc.)
  - Program unit creation (when a new class or method is defined)
- Appearance of a new compilation error
- Use of the quick fixes interface to see suggestions
- Testing and debugging
  - Run

    - Run in debug mode
    - Set/unset breakpoint
    - Reach breakpoint
    - Step over/into
- Hover (the mouse is pointed long enough for an information tool-tip to appear)

Data and consent forms were collected for 60 student dyads, one monad and one triad. For each group, we also associated the grade on the assignment and the grades of the individuals making up the group.

Surprisingly (based on the experience of previous years), a large proportion of students used their laptops rather than the computers provided in the labs, mainly because of confusion over how to move projects between computers in Eclipse (in spite of explanations). For the first week of the assignment, we have data for nearly all dyads. Over the course of the assignment this number decreases. Because we had not provided the Hackystat plugin for installation on personal laptops, we only have a complete data set (data collected over all 3 weeks) for 28 groups. While this is not necessarily a problem (as we do not intend to provide universal indicators for novice programmers but to identify indicators which are reusable for us in similar conditions in subsequent years), for reasons of data homogeneity and ease of analysis, we restricted the analysis presented in this paper to those 28 groups. Of these, several groups worked in parallel on a laptop and on the lab computers, actually using the lab computers very little. We identified 3 groups whose editing activity was two standard deviations lower than that of the next least-active group. Their data was also not considered for this analysis, again for reasons of homogeneity as many statistical methods are not robust against outliers.

Before proceeding further with the analysis, we compared the grades of the 25 remaining groups (24 dyads and 1 monad) against the grades of all 62 groups, summarized in Table 1. We can see that groups who did their lab work on the lab computers scored 1.6 points fewer on average on the assignment. The students who worked partly/entirely on their own computers during labs have a slightly wider spread on the exam grades, but these grades do not differ significantly.

Table 1. Comparison of grades between the whole population and those for whom we have a complete data set

		128 students / 62 groups	**49 students / 25 groups**
Assignment grade (dyad)	Mean	13.56	11.98
	Std dev.	3.35	3.97
Exam grade (individual)	Mean	10.35	10.21
	Std dev.	3.28	2.97

For each of these 25 groups, frequencies for each sub-type (or type in absence of subtype) were computed. These frequencies were correlated with the average, lower and higher exam grade for each group (Average Grade, Min Grade and Max Grade), with the Assignment Grade for each group, and, where applicable, with the difference in within-dyad grade on the exam.

We were unable to easily compute values regarding persistence of compilation errors over time as the captor only provides the appearance of compilation errors and does not (as we had assumed) record their resolution.

3. RESULTS

Because normality cannot be assumed, either for the frequencies, or for the grades, the Spearman Rank Correlation test was used. For the same reason, the difference in grades on the exam was measured by the difference in rank. Because of ties, the p-values for the test are not the exact values, but are estimated. In Table 2, we report the main results for correlation with between frequencies and grade outcomes, showing only significant values (*p<0.05, **p<0.01, ***p<0.001).

Table 2. Correlation between frequencies and grade outcomes (non-significant values are not reported)

	Assignment grade	Average grade	Max grade	Min grade	Rank difference
Save			0.517**		0.434*
Edit			0.435*		0.465*
Code Generation		0.623**	0.79***	0.44*	0.448*
Change Tab	0.486*		0.45*		
Normal Run			0.443*		0.6**
Debug Run			0.47*		
Reach Breakpoint		0.45*	0.519**		

No significant correlations were found for the other event types, with the highest correlations for Mouse Hover and use of Quick Fixes being with Max grade at around $\rho=0.3$ ($0.20<p<0.25$) and the lowest being with Rank difference at around $\rho=0.1$ ($p>0.75$). The various debug events followed their Normal and Debug run counterparts with slight correlations ($0.1<p<0.2$) with the exam grade values and the Rank difference. Open, Close, Compilation Errors and Program Unit Creation provided universally low correlations ($p>0.5$). The number of compilation errors has particularly low correlation

4. DISCUSSION

4.1 Average grade

The highest levels of significance are found in the correlations with exam grades. This would seem to indicate that certain uses of the IDE, in particular use of code generation and having breakpoints placed in such a way that they get triggered, are either indicators of good understanding or lead to an improved understanding of programming concepts. This is most easily explained for breakpoints, since placing them well is both the easiest way to follow unexpected code execution and a strong sign that students understand where the problems in their code arise.

