The Impact of Using Formal Methods Tools in Undergraduate Computing Courses

Toufik Taibi
College of Information Technology, United Arab Emirates University,
P.O. Box 17555, Al Ain, United Arab Emirates, toufikt@uaeu.ac.ae

Abstract

The last decade has shown an increasing interest in integrating formal methods into the software development lifecycle in order to strengthen software reliability. A prerequisite to the success of such integration is the incorporation of formal methods tools in the computing undergraduate curriculum. This paper covers an experiment conducted on undergraduate software engineering student, while taking a formal methods course. The experiment involved comparing the errors found in specifications using the pen and pencil approach and using two tools- Z Type Checker (ZTC) and a Z Animation system (ZANS).The experiment results not only show that the tool-support approach is far superior in reducing the errors found in specifications but also that it has improved the student learning experience and enthusiasm towards formal methods.

Keywords*: Formal methods, Z formal notation, ZTC, ZANS.*

1. Introduction

Software is becoming an integral part of our lives. As such, software reliability is becoming a paramount in the requirements of software projects. Formal methods are well-equipped to provide the needed reliability. However, their widespread application is hindered by the lack of its expertise among software practitioners. Since most of these practitioners are the result of a computing education system, it is important to equip undergraduate students with theoretical as well as practical usage of formal methods.

The success of traditional engineering disciplines was mainly due to the high integration of theory and practice. Computing curricula do not seem to address the integration of theory (mathematics) with practice through the usage of tools.

This paper covers an experiment on the impact of tool support on the learning experience of students. This involves solving the same problems using the pen and pencil approach and the tool support approach. The problems given to students were to write specification of systems using the Z formal specification language [10]. The tool used were the Z Type Checker (ZTC) [5] and Z Animation System (ZANS) [6].

The experiments results have shown a reduction of the number of errors found using tools compared with the P&P approach. Moreover, the usage if the ZANS tool improved tremendously the checking the semantic correctness of operations.

The rest of the paper is organized as follows. Section 2 provides a brief overview of Z, ZTC and ZANS. Section 3 describes the experiment conducted on student learning experience with and without the usage of tools and summarizes its outcomes. Section 4 covers related work, while section 5 concludes the paper.

2. Z, ZTC and ZANS

The Z notation is a model-oriented formal specification language developed by the Programming Research Group at Oxford University Computing Laboratory in the early 80s. Since then, Z has known successful widespread usage [1], the most notable being the specification of IBM's Customer Information Control System (CICS) Application Programming Interface (API) [4]. The above project has shown that the code generated from Z specifications and designs has 2.5 times fewer problems that the code that was not specified in Z.

Z is a strongly typed specification language. ZTC is a type checker for Z, which determines if there are syntax and typing errors in Z specifications. ZTC accepts two forms of input: LaTeX [8] and ZSL [5], an ASCII version of Z. ZTC can perform translations between LaTeX and ZSL. In our experiment LaTeX input is used.

Below is an example LaTeX input file and its corresponding Z specification.

\begin{document}

\begin{zed}
[NAME, DATE]
\end{zed}
\begin{schema}{BirthdayBook}
*known: \power NAME *
birthday: NAME \pfun DATE
\where
known = \dom birthday
\end{schema}
\end{document}

[NAME, DATE]

BirthdayBook

known: Π NAME
birthday: Name § DATE

known= dom birthday

The lexical elements of a LaTeX input can be categorized as follows:

- *LaTeX commands*, which begin with a backslash (\), such as *\begin* and *\end*.
- *Keywords*, such as *schema*, *zed*.
- *Identifiers*, such as DataDictionary and name?
- *Integers*, such as 0 and 65535.
- *Symbols*, which consists of one or more non-alphanumeric characters, such as *::*, ==, and *::=*.

Each specification must be enclosed in *\begin{document}…\end{document}*. Anything outside is ignored by ZTC. A Z specification consists of formal and informal text. ZTC will type check formal text and ignore the informal text. Formal text must be enclosed in one of the following formal environments:

- *\begin{axdef}…\end{axdef}*, used to define axiom boxes.
- *\begin{gendef}…\end{gendef}*, used to define generic boxes.
- *\begin{schema}…\end{schema}*, used to define schema boxes.
- *\begin{syntax}…\end{syntax}*, used to define free types. A *syntax* environment contains a sequence of syntax rules separated by the *\also* command.
- *\begin{zed}…\end{zed}* use to define other paragraphs in Z. These include given sets, schema definition (horizontal format), equivalence definition and predicates. Short free type definitions can also be included in the zed environment. The paragraphs in a *zed* environment must be separated by the \also command.

