No Tests Required: Comparing Traditional and Dynamic Predictors of Programming Success

Christopher Watson
School of Engineering and Computing Sciences
University of Durham
United Kingdom
christopher.watson@dur.ac.uk

Frederick W.B. Li
School of Engineering and Computing Sciences
University of Durham
United Kingdom
frederick.li@dur.ac.uk

Jamie L. Godwin
School of Engineering and Computing Sciences
University of Durham
United Kingdom
j.l.godwin@dur.ac.uk

ABSTRACT
Research over the past fifty years into predictors of programming performance has yielded little improvement in the identification of at-risk students. This is possibly because research to date is based upon using static tests, which fail to reflect changes in a student's learning progress over time. In this paper, the effectiveness of 38 traditional predictors of programming performance are compared to 12 new data-driven predictors, that are based upon analyzing directly logged data, describing the programming behavior of students. Whilst few strong correlations were found between the traditional predictors and performance, an abundance of strong significant correlations based upon programming behavior were found. A model based upon two of these metrics (Watwin score and percentage of lab time spent resolving errors) could explain 56.3% of the variance in coursework results. The implication of this study is that a student's programming behavior is one of the strongest indicators of their performance, and future work should continue to explore such predictors in different teaching contexts.

Categories and Subject Descriptors
K3.2 [**Computer and Information Sciences Education**]: Computer science education.

General Terms
Measurement, Experimentation, Human Factors, Verification.

Keywords
CS1; predictors of success; programming behavior; learning strategies; learning styles; prediction; watwin; error quotient;

1. INTRODUCTION
Programming courses have a reputation for high difficulty and failure rates [1], and as a result, predicting a student's performance in a first programming course is a well studied problem. Although early work focused upon using standardized aptitude testing [5] over the past fifty years, various predictors have been proposed. These include a range of demographic, psychological, academic, and cognitive factors, such as: previous programming experience [2,18,25], math background [17,24], science background [2,6], behavioral traits [2], self-esteem [4], learning styles [7,9,10], learning strategies [3], and attributions of success [8,11,18].

However studies to date are limited by a lack of verification, and a tendency to yield inconsistent results [23]. Additionally the previously researched predictors require the use of lengthy tests to gather predictive data. The learning strategies assessed by [15] for instance requires students to complete over 80 questions. Given potentially high enrollment numbers, the use of such tests to collect predictive data can take a considerable amount of time for an instructor to process. Even if a test was indicative of performance, by the time it was processed, it may be too late for students to withdraw, or for instructors to intervene to prevent students from failing [3]. The criteria used for prediction is a further limitation of these studies. Whilst psychological or background traits may be indicative of performance, they are not directly related to the regular programming behavior of a student, or the programming tasks which they are required to perform. The previously researched predictors therefore cannot reflect changes in the learning behavior and progress of a student over time.

Possibly due to these shortcomings, recent research [12,16,23] has moved towards exploring more data-driven approaches where aspects of the programming behavior of students (such as number of errors made) is directly logged by augmenting an IDE, and the resulting datasets analyzed to identify relations with performance. Compared to traditional tests, the main advantage of this approach is convenience. As predictions are made using directly logged data, neither an instructor, nor student has to process a batch of lengthy tests. Predictions are formed based upon aspects of a student's programming behavior, therefore can reflect changes in their learning progress over time. Also, as well as being able to dynamically identify struggling students, such predictors can be applied to drive an expert system, so that students can be provided with appropriate interventions when required. However to date, no paper has compared the performance of these data-driven predictors against test-based predictors. Contributions include:

1. The verification of 38 traditional test-based predictors of performance where previous research yielded inconsistent results, or a small number of studies had been conducted.
2. An exploration of 10 new data-driven dynamic predictors.
3. The first paper to perform a comparative evaluation of the performance of both data-driven and test-based predictors.

2. RESEARCH DESIGN

There were two main purposes for this study. The first, was to address the need to verify findings of previous research on predictors of programming performance where either a limited number of studies had been performed, or research findings were inconsistent. The second was to compare the performance of these traditional test-based predictors, against predictors that did not require tests and were based upon aspects of a student's ordinary programming behavior.

