I_mean_something_to_somebody (part 1 of 2)

Experiment performed at the 2003 ACCU Conference

First published in CVu vol. 15 no. 6

Derek M. Jones
derek@knosof.co.uk

1 Introduction

Software developers are constantly exhorted to use *meaningful* identifier name. Unless they are chosen at random all identifiers are likely to be have some meaning, at least to the person who created them. The implicit assumptions in the exhortation is that the names are meaningful to subsequent readers of the source code and also that all readers of the source containing them agree on what that meaning is.

Surprisingly there have been no studies investigating whether developers agree on the meaning assigned to uses of particular identifier names. Although there have been studies that have investigated related issues. These studies include, the meanings assigned to (human language) words, and words/phrases invented by people to describe something.

This article reports on an experimental study, performed during the 2003 ACCU conference, that attempted to measure one particular aspect of developer identifier meaning assignment behavior. The study investigated the extent to which belief in the applicable application domain affects the meaning assigned to identifier names.

The experiment is discussed in two articles. The first, this one, discusses the background to the experiment and some of the applicable characteristics of the subjects taking part; the second provides a review of other studies that have investigated meaning assignment and discusses the results of the ACCU experiment.

2 Why is an experiment necessary?

Developers are often unable to give a coherent answer to how they assign a meaning to an identifier. This kind of human behavior (knowing something without being able to state what it is) has been duplicated in many studies.

A study by Reber and Kassin[2] compared implicit and explicit pattern detection. Subjects were asked to memorise sets of words containing the letters *P*, *S*, *T*, *V*, or *X*. Most of these words had been generated using a finite state grammar. However, some of the sets contained words that had not been generated according to the rules of this grammar. One group of subjects thought they were taking part in a purely memory based experiment, the other group were also told to memorise the words but were also told of the existence of a pattern to the letter sequences and that it would help them in the task if they could deduce this pattern. The performance of the group that had not been told about the presence of a pattern almost exactly mirrored that of the group who had been told on all sets of words (pattern words only, pattern plus non-pattern words, non-pattern words only). Without being told to do so, subjects had used patterns in the words to help perform the memorisation task.

3 How do developers assign a meaning to an identifier?

This issue is discussed in some detail in part 2 of this article. For the time being we assume that the meaning assigned to an identifier, by a developer, is created using a repertoire of previously learned techniques operating on a substantial body of knowledge they have in their head. The significant techniques and knowledge are assumed to be:

- experience of human and computer languages,
- knowledge of particular domains (e.g., software engineering concepts, or knowledge of an application domain such as accounting),
- experience in reading and writing source code,
- information obtained from the context in which the identifier occurs. This issue is discussed further in part 2.

3.1 Prior experience

Studies have found that nearly every task that exhibits a practice affect follows the *power law of learning*. This law has the form:

$$RT = a + bN^{-c} \qquad (1)$$

where RT is the response time, N is the number of times the task has been performed (not the amount of time spent performing the task), and a, b, and c are constants.

The value of c is usually greater than 1, which means that ever larger amounts of practice are needed to obtain the same increase in performance.

Studies have found that practice effects exist in peoples use of language. For instance, a large number of studies have verified that a *word frequency effect* exists. The number of times a person has been exposed to a (natural language) word has a significant affect on their recognition and performance in handling that word.

Education within specific knowledge domains has also been found to affect word handling performance. For instance, a study by Gardner, Rothkopf, Lapan, and Lafferty[1] used 10 engineering, 10 nursing, and 10 law students as subjects. These subjects were asked to indicate whether a letter sequence was a word or a nonword. The words were drawn from a sample of high frequency words (more than 100 per million), medium frequency (10-99 per million), low frequency (less than 10 per million), and occupationally related engineering or medical words. The nonwords were created by rearranging letters of existing words, while maintaining English rules of pronounceability and orthography.

The results showed engineering subjects could more quickly and accurately identify the words related to engineering (but not medicine). The nursing subjects could more quickly and accurately identify the words related to medicine (but not engineering). The Law students showed no response differences for either group of occupationally related words. There were no response differences on identifying nonwords. The performance of the engineering and nursing students on their respective occupational words was almost as good as their performance on the medium frequency words.

