An Experiment with Algebraic Specifications of Software Components

Hong Zhu
Department of Computing and Electronics
Oxford Brookes University
Oxford OX33 1HX, UK
E-mail: hzhu@brookes.ac.uk

Bo Yu
Department of Computer Science & Technology
National University of Defense Technology
Changsha, China
Email: hnaxdsjk@163.com

Abstract – A long lasting myth of formal methods is that they are difficult to learn and expensive to apply. This paper reports a controlled experiment to test if this myth is true or false in the context of writing algebraic formal specifications. The experiment shows that writing algebraic specifications for software components can be learnt by ordinary university students. It does not heavily depend on mathematical knowledge and skills. Moreover, it is not time consuming to write algebraic specifications for software components.

Keywords–Formal methods; Algebraic specification; Software components; Learnability; Cost and expense.

I. INTRODUCTION

Algebraic specification was first proposed in 1970s as an approach to the implementation independent definition of abstract data types [1]. In the past three decades, significant progress in the research on algebraic specifications has been made towards a mature formal method [2]. In particular, the approach has advanced in the following areas.

- *Theoretical foundation*. The recent development of hidden algebra [3] and co-algebra [4] approaches enable the formal development of state-based, object-oriented and component based software and it is applicable to nondeterministic distributed concurrent systems.
- *Language and tool support*. A number of formal specification languages and their supporting tools have been developed. For example, OBJ3 [5], CafeOBJ [6] CASL [7] and CoCasl [8] are among the most well-known languages.
- *Applications*. In addition to formal reasoning and proving properties, algebraic specifications can also be used as the basis for software testing and reverse engineering. When applied to software testing, a high degree of test automation can be achieved by automatically generating test cases and checking the correctness of program output. It can be applied to data types in procedural languages [9, 10], classes in object-oriented systems [11 , 12 , 13 , 14 , 15] and software components [16,17]. In reverse engineering, algebraic axioms at a high level of abstraction with good readability can be automatically derived from program source code [18, 19].

However, the algebraic approach has not been widely adopted by the industry. In general, it is widely perceived that formal methods are difficult to learn and expensive to apply [20]. Such myths regarding formal methods were first identified more than 25 years ago [21 , 22]. They still abound despite a significant number of success stories of industrial applications of formal methods [23 , 24], as Bowen and Hinchey observed [25]. In the past 20 years, based on case studies and success stories, formal methodists have argued in vain that these myths are not true. They have failed to convince the rest of the world, especially the software engineering community and IT industry. Therefore, we believe it is necessary to employ software engineering research methods, e.g. controlled experiments, to found out whether such myths are true or false.

This paper reports such an experiment with focus on the following myths of formal methods. They are tested in the context of writing algebraic specifications.

– *Myth 1: Dependence on mathematical skills*. Writing algebraic formal specifications is a job for the well-trained mathematicians. It requires good mathematical skills.

– *Myth 2: Time consuming and expensive*. Writing algebraic specifications is a complicated and time consuming task.

– *Myth 3: Unlearnability*. Writing algebraic specifications is hard to learn. It needs training and practices.

The paper is organized as follows. Section II describes the design of the experiment. Section III analyses the data of the experiment. Section IV draws conclusions from the findings of the experiment, points out their practical implications and limitations, and discusses the potential threats to their validity and the direction for future work.

1550-6002/10 $26.00 © 2010 IEEE
DOI 10.1109/QSIC.2010.54

II. DESIGN OF THE EXPERIMENT

This section presents the design of the experiment.

A. Subjects

Our experiment was conducted with year-three university students of computer science and technology, who have not learned formal methods before the experiment. Therefore, they are suitable for testing how hard it is to learn writing algebraic specifications.

These students had taken three mathematics courses and three programming courses in the previous semesters before the experiment. The mathematics courses were Advanced University Mathematics (Part A and B) and Discrete Mathematics. The programming related courses were C++ Programming, Java Programming and Data Structures. These courses were compulsory courses for all students of the BSc. Degree course of Computer Science and Technology. They were taught following the standard syllabuses set by the Ministry of Education of China.

Thirty five students participated in the experiment. Those students who failed on any of the mathematics and programming courses were eliminated from the experiment. Figure 1 shows the distributions of student capabilities in terms of their average examination scores in mathematics courses, programming courses and the average scores of all courses. It is worth noting that in the marking scheme, a score in the range 90%-100% is grade A, in 80-89% is grade B+, 70-79% is B, 60-69% is grade C, and it is grade F (fail) if the score is below 60%. The numbers of students in each grade are also given in the figure. As shown in the figure, the scores evenly spread from lower 60s to 90s. TABLE I gives the statistical data of student capabilities in terms of the average and standard deviation of mathematics courses, programming courses and all courses. It is worth noting that the cohort of students who participated in the experiment is not highly capable because their average scores are of grade B. They represent the ordinary university students who major in computer science and technology.