2.1 Participants

The introductory programming module at our university was designed to teach Java to students of varying abilities. Students were supported by two weekly lectures and a lab session where they would practice solving programming problems using the BlueJ IDE. The sample of students used in this study consisted of volunteers from both the 2011/12 and 2012/13 cohorts. The structure of the course was similar for both years, with the only difference being the removal of the final exam in 2012/13. The teaching approach and learning materials were identical each year.

2.2 Instruments

Seven instruments were used to collect data from subjects: a questionnaire on the academic background of students and prior programming experience; attributions of success; Rosenberg's self-esteem scale; Kolb's learning style instrument; Gregorc style delineator; the motivated strategies for learning questionnaire (MSLQ); and a logging extension was added to the BlueJ IDE to collect data describing the programming behavior of students.

The background questionnaire was designed in-house, and gathered data on a student's: (1) gender, (2) GPA: high school, college, (3) lectures attended per week, (4) math and science background: courses taken and grades, (5) prior programming: for each language: years of experience, longest program written.

Measuring attributions of success in this study was based upon repeating the method used by [8], where students were asked to rank order four possible reasons for their success in the course. They were attribution to ability, task difficulty, luck, and effort.

Rosenberg's self-esteem scale (RSE) is perhaps the most widely used self-esteem measure in social science research. It consists of 10 questions and has been shown to in general to have a high reliability [15]. The RSE uses 4 point scales, ranging from strongly agree to strongly disagree. The only study to date [4] was replicated by using the reworded questions provided that modified the scale to relate self-esteem with a programming context.

Kolb's Learning Style Inventory (LSI) measures an individual's intrinsic learning style, or predisposition in any given learning situation. Kolb describes learning as a cycle of involvement, starting with concrete experiences and followed by a period of reflection, observation, and application of those experiences to solve problems. The LSI consists of a set of 12 sentences where individuals rank order four completions on a scale of 1 to 4. The LSI provides scores (range 12 to 48) for an individual's predisposition toward concrete experience, reflective observation, abstract conceptualization, and active experimentation.

The Gregorc style delineator describes an individual's learning style based on four dimensions: concrete random, concrete sequential, abstract random, and abstract sequential. The instrument consists of a set of 10 sentences where individuals rank order four completions on a scale of 1 to 4. The highest score among the four channels determines the dominant learning style.

The motivated strategies for learning questionnaire was co-designed by Pintrich [15] and is used to measure the motivations and learning strategies (cognitive, meta-cognitive, and resource management strategies) or students. It measures 17 different scales: 6 motivational and 9 learning strategies. The scales can be used together, but given their modular design, they can also be administered individually. The MSLQ uses a 7 point scale, ranging from 1 (not at all true of me), to 7 (very true of me).

A logging extension was used to gather data describing aspects of a student's ordinary programming behavior as they completed programming exercises using the BlueJ IDE. Each time a student compiled code on a university PC, the extension would log a snapshot of the code being compiled, along with the student's username, a timestamp, event type (compilation success or failure), and the error message reported with line number (if applicable). To explore aspects of a student's programming behavior and how it may relate to performance, we applied the data cleaning and processing procedure presented in [23] to each of our datasets. The procedure takes as an input directly logged compilation and invocation data, and produces a set of successive compilation pairings, for each file that a student attempted to compile during a session. These pairings describes how a student's programming behavior in terms of how their source code evolved between two consecutive compilations. For example, if a student compiled a file and encountered an error, in the next compilation of that file, were they able to resolve it? The procedure we selected has been shown to be more robust [23] than related approaches [12] which simply construct compilation pairings on a per-session basis, and instead constructs parings on per-session, per-file basis. The procedure also uses invocation information to refine estimates of the time a student spent between compilations – allowing their error resolve time for different types of error to be more accurately profiled.

In total 37 students (32 male) from the 2011/12 cohort and 45 students (42 male) from the 2012/13 cohort consented to us logging data describing their programming behavior over a period of 19 weeks. From the 2012/13 cohort 39 students (36 male) completed the six questionnaires. A one-way ANOVA showed no significant differences in the performance of the three samples of students on the reference criterion, $F(2, 118) = .18$, $p = .83$.