The correlation with code generation might perhaps be best explained as an indication that students understand the situations in which Eclipse knows to generate code and thus that they have sufficient understanding of the Java programming language to trust in the code generated by Eclipse.

4.2 Assignment grade

It seems almost paradoxical that data collected on fulfilling a given assignment should be better correlated with the grade on a subsequent exam than on the grade of the assignment itself. This is less surprising when we consider that a perfectly correct program might not have the functionality required for a good assignment grade. Furthermore, as the students had a week after the final lab to turn in their assignment, it may be that we are missing the data of the "sprint" to bring the assignment up to full functionality.

Transitioning from one tab to another is the only factor which correlates with the assignment grade. This may be explained by two hypotheses: first, the assignment suggests that students should proceed by related functionalities, performing changes through the call tree, frequently moving from one Java class to another; second, a frequent behavior of TAs, when assisting students, is to move through all the files (and their parent tabs) to assess the current state of the program.

4.3 Max and Min grades and Rank difference

The hypothesis that, under certain conditions, one member of the dyad might benefit more than the other seems to hold out as several other activities associated with "normal" use of the IDE (saving, editing, testing the program) are correlated with the grade of the better performing student of the dyad and with the difference in rank on the exam.

As we do not give any instructions on how work should be distributed within the dyads, a likely hypothesis to explain this is that one student is a more frequent "typist" and that all these activities lead to loss of alignment [1], i.e. their respective understandings of the problem and its solution do not evolve jointly, as one student speeds ahead before the other can understand what he is doing. Again, the causation could go either way (or both): students with a good understanding may use the IDE more and students who use the IDE a lot may gain better understanding.

A similar hypothesis would explain why running the application in debug mode and reaching breakpoints are associated with a higher maximum grade, but not with difference in rank, whilst running the application in non-debug mode is associated with both: debugging slows things down, allowing alignment to be maintained or re-established. Indeed, testing the program in non-debug mode is the most significant predictor for rank difference and the pairing is the third most significant correlation overall.

Only code generation is significantly correlated with average, maximum *and* minimum grade. Surprisingly, however, it is also correlated with rank difference. Our first hypothesis to explain this was that high grades (min, max and average) would correlate with rank difference and thus that factors associated with one would necessarily be associated with the other. While this is slightly case for the min ($p=0.06$) and the max ($p=0.17$) – a self fulfilling result as dyads where one student has an extreme grades are more likely to see a high rank difference, assuming a distribution centered around the mean – it is emphatically not the case for the average ($\rho=0.09$, $p=0.7$). While we cannot rule out this hypothesis, it seems likely that, given the variety of kinds of code generation (automatic brackets, imports, refactoring, error correction, auto-completion, etc.), a closer look might reveal some kinds of code generation to be correlated with high grades and others with high rank differences. In particular, some kinds of code generation allow "silent" insertion of code of "minor" importance to understanding, such as imports, and could contribute to preservation of alignment.

Another surprising result is that, although we had expected tab changes to cause loss of alignment, they are in fact correlated with maximum grade, but *not* with rank difference. Maybe further investigation into the data will help us explain this – possibly, as suggested above, this is related to TA activity.

Some of the effects found on the maximum grade (particularly those related to debugging) were also present on average and minimum grades, but at non-significant levels of correlation, indicating that, under certain conditions such as pairing students by ability [3] or following a "change role every 15mn" protocol [13], a) the difference in rank might be diminished and b) using the full functionality of an IDE in pairs for learning to program might be beneficial and lead to an increase in understanding for both students.

4.4 Other event types

Several of the events we collected showed no significant correlation with the final grades.

It is hardly surprising that opening and closing files showed no correlation: indeed, it would be hard to explain why this would be true.

Mouse activity was the most far fetched of our indicators. Although the chosen hover activity did not produce any meaningful results, the fact that others hope that mouse activity may be an indicator of affect [17] suggests that we should not give up straight away on this type of event.

As a surprise to us, the number of compilation errors is very poorly correlated with grades. Our unexpected inability to measure how soon errors are corrected has prevented us from exploring the matter further and finding the results the state of the art lead us to expect.

The poor correlations for the use of quick fixes is disappointing, but this metric and mouse hover are among the strongest differentiators between differences in rank and grades and might thus be associated with other events (e.g compile error, quick fix, code generation, in rapid succession) to create robust predictors of poor grades or high within-dyad differences.