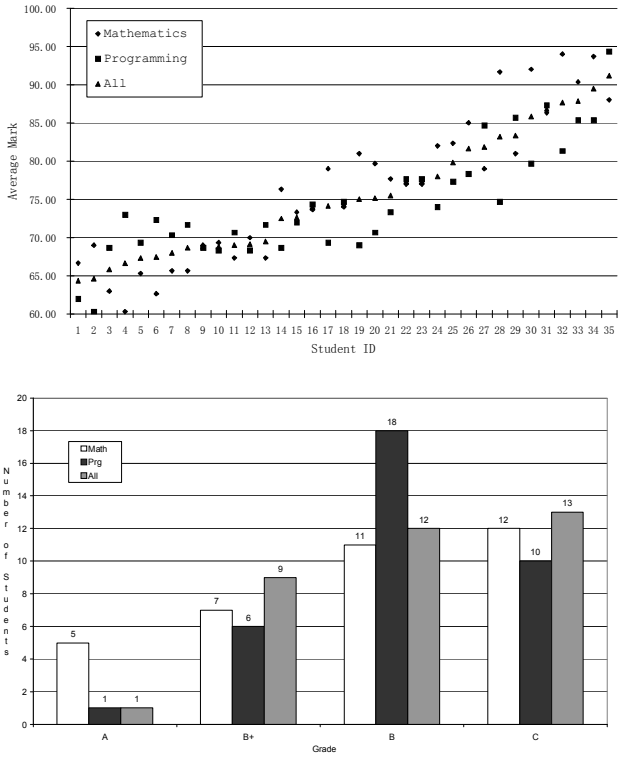


Figure 1. Distribution of Students' Capabilities

TABLE I. STATISTIC DATA OF STUDENT CAPABILITY

	Mathematics Courses	Programming Courses	All Courses
Average	76.44	74.59	75.51
Std Deviation	9.67	7.34	7.83

B. Process of the experiment

The experiment was conducted as a part of a course on software engineering. Four consecutive lessons within three weeks were devoted to teaching and testing students' attainments of algebraic formal specifications. Students' performances in the experiment were counted as 10% of the assessment of the course. Thus, the students have taken the experiments seriously. The process of the experiment consists of the following steps.

(1) In the lesson preceding the experiment, the students were given an introduction to the general concepts of formal methods.

(2) At the first lesson of the experiment, the students were first introduced to algebraic specification in general and the CASOCC specification language in particular. CASOCC is a formal specification language based on the theory of hidden algebra and designed for supporting automated testing of software components [16, 17]. An example formal specification, which is a specification of the stack data structure, was illustrated in the lesson. In the second half of the lesson, the students were assigned a class test to write independently an algebraic specification of the first class test problem using the CASOCC language. Please see TABLE II for the class test problem.

(3) At the second lesson, a sample answer to the class test question of the previous lesson was discussed at the class, and then a second class test question was assigned to the students to attempt at the class independently.

(4) At the third lesson, the sample answer to the second class test was presented and then the students were assigned to the third class test.

(5) At the fourth lesson, the sample answer to the previous class test is presented and the final class test was assigned to the students.

For each class test, students' work was assessed according to the quality of their answers. Because the quality of an algebraic specification depends on two factors, i.e. correctness and completeness, the following marking scheme is used to assess students' work.

- *Correctness of the answer*: 50%. This is assessed according to the correctness of the signature and axioms in the student's work. For each axiom with a minor syntax error that can be detected by CASCAT tool, the mark is reduced by 20%, while axioms with semantic errors were given no marks.
- *Completeness of the axiom system*: 50%. This is assessed according to the coverage of the operators by the axioms. The coverage of each operator was given an equal number of marks.

In the experiment, the students were given no time limit to complete the class tests. But, they were asked to hand in their work as soon as possible. The length of time that each student took to complete a class test was recorded.

It is worth noting that before they started to work on a class test question, the students were briefed about the function and the interface of the component. Therefore, the length of time taken to write the algebraic specification excludes the length of time to understand the components.

C. Sample problems

As stated above, four different software components were selected for the students to specify as the class test problems. The first three components were of similar complexity. They were typical examples of software components and came from the tutorial of J2EE [27]. By the end of the third lesson, it became clear that the majority of students had attained the knowledge and skills taught in the classes. Thus, the fourth class test problem was selected to be significantly harder than the first three in order to test whether the students were capable of applying the knowledge and skill to more complicated problems. This class test problem was selected from classic algebraic specification textbooks. It has also been studied by researchers in the investigation of deriving algebraic specification from Java source code [19]. More information about the class test questions is given in TABLE II.

TABLE II. SOFTWARE COMPONENTS USED IN THE CLASS TESTS

Test	Component	Source	#Operators
1	Complex numbers	Java Textbook [26]	13
2	Bank account	J2EE Tutorial [27]	12
3	Gangster database	J2EE Tutorial [27]	16
4	Linked Lists	Textbooks, also [19]	8

III. RESULTS AND MAIN FINDINGS

This section analyses the main results of the experiment against the myths regarding algebraic specifications.