2.3 Predictor Variables

The relationships between 50 predictors and student performance in the introductory programming course at our university were examined. These predictors fall into 8 categories, including:

1. previous programming experience: has prior experience, number of languages previously studied, longest program written, years of experience.
2. previous academic experience: college grades: physics, chemistry, maths; university grades: discrete, calculus, GPA: college, high school.
3. attributions of success: scores for 4 scales outlined.
4. behavioral traits: lectures attended, hours part time job.
5. self esteem: overall score on Rosenberg.
6. learning styles: 8 scales taken from ILS and Gregorc.
7. learning strategies and motivations: 12 MSLQ scales.
8. programming behavior: 5 measures based on error frequency, 5 measures based on time, 2 measures based on an overall scoring of programming behavior.

2.4 Criterion Variable

To maintain an identical criterion variable for all students, we used overall coursework mark as the measure of programming ability in this study. This mark consisted of a weighting of a student's marks on a mid-term exam (25%), project (25%), a practical exam (40%), and weekly programming exercises (10%).

3. RESULTS

A priori analysis was carried out to verify that no significant differences existed between the mean overall scores of the class on the reference criterion, and those who agreed to participate in the studies conducted. Test assumptions of normality (Shapiro-Wilks test) and equality of variance (Levene test) were satisfied, and a set of *t*-tests showed no significant differences between the performance of those who participated in the: 2011/12 logged data sample ($t(78) = .28$, $p = .77$), 2012/13 logged data sample ($t(97) = 1.08$, $p = .29$), or the 2012/13 questionnaire sample ($t(91) = .56$, $p = .57$) and the remainder of their respective cohorts. In the remainder of this section, the findings on the relationships among each of the predictors examined and performance are presented.

3.1 Previous Programming Experience

All 39 students completed the background questionnaire section on prior programming experience. A *t*-test revealed significant differences in the performance those students who had prior programming experience prior to enrolling on the course ($n = 15$, $M = 71.76$, $SD = 10.47$) and those students who did not ($n = 24$, $M = 64.34$, $SD = 10.70$), ($t(37) = 2.12$, $p < .05$). These findings are consistent with previous research such as [25], but contradict research such as [2, 18]. Further analysis showed more interesting relations between prior experience and performance; however none of the following measures were significant, ($p > .05$). The number of languages that a student had previously studied weakly correlated with performance, $r = .24$, the longest program that a student had written prior to enrolment on the course also weakly correlated, $r = .15$, and surprisingly years of programming experience negatively correlated with performance, $r = -.20$. 5 students had prior Java experience, but no relation was found between the longest Java program they had previously written and performance, $r = .01$. In general, these results suggest that whilst prior programming experience may be useful to students, specific aspects such as years of experience, or the number of languages a student has studied, has little impact on performance.

3.2 Previous Academic Experience

To establish the relationship between previous academic experience in mathematics and science, the achievable grades for each subject were ranked, with the highest rank given to the highest possible grade, and the lowest rank given to the lowest possible grade. No significant correlations were found ($p < .05$) between either: grades in college physics ($n = 26$, $r = .31$), chemistry ($n = 15$, $r = .27$), math ($n = 28$, $r = .20$), university discrete math grade ($n = 15$, $r = .06$), or college GPA ($n = 35$, $r = .21$). But, marginally significant correlations were identified between performance and university calculus grade ($n = 26$, $r = .37$, $p = .06$), and high school GPA ($n = 38$, $r = .27$, $p = .10$). These findings are consistent with previous research that suggest generally academic background factors are weakly correlated with programming performance [2, 6], and that grades obtained in calculus courses are more strongly related to programming performance, than grades obtained in discrete courses [17]. But a total lack of correlation between discrete math and programming performance was contradictory to previous research [24].

