The Misplaced Comma: Programmers' Tales and Traditions

Lindsay Marshall
Department of Computing Science
University of Newcastle upon Tyne, UK
Lindsay.Marshall@ncl.ac.uk

Jim Webber
HP-Arjuna Labs
Newcastle upon Tyne, UK
Jim.Webber@hp.com

Keywords: POP-I.A. group dynamics, POP-I.B. transfer of competence, POP-II.C working practices,

Abstract

Programmers do not exist in isolation and there is communication between them about the technical aspects of their work. Sometimes this takes the form of stories and anecdotes and this paper suggests that study of these programmers' tales and how they spread could provide useful insights into how programmers perceive their work.

"A real Magical Oath cannot be broken: you think it can, but it can't.
That is the advantage of a real Magical Oath."

Aleister Crowley, Magick

Introduction

In an earlier paper (Marshall 2000) we explored, albeit tentatively, the idea that programmers are often in thrall to a variety of taboos that unconsciously influence the way that they approach design and implementation. This paper looks at some other ways in which, what are often described as "primitive" influences may operate on the work of programmers. We believe that by taking this slightly unconventional view, we can bring out interesting aspects of programmer behaviour that might repay further study.

We shall look at two general areas, both of which follow on naturally from thinking in more general terms about taboos. The first, which we categorise as *lore*, concerns the ways in which programmers learn and communicate with each other. The second, which we shall call *magic,* is much more to do with the process of implementation.

Lore[1]

Think about the stories about programming you remember. Not about things that have happened to you, but ones you have heard from other people about their work or about the work of others. Stories that you may even have told to others yourself. How many can you come up with? Probably not too many. The ones that spring instantly to our minds are "the comma that crashed the spaceship" [Tropp 1984] and "upside down over the equator" [Neumann 1980], but after these nothing else that directly relates to the practice of programming. Certainly nothing from which an obvious, simple lesson could be learned and the original essence of a lot of folk tales is the passing on of directly useful information, even if disguised as entertainment or history.

[1] We have used the term Lore rather than Folklore as the latter term is a 19th century coinage often used either to refer to the study of folk tales rather than the tales themselves or else to more general storytelling. Lore is also a nicer word.

Why aren't there more programmers tales? Especially now that the Internet – which is, after all, full of programmers – will let them spread fast, gathering embellishment as all tales do. Is it that programmers don't communicate with each other much? And why are the tales that spring to mind invariably negative – where are the stories about the programmer as hero? Legends about programmers pulling all-nighters to meet deadlines are more lessons about avoiding bad management than programming skill, and the actual skill used is rarely quantified, taking on a rarefied, mystical aspect that tells the listener nothing. (In practice the skills exhibited are often nothing more than good fluency in a programming language and a superhuman ability to survive for periods of time on no sleep, pizza and coffee, but that is a side issue)

The idea of "Not reinventing wheels" is more of a programmers' proverb than a folk tale as it does not come backed up with examples of success and failure. However, the idea of code reuse seems to have caused the beginnings of a set of folk tales with the rise of the Design Pattern community [Gamma 1994], though they behave more like folklorists than story-tellers in that they are by nature classifiers and cataloguers. Design patterns aim to provide little templates for solutions that attempt (not always successfully) to help programmers break down problems thereby helping them to make fewer errors. Just as a hunter's stories tell about how to find the best game. However, design patterns tend to be over general and it is not clear whether or not we will see developments reminiscent of personal recipe books that are passed down to children or apprentices.

Mentioning apprentices raises the issue of programming as craft – where we learn by watching and doing as much as we do by formal teaching. The absorption of taboos is clearly a factor of this process, but taboos are not lore as they are unconscious. As we get further and further away from the actual hardware that runs programs it seems that the amount of lore diminishes. Machine code programmers were always full of ways of saving time and space and (usually) passed their tricks on unstintingly, and with relish. In higher level languages these tricks become idioms as they are less tied to specific circumstances, and at the outermost levels we get design patterns. The trouble is that, just as when a folk tale gets taken out of its context, design patterns lose there roots and so are less useful than they might be. Generalising an idea does not always lead to useful results.

The rise of the open source movement also has aspects of lore to it. By providing freely available examples of how to solve particular problems, practical knowledge spreads and different approaches get compared and evaluated. The experience of looking through other people's code can be highly rewarding in terms of skill development, especially debugging. And some bits of code get reused (for example, unix socket handling code which almost everyone copies) and turn into folk tales. And perhaps this is why there are so few programmers' tales: they exist not in natural language but in code and become embellished in the same ways and changed in the translation to other languages.