A. How easy is it to learn to write algebraic specifications?

The students participated in the experiment as a learning experience. We observed that there is a clear performance improvement from the first class test to the third as shown in the statistical data about the average scores for each class test given in Figure 2. In particular, the average scores increased monotonically from class test 1 to 3. The average score of the final class test is slightly lower than class test 3. This is because the question is significantly harder than the questions for the first three class tests.

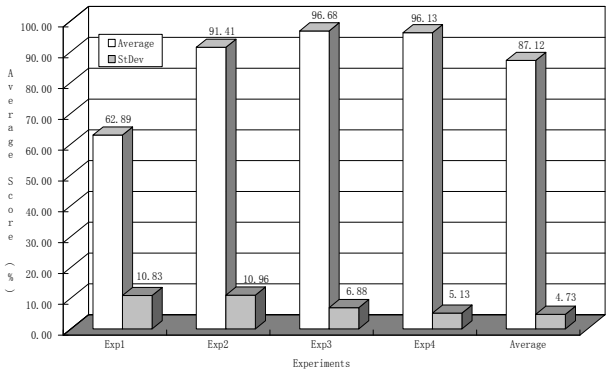


Figure 2. Changes of Scores over Tests

Further evidence of the learnability of algebraic specification is in the distribution of student scores in class tests as shown in Figure 3.

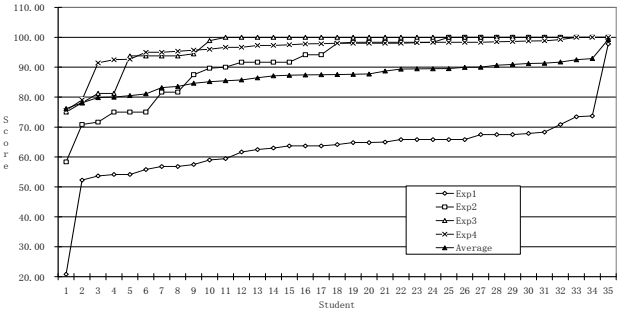


Figure 3. Distributions of Scores

In Figure 3, the distribution of the scores of each class test is shown in one line. It is clear that the line of class test 1 is below the line of class test 2, which in turn is below the line of class test 3. In other words, the distributions of scores improve monotonically from test 1 to 3. The distribution of scores of the final class test is not completely above the line of class test 3. However, they are very close to each other although the question of test 4 is much harder.

TABLE III gives the number of students in different grades in various class tests. Column 6 of the table also

gives the number of students in different grades according to their average marks of all 4 class tests. It shows that the majority of students (33 out of 35, i.e. 92.3%) reached grade A in the final class test while all students passed the test. In fact, the average score of the final test is 96.13% with a standard deviation of 5.13%. This is a significant improvement in the average score in comparison with 62.89% of the first class test.

TABLE III NUMBERS OF STUDENTS IN VARIOUS GRADES

Grade	Test 1	Test 2	Test 3	Final Test	Avg of All Tests
A	1	25	31	33	8
B+	0	4	2	0	23
B	3	5	2	2	4
C	20	0	0	0	0
F	11	1	0	0	0

Therefore, we can conclude that the students easily attained the knowledge and skill of writing algebraic specification in just a few lessons.

B. How expensive is it to write an algebraic specification for a software component?

Is writing algebraic specification an expensive task? To answer this question, we investigated how long it took a student to write an algebraic specification for a software component.

In the experiment, the cohort of students not only performed very well in terms of the scores, but also in terms of the lengths of time that they took to complete the test questions. As shown in Figure 4, the average lengths of time that students took to complete the class test questions decreased monotonically from class test 1 to 3, in which the problems are of the similar complexity. The average length of time of class test 4 is only slightly longer than the third although the question is significantly harder.

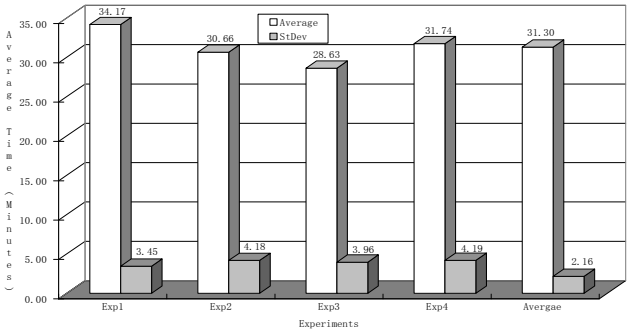


Figure 4. Changes of Average Lengths of Time over Tests

Figure 5 gives the details about the distribution of the lengths of time that students took to complete class tests. As shown in Figure 5, the line for the lengths of time of class test 1 is above the line for class test 2, which is in turn above the line for class test 3. In other words, the distributions of the lengths of time also improve monotonically from class test 1 to 3. In the final class test, the average length of time was 31.74 minutes with a standard deviation of 4.19 minutes.

