VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences
Computer Science

Studying the Language of Programming

Robert Biddle and James Noble

Technical Report CS-TR-02/5
March 2002

School of Mathematical and Computing Sciences
Victoria University
PO Box 600, Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

Studying the Language of Programming

Robert Biddle and James Noble

Technical Report CS-TR-02/5
March 2002

Abstract

In this paper we address the study of programming as the use of programming language, and outline how such study might be undertaken. We believe that actual programming practice needs to be studied more and in new ways. We do not suggest venerating any existing practice, but rather exposing it for further study. We believe this approach should be desirable to support in industry practice, and provide a sound basis for academic study and research

Publishing Information

This paper was presented at the Feyerabend Workshop at the European Conference on object-Oriented Programming, Malaga, Spain, 2002.

Author Information

The authors members of the academic staff in the School of Mathematical and Computing Sciences and the School of Information Management at Victoria University of Wellington.

Studying the Language of Programming

Robert Biddle, James Noble
School of Mathematical and Computing Sciences
School of Information Management
Victoria University of Wellington
New Zealand

April 9, 2002

Die Welt ist alles, was der Fall ist.
(The world is all that is the case.)
— Wittgenstein

1 Programming Language

What is language? In the study of formal languages as introduced to most computer science students, and still regarded as an essential part of the core of the computing curriculum, the term *language* is defined as *a set of strings.* This is an interesting moment for thoughtful students, who may consider how this definition reflects the ordinary use of the word with which they have grown up.

The definition is soon largely forgotten anyway because the study of formal languages within computer science is in fact concerned mostly with grammars for describing and automata for processing strings. Somewhere along the way, the term language usually comes to mean not some set of strings that actually occurs, but rather a possible set of strings which can easily be described or processed. But we can easily forget that the grammars and automata address a superset of the actual language: the actual set of strings that exist in practice.

The study of formal languages in computer science stems from the work initiated by Chomsky in linguistics that explores the cognitive basis of language, looking in particular for relationships between language structure and cognitive models[2]. Interest in the structure of language has a long tradition, and this tradition is often prescriptive, placing rules before practice. This has affected the teaching of language and the meaning of words and pronunciation. Alongside this academic tradition, however, is another widespread pragmatic tradition. In learning how to communicate, whether in one's native language, or in learning another language, the primary issue is simply what is done, and any explanatory framework is regarded as secondary and of limited usefulness.

The role of actual practice in academic study of language is reasonably recent, making its notable impact on the world beginning with the pioneering work of lexicographer James Murray on the New (later Oxford) English Dictionary at the end of the 19th and the beginning of the 20th centuries [7]. Unlike the few prior and most of the many later dictionaries, the OED is based on practice as shown in published writing. This accomplishes a descriptive, rather than prescriptive, approach to language, while still making a contribution both to practical knowledge and academic study of the language.

About the same time, linguist Ferdinand de Saussure addressed the nature of language by distinguishing different aspects of language. In particular, he defined a language as *a system of signs.* Signs carry meaning in communication because the participants understand the structures of the signs that make up the messages exchanged between them [3]. This is the basis for the study of semiotics. An important aspect of this approach is that language is seen as a social artifact, and is essentially a system of social conventions, whether widely appreciated or not.

In computer science, programming languages are typically regarded as formal languages used to formally specify computer behaviour. Indeed, some people regard the term language as inappropriate for what they regard as mere notations. We, however, regard programming languages also systems of signs, and claim they, like natural languages, carry meaning in communication because the participants understand the structures of the signs that make up the messages exchanged between them. In the case of programming languages, the participants include the writers and any readers of the programs, and also people involved in the program and programming language implementation.

Programming languages are somewhat constrained, of course, in ways to allow implementation of programs. We do not believe these constraints are severe, however. To begin with, these very constraints are conventions established by us to allow us to go about our work which does involve setting priorities and going about things in certain ways. To finish with, humans are easily capable of, and practiced at, using constrained communication systems to accomplish other ends. From rigid poetic forms to contract bridge bidding systems, we work with what we have, to do what we want.