Informal comments or remarks inside formal environments can be introduced as follows: *\comm{informal text}* or *\remark {informal text}*. ZTC ignores the arguments of these two commands. Sometimes, it may be needed to ignore some formal text without deleting them. To do so the environments *\begin{comment}…\end{comment}* and *\begin{nocheck}…\end{nocheck}* can be used.

Separators (*;*, *\also*, \\ and *\linebreak*) are used to separate between declarations or predicates in the *axiom*, *generic* and *schema* boxes. Omission of separators between declarations or predicates will cause syntax and/or typing errors.

A line continuing command is a line breaking command followed by a *TAB* command, which is one of the following *\t0* (indents the least amount of space) … *\t9* (indents the most amount of space).

ZTC allows to break a long specification into several input files and then includes them into a master file using either of the following commands *input{filename}* or *\include{filename}*. To invoke ZTC, type *ztc* at the command prompt followed by a *filename*.

ZANS is an animation tool for Z specifications. The version of ZANS used supports type checking of Z specifications, expansion of schema expressions, evaluation of expressions and predicates; and execution of operation schemas. The input to ZANS can be written in LaTeX or ZSL. ZANS supports the Z syntax defined in [10].

To invoke ZANS, type *zans* on the command line. This will enter the *interpretation cycle* (*zans>*), which gets. ZANS to be waiting for a command.

A ZANS command consists of three parts separated by one or more spaces: *command-name [option] [arguments]*. The *option* and *arguments* are optional The option part must begin with a hyphen *(-)*. Most of the commands are *single-line commands*, which means that when the return key is hit, it signals the end of the command and ZANS starts to interpret the command. There are also a number of *multi-line commands*. They are designed to allow lengthy arguments to the commands, such as expressions or paragraphs in a specification. For a *multi-line command* a single return key will not terminate the command, instead a continuation prompt *cont>* will appear. A *multi-line command* is terminated with two consecutive *returns*.

ZANS has two modes of operation: the *initial mode* and the *animation mode*. The *initial* mode is the mode

at the start of ZANS. The *animation* mode is the mode in which specifications can be animated. The two different modes are indicated by two different prompts (*zans*> and *anim*>). Table 1 lists all ZANS commands (in alphabetical order)[6]. The column usage reflects the annotation of the commands, which are defined as follows [6]:

- S: single-line command; M: Multi-line command.
- I: available in the initial mode; A: available in the animation mode.
- → I: transition to the initial mode; → A: transition to the animation mode.

Table 1, ZANS commands

Command	Usage	Interpretation
analyze *filename*	<S,IA>	Analyze the entire specification. The operations generated are saved in the file named *filename*.
animate	<S,I→A>	Start animation.
assign variable:= expression	<M,IA>	Assign the value of the expression to the variable.
clear	<S,A→I>	Clear the current specification.
eval [-e] expression	<M,IA>	Evaluate the expression. -e: eager evaluation.
execute [-at] *schemaname*	<S,A>	Execute the operation schema named *schemaname*. -a: try all branches, -t: non-committal execution.
exit	<S,IA>	Exit ZANS.
expand [-dn] *schemaexp*	<M,IA>	Expand the schema expression. –d: convert to disjunctive normal form, -n: normalized.
expfile [-dn] *filename*	<S,IA>	Expend the entire specification and save the results in the file named *filename*. Options similar to those of command expand.
help	<S,IA>	List all commands of ZANS.
list	<S,IA>	List all the schema names in the currently loaded specification.
load *infile*	<S,IA>	Load and type check the specification file named *infile*.
para *paragraph*	<M,IA>	Enter and type check a paragraph.
pragma *pragmaname [args…]*	<S,IA>	Set pragmas.
pred *predicate*	<M,IA>	Evaluate the predicate.
script *scriptfilename*	<S,IA>	Set the name of the script file to *scriptfilename*.
show [-ov] *schemaname*	<S,IA>	Show the schema named *schemanane* in its original, unexpanded form. –o: shows the operations generated from the schema, -v: show the current value of the schema components.
source *scriptfilename*	<S,IA>	Run the script file named *scriptfilename*.
stop	<S,A>	Stop animation
style [-tlb]	<S,IA>	Set the output style. –t:LaTeX style, -l: ZSL text style, -b: ZSL box style (default).
verbose *digit*	<S,IA>	Set verbosity level: 0-9. 0 the least verbose, 9 the most verbose. Initial value: 5.

3. Experiments and Results

The formal methods course is taken by 9 software engineering students at the college of IT, UAEU. Students were given three specifications as exercises. The first covers a library management system, the second an examination management system, while the third a phone directory system.

The experiment was set in such a way that students are asked to solve the three exercises using the P&P approach then use the tools ZTC and ZANS to type check and animate their specifications respectively.

Table 2 summarizes the findings of the experiments which compared the errors found with P&P approach with those found with the tools (labeled T). Figure 1, plots the histogram of the total errors counts of students using the P&P versus the tool approach.