Magic

Just like hackers' hats, magic comes in different colours and kinds, but here we want to talk mainly about the idea of *sympathetic magic* of which the most well-known example is sticking pins in a voodoo doll. In many ways programming is *all* sympathetic magic: programmers often solve problems by simulation, by constructing models and operating on them. But there is more to sympathetic magic than this.

Programmers all have their own styles. They solve problems in similar ways all the time, for example Rob Pike likes to use little compilers embedded in his applications. Is this laziness? Inertia,? Or perhaps it is a little bit of sympathetic magic, another aspect of which is that if we find a way of doing something that works well we stay with it no matter how irrational, like carrying a lucky charm to an exam.

We see sympathetic magic at work in the naming of functions: we tend to call them after what we want them to do rather than what they actually do, a classic case of this in the AI world is discussed in [McDermott 1982]. And perhaps it is at work too in the design of programming languages where we are often forced always to do things because *sometimes* they are necessary. An example of this is variant record handling in Modula derived languages where you must provide cases to deal with all

possible variants, even when you know that only a subset of the variants will ever crop up at a particular place in a program. Yes, things could go wrong and this would catch them, and so is therefore a "good thing", but often it seems like being forced to do the right thing because it has been deemed to be the right thing (which leads to frustration and hence the sloppy coding of the unused variants, which introduces an error…)

You often see programmers, especially novices, writing slightly odd looking pieces of code. When asked why they do it that way, the answer is often that once they had an error in a similar situation, that they worked round the problem and now always use the work around when they think the case might be the same. A sort of programmatic talisman to protect them from errors, which eventually turns in to a superstition, and possibly sublimated into a taboo. Note that they might also have got the odd code as lore from someone else who had had the same problem. This often happens in introductory programming practicals. A one-off fix becomes a charm that is passed around as a tale of experience. Usually, however, these have only a limited currency and soon die out, particularly as the programmers become more experienced, but sometimes people who do not take their programming further preserve these little quirks because they never learn that they have no actual value. (Interestingly, [Spohrer 1986] refers to "folk wisdom" in the context of people *teaching* programming, though the paper is not concerned with the process by which this wisdom accumulated)

Eventually the charm becomes a reflex. Programmers who grew up with smaller, slower machines will often do things like a shift or an addition to multiply by two, because it used to be faster. Often however this makes the program *slower* because it interferes with the much improved optimisers in today's compilers.

Conclusions

First of all, we should stress that all the above is pure speculation. We have not carried out an anthropological study of programmers' beliefs – we are programmers not anthropologists, or psychologists. Our suggestions may be well off the mark. However, as we said in the introduction, our aim is to suggest possible routes for future exploration, we do not guarantee that they will lead anywhere useful. Two areas for exploration immediately suggest themselves. First, the construction nad classification of a comprehensive collection of software war stories. Several sites on the web have made a start on this (e.g. http://www.cs.tau.cs.ac.il/~nachumd/verify/horror.html) but none are comprehensive. The second would be to monitor introductory programming classes to see if the lore creation process could be observed taking place. It would be particularly interesting to see if it happened where teaching was at a distance and communication between students was not face to face.

What can we glean from this ragbag of ideas? First, that encouraging the spread of programmers' lore could be a good thing. It used to work for machine code programmers. It still works in programming classes with bad ideas, so it ought to be workable with good ideas. If there really is a process of lore creation then perhaps it could be subverted to introduce the "proper" way to do things. The design pattern people are trying to do some of this, though not always in the most accessible or useful way – perhaps they should use stories rather than formalism to get their ideas across.

Second, that there is more to understanding programmers and programming than looking at processes and outputs. As we tried to show with respect to taboos, social interaction can play an important rôle in how programmers' knowledge and experience develops. Their anecdotes and the way they mutate and spread could tell us much about how programmers think about a variety of issues as well as how the way that programmers interact with each other is changing.

References

Gamma, E. et al (1994) *Design Patterns* Addison-Wesley

Marshall L. and Webber J. (2000) Gotos Consider Harmful and other Programmers' Taboos. In the proceedings of *PPIG* 2000

Neumann P.G. (1980) Editorial *Software Engineering Notes,* **5** (2) : 5

Spohrer J.C. and Soloway E. (1986) Novice Mistakes: Are the folk wisdoms correct? *CACM*, **29** (7) : 624-632

Tropp H.S. (1984) FORTRAN Anecdotes *Annals of the History of Computing,* **6** (1): 61-62

McDermott D. (1982) Artificial Intelligence Meets Natural Stupidity in *Mind Design* John Haugeland Ed., MIT Press 143-160