We believe that the time has come for study of the language of programming.

Quotations will tell the full measure of meaning, if you have enough of them. — Murray

2 Learning from Practice

The empirical study of programming is not new. In fact there have been empirical studies of various programming topics for many years, and there are regular conferences and coordinated work on the topic[6]. These studies, however, typically concentrate on the activity of programmers, rather than programming itself, and have a focus on very specific issues. For these issues, controlled conditions may be possible, and accurate data may be obtainable.

We believe that programming language practice needs to be studied with a linguistic approach. This must include examining actual language practice in the large. In a way, this would reflect successful practice in requirements gathering through study of use cases of existing practice, or with user interface design based on task analysis of existing practice. That is to say, it would not be intended to venerate existing practice, or to cement it for eternity. But it would be intended to identify it, document it, and open it for explanation, discussion, and potentially change.

2.1 Reviews

In software engineering practice, there are various approaches to review, such as walkthroughs and inspections[11]. These are often suggested as ways to share ownership, as well as to allow commentary and critique on practice. Despite this highlighting, and even despite institutional support, walkthroughs have not fulfilled the initial expectations. In industrial settings support can even be too strong. In some settings, we know that the date of the review was approached with fear, because it was a field day for disinterested senior staff to pick apart junior work. In other settings, we know that the date was approached with shopping, because the business presentation was everything and the right suit sent the right message. Both these kinds of treatment severely limited the good of a review.

Even so, the review was never suggested as a an academic study. However, we believe that, with a focus on presentation of design with rationale, this would have been very suitable. Especially in a diverse and eclectic milieu, with a mix of processes systems, and organisations, we believe there is an important role for for reviews in general. We also think that as well as being attractive business practice, they also provide a valuable exposure allowing academic study, and also the useful prospect for creating links between business process and academic consideration.

2.2 Case Studies

Another approach is to address programming though case studies. A case study addresses the same design topics as a review, but presented forward as a playing out of an initial case, rather than backward as an exposition of a system. These are especially suitable for study and documentation and especially useful when recorded for further study and commentary. In particular, these lead to a common literature of cases, and so assist general discussion not only about the case itself, but as a basis of comparison and common set of reference points.

2.3 In-situ Studies

A further more general approach is possible through in-situ studies. Such studies would not only look at a case as it developed, but study the processes of development among the contextual community. Such studies background an individual case, but foreground the processes and considerations of all the people involved, and can also see relationships between cases that may be involved at the same time. This kind of study can therefore highlight the process and "peopleware" issues, but also highlight how design issues were detected and were dealt with from the human or business process perspective. This tends to expose some design issues that might otherwise be concealed, and also allow the knowledge to be indexed in such a way that it can be easily brought to mind. This is one way of capturing, presenting, passing on, and raising for discussion various issues that can figure significantly in development as tacit knowledge.

2.4 Corpus Analysis

At the other extreme, another approach is to assemble a corpus of knowledge about actual programming. This addresses programming from the artifacts produced, and so can focus on the program code, the design documents, requirement documents, or even development logs, bug reports, maintenance reports, or time sheets. An important consideration is that the corpus should be coherent and comparable, so that aggregation of the artifacts in the corpus is reasonable. Exactly how the corpus is analysed or presented depends on the kind of artifacts. The main idea has to be to detect meaningful practice, and then present this for consideration. For example, our recent work in this regard is to study visual languages that harness two dimensional layout, and then present this aggregated information two dimensionally[9].

2.5 Semiotics

As a human activity, the use of programming language involves much use of signs, both overt and subtle, both well recognised and often ignored. Semiotics is the study of signs, and we believe that it has a useful place in the study of programming. The applicability of semiotics to computing has already been recognised, especially by Anderson[1]. Our recent work has shown to relevance of semiotics to understanding of design patterns[10, 8], and dicuss the work that has influenced us. Much of the work to date has been in demonstrating the explanatory power of semiotics in showing the existence of signs and relations between them. We believe that with greater familiarly with the signs of programming, semiotics can also play a role in detecting hidden meaning and tacit knowledge, and create a foundation for more explicit recognition and use of signs in programming.