3.3 Attributions of Success

All 39 students completed the background questionnaire section on their attributions of success. To date only 3 studies [8, 11, 18] have explored relationships between attributions and performance. Significant ($p < .05$), but weak, correlations were found between performance and attributions of success to task difficulty ($r = -.10$), and attributions to effort ($r = .07$). A moderate and marginally significant correlation was found for attribution of success to luck ($r = -.31$, $p = .05$) and a moderate, significant correlation was found for attribution of success to ability ($r = .40$, $p < .05$). The correlations reported by this study are consistent with, and within the range of correlations reported by the previous three studies on attributions to: task difficulty ($r = -.20$ to $.20$), and effort ($r = .07$ to $.16$). However much stronger relations for both attributions to ability ($r = .07$ to $.16$) and attributions to luck ($r = -.22$ to $.05$) were found. These conflicting results suggest that further research on how attributions of success relate to performance is required.

3.4 Behavioral Traits

All 39 students reported their lecture attendance. No relationship was found between lectures attended and performance, ($r = .02$, $p > .10$). 7 students reported the weekly hours that they worked in a part time job whilst studying. A strong negative correlation between the hours a student worked in a part time job and programming performance was found ($r = -.64$, $p < .01$). But this result should be interpreted with caution due to the small sample.

3.5 Self-Esteem

All 39 students completed Rosenberg's self-esteem scale. A weak, not significant, relation between score obtained on the instrument and programming performance was found ($n = 39$, $r = .13$, $p = .42$). Only one other study to date [4] used Rosenberg's instrument to examine the relationship between self-esteem and programming performance. However a moderate correlation between these two variables ($n = 54$, $r = .36$) was reported. The differing results between this study and prior research suggests that further research on the relations between programming self-esteem and programming performance would be beneficial.

3.6 Learning Styles

38 students completed both learning style instruments. Only 3 studies to date [7, 9, 10] have reported correlations between scores on the 4 dimensions of Kolb's ILS and performance. In this study no significant relations between learning style and performance were found for any of the 4 dimensions: concrete experience (CE) ($r = -.18$, $p = .29$), reflective observation (RO) ($r = -.07$, $p = .69$), abstract experimentation (AE) ($r = .14$, $p = .39$), abstract conceptualization (AC) ($r = .10$, $p = .53$). These correlations are consistent with, and within the range of the correlations reported by the previous three studies, (CE: $r = -.16$ to $-.23$; RO: $r = -.36$ to $.06$; AE: $r = .02$ to $.16$; AC: $r = .15$ to $.26$). These results suggest that in general that there is little to no relation between Kolb's ILS and the performance of students. Results for the Gregorc learning style were more encouraging. No significant correlation for the concrete/random dimension and performance was found ($r = -.14$, $p = .39$); But moderate and marginally significant correlations were found for each of remaining dimensions: abstract/random ($r = -.33$, $p = .05$), concrete/sequential ($r = .27$, $p = .10$) and abstract/sequential ($r = .29$, $p = .08$). Only 2 studies to date [13, 14] have explored the use of Gregorc's instrument as a predictor. Our findings are consistent with these studies.

Table 1. Pearson correlations (*r*) of this study compared to previous research MSLQ. (* $p < .10$, ** $p < .05$, *** $p < .01$)

MSLQ Dimension	This Study	[3]	[4]
Critical thinking	.28 *	.57	
Total metacognitive	.14	.54	
Resource strategy; Effort	.28 *	.62	
Resource strategy: Peer	-.06	.37	
Total resource strategy	.04	.56	
Task value	.06	.54	.44
MSLQ total	.22 *		.49
Intrinsic goal orientation	.33 *		.51
Total self-efficacy	.54 ***		.54

Compared to previous research, a similar moderate correlation for the concrete/sequential dimension ($n = 218$, $r = .35$) was reported by [13]. [14] found an identical moderate correlation ($n = 131$, $r = .30$) for abstract/sequential dimension, compared to this study. This suggests that the Gregorc learning style delineator may perform as a reasonable indicator of programming performance.