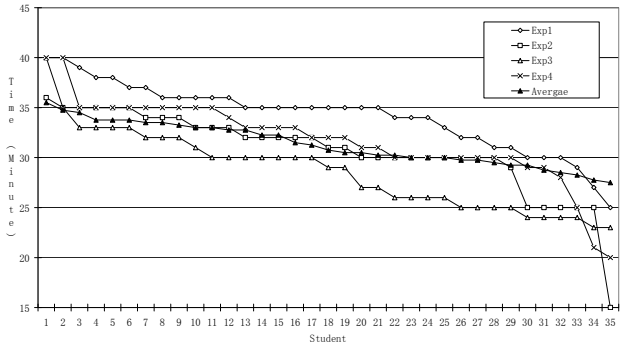


Figure 5. Distributions of Lengths of Time

The experiment data show that on average a student took about half an hour to complete the writing of an algebraic specification for a typical software component.

C. Does writing algebraic specification need good mathematical skills?

To analyze to what extent writing algebraic specifications depends on mathematical knowledge and skills, we calculated the correlation coefficients between students' performances in class tests and their performances in mathematics courses and compared the correlation coefficients with those between programming courses and the average scores of all courses. The results are shown in TABLE IV.

TABLE IV CORRELATION COEFFICIENTS

	Length of Time	StDev of Lengths of Time	Score	StDev of Scores
Mathematics Courses	-0.44	0.34	0.10	-0.10
Programming Courses	-0.52	0.41	0.46	-0.44
All Courses	-0.52	0.40	0.28	-0.27

As shown in TABLE IV, the correlation coefficients between the average scores of mathematics courses and average scores of class tests on algebraic specification is only 0.10. When the absolute value of a correlation coefficient is close to 1, there is a strong correlation between the random variables. Otherwise, when the absolute value of the coefficient is close to 0, there is no correlation between them. Therefore, since the coefficient is very close to 0, the experiment data show almost no correlation between the average score of mathematics courses and the performance in writing algebraic specifications. They are almost independent.

The correlation coefficient between the average score of mathematics courses and the average lengths of time for the

students to complete specifications is much greater, which is -0.44. This means there is some dependence between the average score of mathematics and the average length of time for the student to complete the specifications. The higher score is, the less time used. However, the absolute value of the coefficient is less than 0.5, thus the dependence between these two random variables is not conclusive.

When these correlation coefficients are compared with the correlation coefficients of programming courses and all courses, it becomes clear that the link between students' average scores of programming courses and their performances in class tests is stronger than the links between students' average scores of mathematics courses and their performances in class tests.

The following statistical analysis of the data provides further evidence to support the statement on the link between programming capabilities to algebraic specification. We divided the students into the following four groups and calculated their scores in class tests.

- *Group 1 (P>M: More capable of programming than mathematics):* A student falls into this group if his/her average score of programming courses is higher than his/her average score of mathematics courses by at least 5 marks. There are 6 students in this group.
- *Group 2 (P<M: More capable in mathematics than programming):* A student is in this group if his/her average score of mathematics courses is higher than his/her average score of programming courses by at least 5 marks. There are 13 students in the experiment who belong to this group.
- *Group 3 (P~M High: Equally capable of programming and mathematics):* A student is regarded as of equal capability of programming and mathematics if the difference between his/her average scores of programming courses and mathematics courses is within 5 marks. These students are further divided into two groups from the median according to their average scores of all courses. Group 3 consists of those have higher average scores of all courses. There are 8 students in this group. They all have an average score above 70.
- *Group 4 (P~M Low: Equally incapable of programming and mathematics):* This group consists of students of equal capability of programming and mathematics, but their overall average scores are lower than those in group 3. There are also 8 students in this group. They all have an average score below 70.

The statistical data for each group is given in TABLE V. For Group 1 (P>M), although the average score of all courses is lower than that of Group 2 (P<M), the average score of class tests is observably higher that group 2. Moreover, for Group 2 (M>P), the average score of class tests is the lowest among all four groups, even lower than Group 4, whose capabilities of both programming and mathematics are the poorest.

TABLE V. STATISTICS OF THE PERFORMANCES OF GROUPS

	Avg. Math	Avg. Prog	Avg. All	Avg. Score	Avg. Final Score	#Stds
P>M	69.78	77.44	73.61	90.82	96.91	6
P<M	84.31	74.92	79.62	85.44	95.97	13
P~M (High)	77.50	77.83	77.67	87.23	96.46	8
P~M (Low)	67.58	68.67	68.13	85.49	95.50	8

Therefore, the students' performances in class tests are more closely related to their programming capability than to mathematics knowledge and skills. If a student is more capable of programming, he/she performs better in the class tests on algebraic specification.

It is worth noting that the above statistical analysis is on the average score of all class tests. Therefore, the link between students' performances in class tests and programming capability should be interpreted as their capability to learn algebraic specification rather than their final attainment. In Section A, we have shown that there is little difference in the final attainments for the cohort of students. It is also worth noting that even for the relationship between programming capability and class test performance, the link is not strong since the absolute values of the correlation coefficients are in the range from 0.41 to 0.52.