2.6 Patterns

The great recent contribution of the study of the language of programming has been the identification, documentation such as begun with the *Design Patterns* book [5]. The patterns "movement" takes useful practice, codifies it and presents it for study. This has the effect of highlighting and circulating good practice, and also simply of exposing these highly regarded practices for further research.

As well as the main achievement, the patterns movement has also shown wise procedural practice. One example is the pride about "aggressive disregard for originality"[4], which ensures that it is actual practice that is the subject of study. Another example is the patterns workshop procedure, which is non-threatening and encourages contribution, and then fosters further consideration and development in a supportive way. These are the kind of considerations that were often lacking in review processes.

Despite this success, there are aspects to the patterns work that we believe run counter to the idea of learning from practice.

One issue is the concern sometimes voiced that documented patterns merely address weaknesses in some programming systems that could not arise in others. Our view is that learning from practice must be based in language communities: and that it does not make sense to reject knowledge gained in a community simply because other communities began from a base that did not require such knowledge. We do believe it makes sense for scholarly work to make links between programming practice in one community with programming language support in another. However, in documenting the use of language, we must be careful and respectful of the communities. It may make sense to change practice, to develop new programming language and new communities, but we believe it is an important academic principle that these activities be separated from the study of programming as it is.

Another issue is the desire sometimes voiced for large over-arching theories connecting practice. Where practices are connected in some large way in the communities of usage, we believe it is important that this be identified and studied. Where the linking comes from academic theory outside the practice, however, great caution is needed in the procedures of study. We believe it is a reasonable academic occupation to postulate large theories, but we do not believe it is necessary to require their existence. Most especially we believe that practice should never be seen as merely reflecting such large theories, because this runs counter to the study of language as it is used.

3 Envoi

In this paper we have addressed the study of programming as the use of programming language. We believe that actual programming practice needs to be studied more and in new ways. We do not suggest venerating any existing practice, but rather exposing it for further study. We believe this approach should be desirable to support in industry practice, and provide a sound basis for academic study and research. The topic we propose for dicussion is: how might a linguistic approach to the study of programming change the way we think?

> *Die Eule der Minerva beginnt erst mit der einbrechenden Dämmerung ihren Flug.*
> *(The owl of Minerva spreads its wings only with the falling of the dusk.)*
> *— Hegel*

References

[1] Peter Bøgh Andersen. A semiotic apporach to programming. In Peter Bøgh Andersen, Berit Holmqvist, and Jens F. Jensen, editors, *The Computer As Medium*, Learning in doing: Social, cognitive and computational perspectives, chapter 1. Cambridge University Press, 1993.

[2] Noam Chomsky. *Language and Responsibility.* The Harvester Press, 1979.

[3] Ferdinand de Saussure. *Cours de linguistique générale.* Charles Bally et Albert Sechehaye, 1916.

[4] Brian Foote. Hybrid vigor and footprints in the snow. *Pattern Languages of Program Design 3*, 1998.

[5] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns.* Addison-Wesley, 1994.

[6] Wayne D. Gray and Deborah A. Boehm-Davis, editors. *Empirical Studies of Programmers: Sixth Workshop.* ACM Press, 1996.

[7] K.M. E. Murray. *Caught in the Web of Words: James Murray and the Oxford English Dictionary.* Yale University Press, 1995.

[8] James Noble and Robert Biddle. Patterns as signs. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP 2002).* Springer, 2002.

[9] James Noble and Robert Biddle. Program visualisation for visual programs. In *Proceedings of the Australasian Uer Interface Conference.* Australian Computer Society, Conferences in Research and Practice in Information Technology, 2002.

[10] James Noble, Robert Biddle, and Ewan Tempero. Metaphor and metonomy in object-oriented design patterns. In *Proceedings of the Australasian Computer Science Conference.* Australian Computer Society, Conferences in Research and Practice in Information Technology, 2002.

[11] Gerald M. Weinberg and Daniel P. Freedman. *Handbook of Walkthroughs, Inspections, and Technical Reviews.* Dorset House Publishing, third edition, 1990.