5. FURTHER WORK

The analysis presented in this paper is a first step towards our goal of being able to identify "at risk" students as early as possible, based on their activity. We plan to construct robust indicators based on these encouraging preliminary results and to test them on the data for week one (which has data for nearly all the groups).

As shown in Figure 1, it might be possible to construct predictors based on the statistics we currently have alone. Indeed, placing a threshold on the code generation metric at 350 events would yield 58% recall/100% precision and a threshold at 460 events would yield 100% recall/75% precision. However, we believe that examining how events unfold over time and establishing a more detailed hierarchy of event categories will produce even better results.

We therefore plan to use exploratory sequential data analysis tools such as Tatiana [5] to discover the patterns in dyads identified as being typical (e.g. particularly successful groups, particularly unsuccessful groups or groups for which the indicators we have established so far are particularly poor).

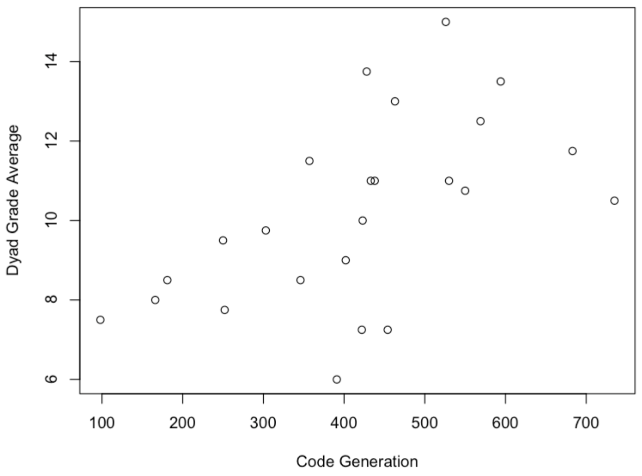

Figure 1. Plot of number of instances of code generation against average grade for each dyad.

For the purposes of detection of "at risk" students, causation is not an issue. However, the question arises with regard to how best to intervene in order to help these students. Aside from extra instruction or explanations, it may be possible that encouraging students to follow certain potential best practices (debugging, following the call tree when writing code, etc.) would be beneficial. Further studies must be carried out to validate whether our findings do in fact give hints as to best practices for using an IDE in a learning situation and whether following these practices leads to better learning.

Finally, correlation between these indicators and detailed grading information, both on the assignment and the final exam, may allow us to predict more precisely which competencies students lack.

6. CONCLUSION

In this paper, we presented the results of preliminary analysis on a study designed to correlate novice-programmer pair-programming activities using the Eclipse IDE with their grades on the assignment, in the final exam and with the within-dyad difference in rank on the final exam. Our long term goal is to use such data to identify and assist "at risk" students. Our results show that the use of code generation functionalities and coming across breakpoints while debugging are both significantly correlated with the final grade. Frequencies of editing, saving, code generation and testing the program in non-debug mode are significantly correlated with within-dyad rank differences on the final exam. Only the frequency of transition from one tab to another proved to be significantly correlated with the grade on the assignment the students were working on when the data was collected.

These results are highly encouraging as they suggest that it will be possible to develop robust indicators of students who are likely to perform poorly and of dyads who are likely to have large differences in rank on the final exam. They also show the value of collecting data in more detail than information about compiling, particularly as our results are strong predictors of performance on the exam, indicating that they may help us access student understanding.

Further avenues for study include a look at the sequentiality of events in order to identify patterns which increase the distinguishing power of the already identified predictors, determining how little data is necessary to predict learning outcomes (earlier detection leads to earlier intervention), answering the question of how best to support "at risk" students, and determining whether the factors we have identified so far could form the basis of best practices for learning to program with an IDE.

7. ACKNOWLEDGMENTS

Our thanks to Gauthier Picard for allowing this study to be run in his class and sharing detailed data regarding the grading, Adrien Hery for assistance with data collection, and to all the students who agreed to take part in this study.

8. REFERENCES

[1] Baker, M. 2002. Forms of cooperation in dyadic problem-solving. In P. Salembier & T. H. Benchekron (Eds.), *Cooperation and complexity in sociotechnical systems* (Vol. 16, pp. 587–620).