D. Is writing algebraic formal specifications a job only for the most capable?

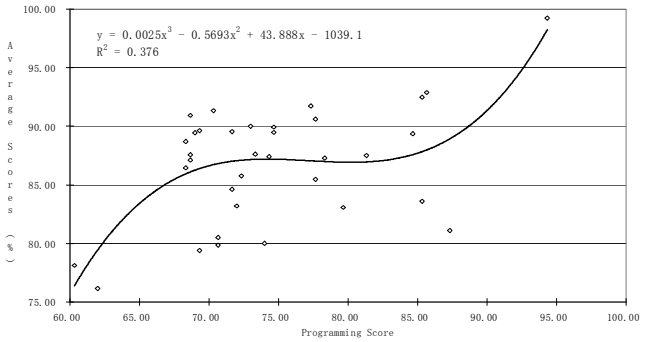
Having shown that writing algebraic specifications is less dependent on mathematical knowledge and skills than on programming capability, a question still remains to be answered. That is, is writing algebraic formal specifications a job only for the most capable?

To answer this question we look into the details about how students perform in class tests in the context of their programming capabilities. Figure 7 shows the distributions of the averages of class test scores and the average length of times taken to complete the tests over all 35 students when the students are indexed by their average scores of programming courses. The curves in the figures are the 3rd order polynomial fittings. The polynomial functions of the fitting curves for average scores and lengths of time are given below in equation (1) and (2), respectively.

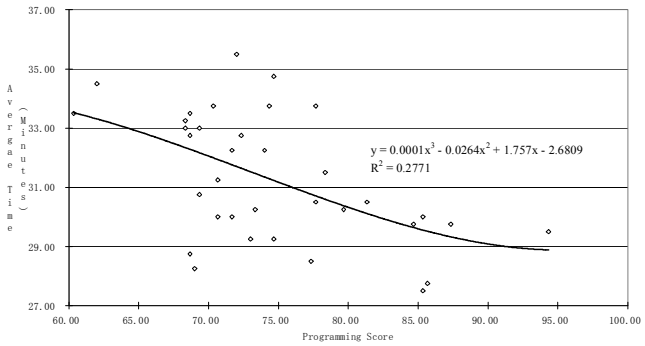
$$y = 0.0025x^3 - 0.5693x^2 + 43.888x - 1039.1 \quad (1)$$

$$y = 0.0001x^3 - 0.0264x^2 + 1.757x - 2.6809 \quad (2)$$

Equation (1) shows that the relationship between

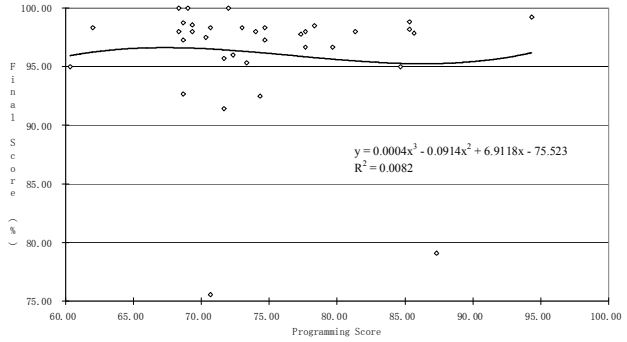


(a) Class Test Scores vs Programming Capability

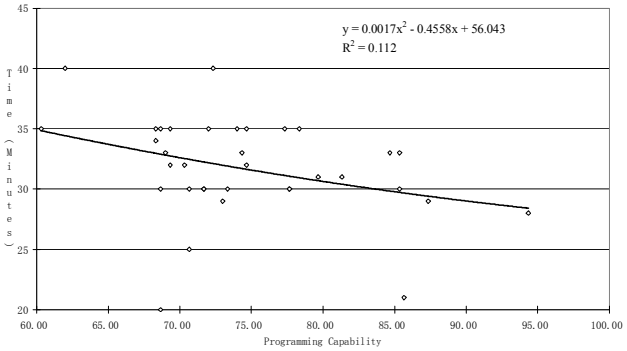


(b) Lengths of Time vs Programming Capabilities

Figure 6. Dependence on Prg Capability during Training



(a) Final class test score vs programming capability



(b) Lengths of time in final test vs programming capability

Figure 7. Dependence on Prg Capability after Training

average score and programming capability is essentially a second order polynomial function since the coefficient of the 3rd order term is negligible. Equation (2) shows that the average length of time depends on programming capability more or less linearly because the coefficients of the 2nd and 3rd order are negligible.

From the above statistical analysis one might draw a conclusion that writing algebraic specification must be the job of the most capable programmers. However, there is a potential bias in the above analysis because the average scores and average lengths of time contain the results of the first and second class tests. Thus, they do not reflect the situation after the students completed their training. Therefore, we should look into the distributions of the scores and lengths of time in the final class test, which reflect their performance when finishing the learning process.

Figure 6 (a) and (b) show the distributions of students' performance in the final class test in terms of the scores and lengths of time, respectively, where the students are indexed by their average scores of programming courses. The trend line shown in Figure 6(a) is mostly linear and almost constant. Its 2nd order polynomial fitting function is give below.

$$y = -0.0002x^2 - 0.0148x + 98.405 \qquad (3)$$

The trend line shown in Figure 6(b) is also mostly linear, but not constant. Its polynomial fitting function is:

$$y = 0.0017x^2 - 0.4558x + 56.043 \qquad (4)$$

Therefore, after taking three lessons and class tests, the students are capable of writing algebraic specifications of almost equal quality, but the most capable ones took slightly less time. From this point of view, writing algebraic specifications can be a job for any well trained software developer rather than just for the few most capable ones.