3.7 Learning Strategies and Motivations

All 39 students completed the MSLQ. Results are presented in Table 1. Only 2 studies to date [3, 4] have explored the relations between programming performance and scores on the various motivational and learning strategies scales on the MSLQ. Compared to prior research, an identical strong correlation for the self-efficacy for learning and performance dimension ($r = .54$, $p < .01$) was found. Marginally significant ($p < .10$) correlations were found for: intrinsic goal orientation ($r = .33$, $p = .04$), critical thinking ($r = .28$, $p = .08$), resource strategies: effort ($r = .29$, $p = .08$), and MSLQ total score ($r = .22$, $p = .09$). These findings confirm the research by [3, 4] who suggested that students who perform well in programming courses have high levels of intrinsic motivation and self-efficacy. However the findings of this study differed with previous researchers on a number of dimensions. No significant correlations were found between the total scores on the resource strategies scale ($r = .05$, $p = .76$), task value ($r = .06$, $p = .70$) scale, and the metacognitive strategies scale ($r = .15$, $p = .37$) was found to have a significantly lower correlation than previous researchers reported. These findings suggest that whilst certain dimensions of the learning strategies employed by students are related to their programming performance, further research is required to identify the dimensions that are the most significant.

3.8 Programming Behavior

10 metrics based upon the programming behavior of students were examined. Each metric was based upon the specific types of event pairings that students produced, measured as a percentage of their total number of pairings. Percentages were chosen as a means of standardizing the number of event pairings against all pairings each student produced, and as prior research has shown that metrics based upon event counts alone are poor indicators of performance [23]. 5 metrics were based upon the frequency of specific types of pairings a student produced. 5 metrics were based upon the percentage of lab time a student spent working on specific types of pairing. Results are shown in Table 2.

Unlike the traditional test-based predictors that we have examined throughout this section, an abundance of strong and significant relations were found between metrics of programming behavior and the performance of students. In terms of the percentage of specific types of pairings logged for each student, a strong significant correlation was found for the percentage of pairings where an error persisted for two successive compilations ($n = 82$, $r = -.51$, $p < .01$). Moderate correlations were also identified for the percentage of pairings where any errors existed in two successive compilations ($n = 82$, $r = -.46$, $p < .01$), and for the percentage of pairings where two successive compilations were successful ($n = 82$, $r = .38$, $p < .01$). Consistent with [12, 23], these results suggest weaker students are characterized by a high percentage of successive errors during lab sessions, whilst stronger students are characterized by having a high percentage of successive successful compilations.

Examining the relations between the time students spent on different types of event pairings, further significant correlations were found. A strong significant correlation was found for the percentage of lab time that students spent working on pairings where any errors existed in two successive compilations ($n = 82$, $r = -.50$, $p < .01$). An inverse relation was found for the percentage of lab time that students spent working on pairings where two successive compilations were successful ($n = 82$, $r = .39$, $p < .01$). A further moderate correlation was found for the percentage of lab time that students spent working on pairings where an error persisted for two successive compilations ($n = 82$, $r = -.42$, $p < .01$). These results suggest that not only are weaker students characterized by producing a high percentage of successive errors during lab sessions, but also, weaker students will generally spend a greater percentage of their lab time interacting with uncompilable code, than stronger students.

Table 2. Pearson correlations (*r*) between programming behaviors and performance (* $p < .10$, ** $p < .05$, *** $p < .01$)

Programming Behavior	2011/12	2012/13	Total
	$n = 37$	$n = 45$	$n = 82$
Based on Frequency of Events. Percentage of pairings:			
Error to Same Error	-.50 ***	-.54 ***	-.51 ***
Error to Different Error	-.32 *	-.43 ***	-.37 ***
Error to Any Error	-.48 ***	-.48 ***	-.46 ***
Error to Success	.18	.54 ***	.38 ***
Success to Success	.44 **	.37 **	.38 ***
Based on Time. Percentage of Lab Time Spent On:			
Error to Same Error	-.35 **	-.51 **	-.42 ***
Error to Different Error	-.44 ***	-.43 ***	-.41 ***
Error to Any Error	-.54 ***	-.51 ***	-.50 ***
Error to Success	.07	-.09	-.01
Success to Success	.41 **	.38 **	.39 ***
Overall Quantification Measures:			
Error Quotient [12]	-.42 **	-.47 ***	-.44 ***
Watwin Score [23]	-.60 ***	-.65 ***	-.60 ***