[2] Baker, R. S., Corbett, A. T., Koedinger, K. R., and Wagner, A. Z. 2004. Off-task behavior in the cognitive tutor classroom: when students "game the system". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria, April 24 - 29, 2004). CHI '04. ACM, New York, NY, 383-390. DOI= http://doi.acm.org/10.1145/985692.985741

[3] Braught, G., MacCormick, J., and Wahls, T. 2010. The benefits of pairing by ability. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (Milwaukee, Wisconsin, USA, March 10 - 13, 2010). SIGCSE '10. ACM, New York, NY, 249-253. DOI= http://doi.acm.org/10.1145/1734263.1734348

[4] De Laat, M. F., M, C., and Wegerif, R. 2008. Facilitate the Facilitator: Awareness Tools to Support the Moderator to Facilitate Online Discussions for Networked Learning. *In Proceedings of the 6th International Conference on Networked Learning* (pp. 80–86). Halkidiki, Greece.

[5] Dyke, G., Lund, K., and Girardot, J.-J. 2009. Tatiana : an environment to support the CSCL analysis process. *CSCL 2009*. Rhodes, Greece, 58–67.

[6] Edwards, S.H., Snyder, J., Pérez-Quiñones, M.A., Allevato, A., Kim D., and Tretola, B., 2009. Comparing Effective and Ineffective Behaviors of Student Programmers. *ICER'09*, August 10–11, 2009, Berkeley, California, USA.

[7] Fenwick, J. B., Norris, C., Barry, F. E., Rountree, J., Spicer, C. J., and Cheek, S. D. 2009. Another look at the behaviors of novice programmers. SIGCSE '09. ACM, New York, NY, 296–300.

[8] Hanks, B., McDowell, C., Draper, D., and Krnjajic, M. 2004. Program quality with pair programming in CS1. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Leeds, UK, June 28-30, 2004). ITiCSE '04. ACM Press, New York, NY, 176-180.

[9] Humphrey, W. S. 1995. *A Discipline for Software Engineering*. Addison-Wesley.

[10] Jadud, M. 2005. A first look at novice compilation behaviour using BlueJ. *Computer Science Education*, 15(1):25–40.

[11] Johnson, P. M., Kou, H., Agustin, J., Chan, C., Moore, C., Miglani, J., Zhen, S., and Doane, W. E. 2003. Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined. In *Proceedings of the 25th international Conference on Software Engineering*. IEEE Computer Society, Washington, DC, 641–646.

[12] May, M., George, S., and Prévôt, P. 2008. A closer look at tracking human and computer interactions in web-based communications. International Journal of Interactive *Technology and Smart Education*, 5(3): 170–188.

[13] McDowell, C., Werner, L., Bullock, H., and Fernald, J. 2002. The effects of pair-programming on performance in an introductory programming course. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science* Education (Cincinnati, Kentucky, February 27 - March 03, 2002). SIGCSE '02. ACM, New York, NY, 38-42. DOI= http://doi.acm.org/10.1145/563340.563353

[14] Murphy, L., Fitzgerald, S., Hanks, B., and McCauley, R. 2010. Pair debugging: a transactive discourse analysis. In *Proceedings of the Sixth international Workshop on Computing Education Research* (Aarhus, Denmark, August 09 - 10, 2010). ICER '10. ACM, New York, NY, 51-58. DOI= http://doi.acm.org/10.1145/1839594.1839604

[15] Nüssli, M.-A., Jermann, P., Sangin, M,. and Dillenbourg, P. 2009. Collaboration and abstract representations: towards predictive models based on raw speech and eye-tracking data. *CSCL 2009*. Rhodes.

[16] Petrou, A. and Dimitrakopoulou, A. 2003. Is synchronous computer mediated collaborative problem-solving "justified" only when by distance? Teachers' point of views and interventions with co-located groups, during every day class activities. In *Proceedings of the International Conference on Computer Support for Collaborative Learning 2003*.1-10.

[17] Rodrigo, M. T., Baker, R. S., Jadud, M. C., Amarra, A. M., Dy, T., et al. 2009. Affective and behavioral predictors of novice programmer achievement. *ITiCSE '09*. Paris, France. 156-160.

[18] Tabanao, E., Rodrigo, M. M. T. and Jadud, M. 2008. Identifying at-risk novice programmers through the analysis of online protocols. *Philippine Computing Society Congress 2008*, (UP Diliman, Quezon City, February 23-24, 2008).