IV. CONCLUSION

We now discuss what conclusions can be drawn from the experiment, their implications and limitations, and the potential threats to the validity of such conclusions. Further work will also be identified in the discussion.

A. General conclusions

From the experiment data, we can draw the following general conclusions.

Statement 1 (*Learnability*)

Writing algebraic specification is learnable for ordinary software developers.

This statement is supported by our experiment with university students of computer science and technology. The cohort of students who participated in the experiment had capabilities range from grade C to grade A in studying their university courses. In the experiment, all of them

passed the class tests and 32 out of 35 students got grade A in the final class test. The conclusion generalizes the finding of the experiment on the ground that if ordinary students can learn writing algebraic specifications, then everybody of similar or better background than these students can, too. In particular, software developers graduating with a computer science degree should have a background at least the same as the students if not any better.

Statement 2 (*Independence of mathematical skills*)

The knowledge and skill of programming is more important than mathematics when writing algebraic specifications of software components.

This statement is supported by the statistical analysis of the correlation between the students' performance in the class tests and mathematics and programming subjects. The correlation coefficients between the average score of programming courses and class test scores is greater than the correlation coefficients between the average score of mathematics courses and class test scores. Moreover, statistical data also show that students who are more capable of programming than mathematics outperformed those students who are better in mathematics than programming. This means that students who are good at programming learn algebraic specification easier and quicker.

The conclusion generalizes the findings of the experiment on the grounds that the software components used in the experiment are typical in software development practices. This may not be true for developing software components that heavily depend on mathematical knowledge.

Statement 3 *(Cost efficiency)*

Writing algebraic specifications can be as cost efficient as programming in high level programming languages.

The evidence that supports this statement is that in the experiment, students took about half an hour on average to write a good specification of a software component. The standard deviation of the lengths of time taken to write an algebraic specification was about 4 minutes. The length of time taken to write an algebraic specification for a component was comparable to coding in high level programming languages if not shorter. The conclusion generalizes the findings of the experiment on the grounds that with the accumulation of experience, a software developer can write algebraic specifications more and more efficiently. Thus, if a student is efficient, then an experienced developer can be at least equally efficient.

Statement 4 (*Equality in performance*)

Writing algebraic specification can be a skill of every well trained software developer. Although their efficiency in writing algebraic specifications depends on their capabilities, there should be no significant difference in quality.

This statement is supported by the statistical data of student performance in the final class test. The data show little difference in the scores of their work and the lengths of time they took to complete the class test. The relationship between length of time and the student's capability is linear, while the relationship between the score and capability is almost constant. In other words, once a software developer is trained to be able to write algebraic specifications, he/she should be able to produce equally good quality output and take almost the same amount of time as his/her colleagues. Thus, writing algebraic specification can be a job of any ordinary software developer. This conclusion generalizes the findings of the experiment on the grounds that the sample subjects who participated in the experiment are representative.

B. Implications and Limitations

The existence of a formal specification is the precondition of formal software development. Much work has been reported in the literature on the development of automated or semi-automated software tools to support formal methods, such as model checkers, theorem provers, interpreters of executable formal specifications, automated testing tools, etc. All these tools require a formal specification as an input. Although to some extend formal specifications can be derived semi-automatically from semi-formal graphical models (see, for example, [28]), formal specifications still need to be written manually by software developers. Our experiment demonstrates that writing formal algebraic specifications is practical in the sense that it is learnable for ordinary software developers. It is also cost efficient and the output can be of high quality.

We recognized two particular practical implications. First, the way that we teach students in the experiment is effective. The students learned writing algebraic specifications via trial-and-error and improved their attainments quickly. This can be a good approach to teaching formal methods.

Second, in the experiment, we taught students to write algebraic specifications of software components with a behavioural approach. The experiment shows that it is a practical approach. A particular usage of such algebraic specifications is to test software components automatically [16, 17].

It is worth noting that there are also a number of limitations of the conclusions stated in the previous subsection.

First, the conclusions are only applicable to writing algebraic formal specifications. They do not necessarily

imply that writing formal specifications in other formalisms has the same properties. Further research is necessary to investigate whether the same claim can be made to other formalisms such as Z, Petri-nets, process algebras like CSP, CCS and π-calculus, and labelled transition systems in general. A notable advantage of algebraic specification is that the syntax and semantics of the specification language are simple and easy to understand. They use few mathematics symbols.

Second, the fact that writing formal specification is learnable does not necessarily imply that reasoning about software properties, proving software correctness using formal specifications and deriving program code from specifications are equally learnable. These activities may well require much deeper understanding of the theories of formal methods and the semantics of formal specification languages. They may also rely on skills of using software tools that support formal methods.