The use of programming behavior as a predictor of performance was further explored by applying two overall quantification algorithms to our datasets: Error Quotient (EQ) [12] and Watwin Score [23]. Both algorithms work by applying a scoring algorithm to the different types of compilation pairings that a student produces during a lab session. The major difference between the two algorithms is that Watwin [23] relatively penalizes a student based upon how their resolve time for different types of error, compares to the resolve times of their peers on the same error. The EQ moderately correlated with performance (n = 82, r = -.44, p < .01). However Watwin score showed a significantly stronger correlation (n = 82, r = -.60, p < .01). These findings suggest that aspects of a student's programming behavior are strongly related to their performance in programming courses and that hybrid algorithms may be one of the best data-driven predictors.

3.9 Regression Analysis

To investigate whether the various factors examined were predictive of performance in the module, three regression analyses were performed. As 5 of the students who completed the questionnaires did not provide consent for us to log their programming behavior, our sample size is reduced to 34 students.

The first model was designed to determine the predictive potential of the six written questionnaires. Consideration was given for all the traits examined in this study, with the exception of: previous programming experience, academic background (apart from GPA), hours worked in a part time job, due to a small number of students in each of these categories. Using a stepwise regression, a significant model was found with $F(3, 24) = 8.56$, $p < .01$, and an adjusted R-square of 45.6%. Significant values were found for MSLQ test anxiety (β = -.33, p = .04), MSLQ total metacognitive self regulation (β = .34, p = .03), and score on the Gregorc abstract/random dimension (β = -.40, p = .01).

The second model was designed to determine the predictive potential of the 12 programming behavior traits. Using a stepwise regression, a significant model was found with $F(2, 33) = 22.21$, $p < .01$, and an adjusted R-square of 56.3%. Significant values were found for Watwin score (β = -.56, p < .00) percentage of lab time spent working on error to success pairings (β = .41, p < .01).

A third model was designed to determine whether a hybrid of both traditional predictors and those based on programming behavior could explain more variance than the previous two models. All characteristics used to construct the previous two models were entered into the regression. A significant model was found with $F(3, 27) = 17.92$, $p < .01$, and an adjusted R-square of 60.6%. Significant values were Watwin score (β = -.77, p < .01), MSLQ total resource management strategies (β = .29, p < .01), and MSLQ total control of learning beliefs (β = .29, p < .01).

4. DISCUSSION

For almost 50 years, researchers have examined how prior academic experience, attributions of success, behavioral traits, self-esteem, learning styles, and learning strategies relate to the programming performance of students. But, with the exception of self-efficacy measured by the MSLQ (r = .54, p < .01) this study found no predictor within any of these traditional categories that strongly correlated with the performance of our students. Figure 1 shows the top 20 predictors found by this study. As can be seen, whilst 9 of the traditional test-based predictors are in the top 20, the strength of their correlations with programming performance are concentrated around the weak-moderate range. The remaining 29 traditional predictors outside the top 20 performed similarly.