3.2 Domain knowledge

Given the likelihood that subjects taking part in the experiment would have a wide variety of software related backgrounds it was decided to attempt to restrict the domains they considered when answering experimental questions. Two domains that subjects were likely to be broadly familiar with were chosen. These two domains were operating systems (the Linux Kernel) and computer games (Doom, a product of ID Software).

It was hoped that specifying a particular domain would provide a global context within which it would be possible for subjects to provide a few answers, that they considered to be the likely meanings. For instance, the word *bank* has a number of possible meanings. Being told that it occurs in a financial context is likely to cause people to associate a meaning with it that is different than if they had been told it occurred in a discussion about rivers. It is also possible for a localised context to override a global context. For instance, discussing the watery view from an accountancy companies offices might suggest a river bank, while a discussion of the cost of restocking a river with fish might trigger finance related thoughts.

A number of subjects indicated (both after taking part in the experiment and through writing on their response sheet) that they were unable to provide a meaning to some identifiers because insufficient context was available to them.

3.3 Source code experience

Given that a subjects performance is driven by the amount of time they have spent performing the task, we need some way of measuring the amount of time spent working directly with source code.

Traditionally, developer experience is measured in number of years of employment (performing some software related activity). It is relatively easy for a person to calculate the amount of time they have been employed in a software development related role. However, the extent to which the amount of time spent in software related employment correlates with experience working on source code is not known (there are many software related employment activities that do not involve a person working at the source code level).

The quantity of source code (measured in lines, not time spent) read and written by a developer (developer interaction with source code overwhelmingly occurs in its written, rather than spoken, form) is a more direct measure of experience. Interaction with source code is rarely a social activity (one situation where it does occur is during code reviews) and the time spent on these activities may be small enough to ignore. The problem with this measure is that it is very difficult to obtain reliable estimates of the amount of source read and written by a developer. The problems include:

- readers don't always read code on a line by line basis. For instance, they may scan the source looking for some construct, or may only read part of a line,
- the same code is often read several times. Does each instance of reading have the same learning affect?
- code may be written and then rewritten or deleted (existing productivity measures are based on final number of lines of debugged code),
- few developers regularly measure the amount of code they write. This means they are very unlikely to be able to make an informed estimate of the total amount of code they have read or written.

Even although there appear to be significant problems in obtaining reliable answers, from developers, on the amount of source they have read and written your author believed something could be learned from the subjects responses.

4 Experimental setup

The experiment was run by your author during two (on different days) 30 minute sessions of the 2003 ACCU conference held in Oxford, UK. Approximately 250 people attended the conference, 45 (18%) of whom took part in the experiment. Subjects were given a brief introduction to the experiment, during which they filled out background information about themselves, and they then spent 15 working on the identifier list. All subjects (31 on the first day, 15 on the second) volunteered their time and were anonymous.

The requested subject background information was as follows:

- What is is your native language?
- Please list any human languages that you speak fluently:
- Please list any of these languages that you use at least once a week:
- Please list the computer languages that you have spent a significant amount of time reading and writing (at least 100 hours, i.e., 3 work weeks) over the last two years:
- How many lines of code would you estimate you have **written**, in total, over your career:
    1. 10,000
    2. 25,000
    3. 50,000
    4. 75,000
    5. 100,000
    6. 150,000+
- How many lines of code would you estimate you have **read**, in total, over your career:
    1. 10,000
    2. 25,000
    3. 50,000
    4. 75,000
    5. 100,000
    6. 150,000+
- How many years have you been writing software professionally?

4.1 Subjects background

On the first day 30 subjects handed in their completed response sheets (one subject was not happy with their performance and it was agreed that they could keep the sheet containing their responses), on the second day 15 response sheets were handed in.
Most of the subjects were native speakers of English, Table 1.

Table 1: Number of subjects having the given language as their native language. The response "*NL*" is assumed to refer to the country code for the Netherlands.