Finally, it is also worth noting that all the students who participated in the experiments had passed their courses on mathematics. Our experiment result does not mean that students who had not learned mathematics courses at all would perform equally well. Further experiments should be conducted to find out whether mathematics courses have an effect.

C. Potential Threats to the Validity

In [29], the threats to the validity of experiments and the conclusions drawn from the findings were classified into four types. We now discuss the applicability of these threats and how we deal with them in our experiments. We also identify future work in which the threats can be better dealt with.

1) Conclusion Validity. Conclusion validity is concerned with the correctness of the conclusions drawn from the experiment data. The threats to the conclusion validity may come from the following sources.

- *Low statistical power*: We used 35 subjects in the experiment, which is a reasonably large number. To improve the statistical power, we can use more students and repeat the experiment in the future.
- *Reliability of measures*: Two measurements of student's performance in the experiment are used: the assessment scores and the length of time to complete the test. The time taken by the students to complete the tests is measured to the accuracy of minutes. We believe it is unnecessary to be measured to the unit of seconds so that random errors are reduced. The marking of student class tests is conducted systematically following a pre-set marking scheme as in all university examinations. Although the method suffers from subjective judgment of students' work, potential systematic bias were carefully dealt with and the students were given fair marks by strictly following a predefined marking scheme. This is the only practical method that is affordable to us for an experiment of such number of students and the number of class tests. If more resource is available, a more objective measurement of the quality of student works or simply more markers could be used. This can be improved in future work.
- *Random irrelevancies in experimental setting*: We have identified the following main factors that may affect the outcome of the experiment: the subjects' background and experiences in studying and using formal methods, the students' knowledge and skill in mathematics and programming, the students' capability of learning in general, the sample components used in class tests. Data on all of these factors have been collected and taken into consideration in the statistical analysis. Other minor factors may have affect on the outcome, such as the classroom environment in which the tests took place, the date and time of the class tests, etc. Although classroom environment differs from office environment in which real software development takes place, there should be no significant impact on the students' performance because the classroom is an environment that students are familiar with.
- *Random heterogeneity of subjects*: The students who took part in the experiment were from an ordinary university with an even distribution of capability. None of them had been exposed to formal methods at all before the experiment. The subjects were therefore highly homogenous and representative. This threat is not present in the experiment.

2) Internal validity. Internal validity is concerned with the possible internal properties of the subjects that may affect the validity of the observed phenomena in the experiment. The applicable threats of this type are as follows.

- *Statistical regression*: This threat does not exist in our experiment because the conclusions drawn from the experiment do not depend on the manner in which the subjects were grouped. All the students completed their class tests individually and independently.
- *Ambiguity about direction of causal influence*: This threat does not exist in the experiment because the causal relationships have been clearly stated without ambiguity.

3) Construct Validity. Construct validity is concerned with the validity of generalizing the result of the experiment to the concept or the theory beyond the experiment. The potential threats are discussed as follows.

- *Inadequate preoperational explication of constructs*:

This threat does not exist because the experiment was carefully designed and clearly defined before the operation.

- *Mono-operation bias*: This threat was dealt with by using a fairly large number of subjects. A future work to improve in dealing with this threat is to repeat the experiment with students from different universities and of different year.
- *Mono-method bias*: This threat exists because the experiment used only one method of teaching and testing student's learning performance and attainment. However, the approach has a very important advantage. That is, the test scores are highly comparable and thus statistical analysis techniques are applicable. For future work, different methods could be applied to inter-check the validity of the conclusions.

4) External Validity. External validity is concerned with the external conditions in which the experiment is conducted. These conditions may restrict the generalization of the conclusion from the experiment condition to the real world situation. The factors that may threat the validity of this experiment are the sample problems used in the experiment as the class test problems and the selection of subjects, which are university students. Three of the sample problems were selected from tutorial of J2EE technology. The tutorial was written with professional software developers as the target readers. The final sample problem was selected from traditional textbook on algebraic specification and it is more difficult than the other three. To further improve the validity of the experiment, sample problems could be selected from real components from the industry. Moreover, professional software engineers could be used as the subjects.

In summary, the potential threats to the validity of the experiments have been dealt with properly in the design and conduct of the experiment. Where potential threats exist, caution has been paid to draw conservative conclusions with explicit statements of their limitations. Further research has also been identified according to the remaining threats.

ACKNOWLEDGEMENT

The authors would like to thank the students who participated in the experiment. They are from the Department of Computer Science of The Swan College of The Central South University of Forestry and Technology in China. The authors also thank our colleague Dr. Rachel Harrison of Oxford Brookes University for her invaluable comments on a draft of the paper.

REFERENCES

[1] J. A. Goguen, et al. "Initial Algebra Semantics and Continuous Algebras", Journal of ACM, vol. 24, no.1, 1997, pp68 – 95.

[2] D. Sannella and A. Tarlecki, "Algebraic methods for specification and formal development of programs", ACM Computing Surveys, vol.31, No.3es, Article 10, Sept. 1999.