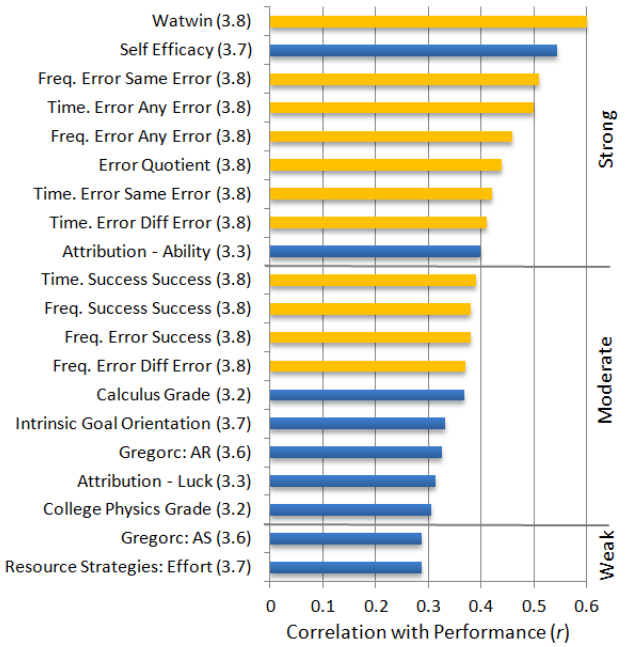

Figure 1. Bar chart showing the top 25 predictors of programming performance identified by this study. Correlations shown are absolute values and references to corresponding sections are shown in brackets. Programming behavior predictors are yellow, test-based predictors are blue.

In contrast, 11 of the 12 predictors based upon programming behavior were within the top 20, and were found to significantly relate to the programming performance of students. As can be seen from Figure 1, a total of 7 predictors based on programming behavior were found to strongly correlate with performance, and the remaining 4 moderately correlated. The implication of this research is that traditional test-based predictors are substantially less effective at reflecting the programming ability of students, and that data-driven approaches offer a more accurate method of prediction. In this study the results for test-based predictors were mostly inconsistent. The results for the programming behavior metrics (Table 2) were mostly consistent. It is worth stressing the further advantages of using predictors based on programming behavior. Traditional test-based approaches mainly examine traits that are static in nature and fail to reflect changes in the students learning progress over time. Although such approaches may have a chance of identifying weaker students, their one-shot, static nature, means that they cannot be dynamically used to support such students, e.g., by providing automatic interventions to assist weaker students when they are struggling. The programming behavior metrics however could be used in such circumstances, without the requiring any additional workload for either instructors or students.

Finally we acknowledge the limitations of this study. There are numerous difficulties associated with identifying predictors of programming performance. Our data was consistent with previous researchers in terms of the frequency and distribution of different types of error [12,16,23]. However conditions, such as the language taught or tools used to program vary considerably across different teaching contexts. Whilst this study has shown that several aspects of programming behavior can significantly correlate with a student's performance, further verification is required to determine the general applicability of these metrics across a variety of different teaching contexts and situations.

5. CONCLUSION AND FUTURE WORK

In this paper, 38 traditional test-based predictors of programming performance were reexamined, and compared to 12 dynamic predictors that were based upon analyzing aspects of a student's regular programming behavior. Whilst only one strong relation was identified between traditional predictors and performance, an abundance of strong and significant relations were found between aspects of programming behavior and performance. A model based upon two aspects of programming behavior could account for 56.3% of the variance in coursework marks, an improvement of approximately 25% when compared to a model based on traditional predictors alone. The results are encouraging, and the implication of this study is that predictors based upon aspects of programming behavior may be one of the strongest predictors of performance. Researchers should continue to explore their potential further, and work is essential to verify the performance and applicability of such predictors across a variety of teaching contexts. Future work will aim to develop techniques of applying the metrics within practical contexts, such as visualizations of learning progress [19], game-based tools [21,22], or tools to improve the compiler feedback provided to novice students [20].

6. REFERENCES

[1] Bennedsen, J. and Caspersen, M.E. 2007. Failure rates in introductory programming. *SIGCSE Bull*. 39(2), 32-36.

[2] Bergin, S. and Reilly, R. 2005. Programming: factors that influence success. *SIGCSE Bull.* 37(1), 411-415.

[3] Bergin, S., Reilly, R. and Traynor, D. 2005. Examining the role of self-regulated learning on introductory programming performance. In *Proc. of the 1st Int. Workshop on Computing Education Research (ICER)*, 81-86.

[4] Bergin, S. and Reilly, R. 2005. The influence of motivation and comfort-level on learning to program. In *Proc. of the 17th PPIG Workshop*, 293-304.

[5] Biamonte, A.J. 1964. Predicting success in programmer training. In *Proc. of the 2nd SIGCPR Conference on Computer Personnel Research*, 9-12.

[6] Byrne, P. and Lyons, G. 2001. The effect of student attributes on success in programming. *SIGCSE Bull*. 33(3), 49-52.