Native Language	Number of subjects
English	35
French	2
German	2
Italian	1
NL	1
Russian	2
Slovenian	1
Swiss German	1

The subjects regularly used a wide range of computer languages, Table 2. The most commonly used languages was C++, followed someway down the list by C, and Java (the ACCU is the Association of C and C++ Users, and a Python conference was also taking place in the same place at the same time).

Table 2: Number of subjects regularly using a particular computer language (*shell* was the generic term used for a variety of command line shells).

Computer Language	Number of subjects	Computer Language	Number of subjects
Assembler	1	ML	1
Basic	1	OLC	1
C	20	Pascal	2
C#	2	Perl	9
C++	41	PHP	3
Cobol	1	Python	6
Fortran	3	Rebol	1
Haskell	1	shell	6
HTML	3	SQL	3
IDL	1	TCL	1
Java	12	VB	7
Javascript	4	VBA	1
Lisp	1	VBscript	2
make	1	XML	3

Over 50% of the subjects had more than 11 years of experience in software development, Figure 1. Whether this figure is representative of the general population of developers (or even of those attending the conference; there were other events taking place throughout the extended lunch break, during which the experiments were run) is not known.
As Figure 2 shows, your author clearly underestimated the number of lines of code that subjects believe they have read and written (he also has to admit to thinking that the average number of years of professional software development would be lower).

4.2 Conclusions

Based on years of employment the majority of the subjects have a significant amount of software development experience. The measurements based on lines of code read are likely to suffer from incorrect self calibration on the part of subjects and a ceiling effect caused by overly restrictive response options.
Further conclusions are given in part 2 of this article.

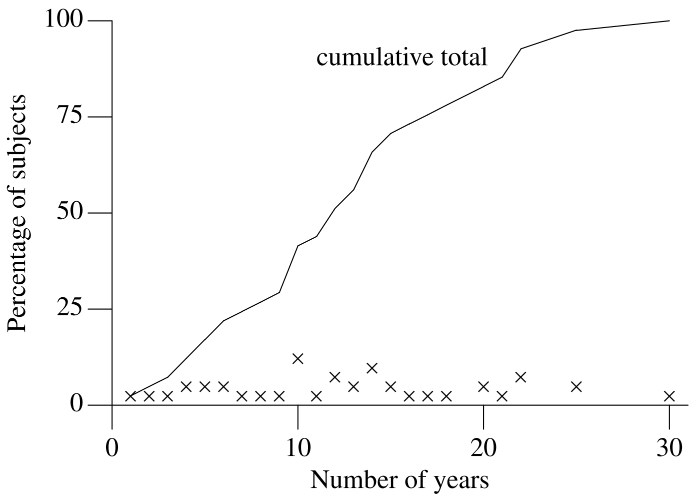

Figure 1: Cumulative percentage (line) of subjects who have been writing software professionally for the given number of years. The crosses represent the relative number of subjects having the given number of year employment. Four subjects did not provide an answer to this question.

Figure 2: The graph of the left depicts number of line of code read against number of years of writing software professionally. The graph on the right depicts the number of lines of code read against the number of lines of code written, for each subject. The size of the circle indicates the number of subjects specifying the given values. In those cases where subjects listed a range of values (i.e., 50,000-75,000) the median of that range was used.

4.3 Further reading

“*The psychology of language*” by Tevor Harley is an undergraduate level introduction to the subject. Those wanting a lighter read might like to try “*Word in the mind*” by Jean Aitchison,

“*Learning and Memory*” by John Anderson is an undergraduate level introduction to the subject. Those wanting a lighter read might like to try “*Essentials of human memory*” by Alan Baddeley.

4.4 Acknowledgements

The authors wishes to thank everybody who volunteered their time to take part in the experiment and ACCU for making conference slots available to run it.

References

At the end of each entry, the pages on which that entry is cited are listed in parentheses.

[1] M. K. Gardner, E. Z. Rothkopf, R. Lapan, and T. Laferty. The word frequency effect in lexical decision: Finding a frequency-based component. *Memory & Cognition*, 15(1):24–28, 1987.

[2] A. S. Reber and S. M. Kassin. On the relationship between implicit and explicit modes in the learning of a complex rule structure. *Journal of Experimental Psychology: Human Learning and Memory*, 6(5):492–502, 1980.