[3] J. Goguen, and G. Malcolm, "A Hidden Agenda", Theoretical Computer Science, vol. 245, no. 1, 2000, pp55-101.

[4] J. M. Rutten, "Universal coalgebra: a theory of systems," Theoretical Computer Science, vol. 249, no. 1, 2000, pp. 3–80.

[5] J. Goguen, et al., Introducing OBJ3. Technical Report SRI-CSL-88-9, SRI International, USA, 1988.

[6] K. Futatsugi and A. Nakagawa, "An Overview of CAFE Specification Environment -- An Algebraic Approach for Creating, Verifying, and Maintaining Formal Specifications over Networks", Proc. of ICFEM'97, Nov, 1997.

[7] M. Bidoit, D. Sannella, and A. Tarlecki, "Architectural Specifications in CASL" Formal Aspects of Computing, vol. 13, 2002, pp252-273.

[8] T. Mossakowski, L. Schroder, M. Roggenbach and H. Reichel, "Algebraic-coalgebraic specification in CoCasl," J. Log. Algebr. Program., vol. 67, no. 1-2, pp. 146–197, 2006.

[9] J. Gonnon, P.McMullin, and R. Hamlet, "Data-Abstraction Implementation, Specification and Testing", ACM TOPLAS vol. 3, no. 3, 1981, pp211-223.

[10] G. Bernot, M. C. Gaudel and B. Marre, "Software Testing based on Formal Specifications: a Theory and a Tool", Software Engineering Journal, Nov. 1991, pp387- 405.

[11] K. Doong and P. Frankl, "The ASTOOT Approach to Testing Object-Oriented Programs", ACM TOSEM vol.3, no.2, 1994, pp101-130.

[12] M. Hughes and D. Stotts, "Daistish: Systematic Algebraic Testing for OO Programs in the Presence of Side-Effects", Proc. of ISSTA'96, 1996, pp53-61.

[13] H. Y. Chen, T. H. Tse, and T. Y. Chen, "TACCLE: a Methodology for Object-Oriented Software Testing at the Class and Cluster Levels", ACM TSEM, vol. 10, no.1 2001, pp56-109.

[14] H. Y. Chen, et al. "In black and White: an Integrated Approach to Class-Level Testing of Object-Oriented Programs", ACM TSEM, vol. 7, no. 3, 1998, pp250-295.

[15] H. Zhu, "A Note on Test Oracles and Semantics of Algebraic Specifications", Proc. of QSIC'03, 2003, pp91-99.

[16] L. Kong, H. Zhu, and B. Zhou, "Automated Testing EJB Components Based on Algebraic Specifications", Proc. of COMPSAC'07, Vol. 2, pp717-722.

[17] B. Yu, L. Kong, Y.Zhang, and H. Zhu, "Testing Java Components Based on Algebraic Specifications", Proc. of ICST'08, April 2008, pp190-199.

[18] J. Henkel, C. Reichenbach, and A. Diwan, "Developing and debugging algebraic specifications for Java classes", ACM Trans. Softw. Eng. Methodol, vol. 17, no. 3, 2008.

[19] J. Henkel, C. Reichenbach and A. Diwan, "Discovering Documentation for Java Container Classes", IEEE Trans. Software Eng., vol. 33, no. 8, 2007, pp526-543.

[20] M. Hinchey, M. Jackson, P. Cousot, B. Cook, J. Bowen, and T. Margaria, "Software engineering and formal methods", C. ACM, vol. 51, no. 9, Sept. 2008, pp54-59. DOI= http://doi.acm.org/10.1145/1378727.1378742

[21] A. Hall, "Seven Myths of Formal Methods", IEEE Softw, vol.7, no.5, Sept. 1990, pp11-19.

[22] J. P. Bowen and M. G. Hinchey, " Seven More Myths of Formal Methods", IEEE Softw. vol. 12, no.4, Jul. 1995, pp34-41.
[23] M. G. Hinchey, and J. P. Bowen, (eds.). Applications of Formal Methods, Prentice Hall and Englewood Cliffs, 1995.
[24] M. G. Hinchey, and J. P. Bowen (eds.), Industrial-Strength Formal Methods in Practice, Springer-Verlag FACIT Series, London, 1999.
[25] J. P. Bowen, and M. G. Hinchey, "Ten commandments revisited: a ten-year perspective on the industrial application of formal methods", In Proc. of the 10th international Workshop on Formal Methods For industrial Critical Systems (FMICS '05), Lisbon, Portugal, Sept. 05 - 06, 2005, pp 8-16.
[26] C. Zhou, "Programming Numerical Algorithms in Java", Publishing House of Electronics Industry, China. (In Chinese), 2007.
[27] S. Bodoff, et al.. The J2EE Tutorial, 2nd Edt., Pearson 2004.
[28] L. Jin and H. Zhu, "Automatic generation of formal specification from requirements definition", Proc. of IEEE 1st International Conference on Formal Engineering Methods (ICFEM'97), Hiroshima, Japan, Nov. 1997, pp243-251.
[29] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B.Regnell, and A. Wesslen, Experimentation in software engineering: an introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.