[7] Campbell, V. and Johnstone, M. 2010. The Significance of Learning Style with Respect to Achievement in First Year Programming Students. In *Proc. of the 21st Australian Software Engineering Conference (ASWEC)*, 165-170.

[8] Cantwell-Wilson, B. and Shrock, S. 2001. Contributing to success in an introductory computer science course: a study of twelve factors. *SIGCSE Bull.* 33(1), 184-188.

[9] Chamillard A.T. and Karolick, D. 1999. Using learning style data in an introductory computer science course. *SIGCSE Bull.* 31(1), 291-295.

[10] Corman, L.S. 1986. Cognitive style, personality type, and learning ability as factors in predicting the success of the beginning programming student. *SIGCSE Bull*. 18(4), 80-89.

[11] Henry, J.W., Martinko, M.J., and Pierce, M.A. 1994. Attributional style as a predictor of success in a first computer science course. *Computers in Human Behavior,* 9(4), 341-352.

[12] Jadud, M.C. 2006. Methods and tools for exploring novice compilation behaviour. *Proc. of the 2nd Int. Workshop on Computing Education Research (ICER)*, 73-84.

[13] Lau, W.W., and Yuen, A.H. 2009. Exploring the effects of gender and learning styles on computer programming performance implications for programming pedagogy. *British Journal of Educational Technology*, 40(4), 696-712.

[14] Lau, W.W., and Yuen, A.H. 2011. Modelling programming performance: Beyond the influence of learner characteristics. *Computers & Education*, 57(1), 1202-1213.

[15] Pintrich, P. R., Smith, D. A., García, T., & McKeachie, W. J. 1993. Reliability and predictive validity of the Motivated Strategies for Learning Questionnaire (MSLQ). *Educational and psychological measurement*, 53(3), 801-813.

[16] Rodrigo, M.M.T., Baker, R.S., Jadud, M.C., Amarra, A.C.M., Dy, T., Espejo-Lahoz, M.B.V, Lim, S.A.L., Pascua, S.A.M.S., Sugay, J.O., and Tabanao, E.S. 2009. Affective and behavioral predictors of novice programmer achievement. *SIGCSE Bull*. 41(3), 156-160.

[17] Stein, M.V. 2002. Mathematical preparation as a basis for success in CS-II. *Computing Sciences in Colleges*, *17*(4), 28-38.

[18] Ventura, P. 2005. Identifying predictors of success for an objects-first CS1. *Computer Science Education,* 15(3), 223-243.

[19] Watson, C., Li, F.W.B., and Lau, R.W.H. 2010. A pedagogical interface for authoring adaptive e-learning courses. In *Proc. of the 2nd Int. Workshop on Multimedia Technologies for Distance Learning,* 13-18.

[20] Watson, C., Li, F.W.B., and Godwin, J.L. 2012. BlueFix: using crowd sourced feedback to support programming students in error diagnosis and repair. In *Proc. of 11th Int. Conference on Advances in Web-Based Learning*, 228-239.

[21] Li, F.W.B., and Watson, C. 2011. Game-based concept visualization for learning programming. In *Proc. of the 3rd Int. Workshop on Multimedia Technologies for Distance Learning (MTDL)*, 37-42.

[22] Watson, C., Li, F.W.B., and Lau, R.W.H. 2011. Learning programming languages through corrective feedback and concept visualisation. In *Proc. of 10th Int. Conference on Advances in Web-Based Learning (ICWL)*, 11-20.

[23] Watson, C., Li, F.W.B., and Godwin, J.L. 2013. Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. In *Proc. of the 13th IEEE International Conference on Advanced Learning Technologies (ICALT)*, 319-323.

[24] White, G., and Sivitanides, M. 2003. An empirical investigation of the relationship between success in mathematics and visual programming courses. *Information Systems Education*, 14(4), 409-416.

[25] Wiedenbeck, S. 2005. Factors affecting the success of non-majors in learning to program. In *Proc. of the 1st Int. Workshop on Computing Education Research (ICER)*, 13-24.