Table 2, Errors Counts with P&P and with Tool

	Exercise 1		Exercise 2		Exercise 3		Total	
	P&P	T	P&P	T	P&P	T	P&P	T
1	11	9	10	8	8	5	29	22
2	10	7	8	5	7	6	25	18
3	8	6	7	4	4	3	19	13
4	11	8	9	6	7	5	27	19
5	6	3	5	4	4	3	15	10
6	7	5	6	3	5	2	18	10
7	10	8	9	7	7	5	26	20
8	12	7	10	6	8	5	30	18
9	9	5	8	5	6	2	23	12

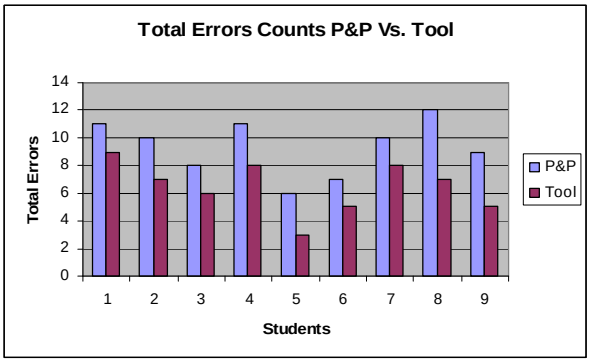

Figure 1, Total Errors Count P&P Vs. Tool

The above results confirm indeed that the usage of tools led to the generation of fewer errors than the P&P approach. Overall, error count in developing Z

specification dropped by 30% compared to the P&P approach. Moreover, the overall results obtained from this experiment suggest the following outcomes:

- Students' errors (using both P&P and Tool) decreased from exercise to another. This is a logical consequence since students gain more experience in Z with more practice.
- The errors count with using the tool was always less than the one using the P&P approach in all exercises. The tools really helped students strengthen their theoretical understanding of the Z specification language.

The above findings are quantitative in nature. As for the qualitative outcomes of the experiment they can be summarized as follows:

- Students' interest and motivation to learning Z has increased when the usage of tools was introduced.
- Using the tools has increased the cooperative learning of students.

4. Related Work

A number of research projects [7] have clearly showed the advantages of introducing formal methods into undergraduate curricula. Also, a number of initiatives have been announced and implemented [3] aiming at integrating the formal methods into the software engineering curricula.
It is only through the inclusion of formal methods that we can introduce rigorousness to the software development process. However, the study done in [2] found that there are only few formal methods course offered. Moreover, there is a lack of appropriate software tools that can assist students in mastering the theory and practical usage of formal methods.
The developed formal methods tools were only targeted for the industry and few open-source tools were available for academia [9].

5. Conclusion

This paper presented an experiment aimed at assessing the impact of using formal methods tools in a formal methods undergraduate course.
Students were given three different exercises and were asked to write Z specifications for them using both the P&P approach and the tool support approach.
The experiment results show that overall errors found in using the tool support approach were 30% less than those found in using the P&P approach.
These initial findings of the experiment are very promising to us as educators. Indeed it confirms what we have stated in the introduction that computing education of the mathematical based course need to be coupled with tool support that strengthen the theoretical knowledge learnt during the lecture.

References

[1] Craigen D., Gerhart S. and Ralson T., An international Survey of Industrial Applications of Formal Methods, Volume I and II, National Institute of Stardands and Technology, GCR/93/626, March 1993.
[2] Dean C.N., Boute R.T. (Eds.). Teaching Formal Methods, CoLogNET/FME Symposium, TFM 2004, Ghent, Belgium, Lecture Notes in Computer Science, Springer, 2004.
[3] Dean C.N. and Hincey M.G. (Eds.). Teaching and Learning Formal Methods Academic Press 1996,
[4] Houston I., and King, S. "CICS Project: Experiences and Results From the Use of Z in IBM", Proc. VDM'91 – Formal Software Development Methods, LNCS No. 552, pp. 588-596, 1991.
[5] Jia X. ZTC: A Type Checker for Z Notation, User's Guide, October 2003.
[6] Jia X. A Tutorial of ZANS- A Z Animation System, July 1998.
[7] Sobel, A. Final Results of Incorporating an Operational Formal Method into a Software Engineering Curriculum. In Proceeding of the IEEE Frontiers in Education, November 1999.
[8] Lamport L., LaTeX: A Document Preparation System, 2nd edition, Addison-Wesley, 1994.
[9] Skevoulis S., Markov V., Integrating Formal Methods Tools into Undergraduate Computer Science Curriculum, 36th ASEE/IEEE Frontiers in Education Conference, 2006.
[10] Spivey J.M., The Z Notation, A Reference Manual, Second Edition, Prentice Hall International, 1992.