Technosectarianism: Applying Religious Metaphors to Programming

Stephanie Stuart Mawler[1]

Abstract. Technosectarianism is a new term presented and discussed in this paper, and intended to represent the group definition actions taken by programmers through their supposedly purely objective and technical interactions, which are driven more by religious-like concepts including orthodoxy, partisanship, apostasy, and heresy.

1 INTRODUCTION

The passion held by programmers for their tools, products, and practices can be viewed as a form of sectarian competition within a technologically-dependent profession – a situation that can be labeled technosectarianism. This new term couples the traditional notion of sectarianism with the technological systems and surrounding beliefs to which a given community of programmers adheres. Formally defined, technosectarianism is boundary maintenance behavior within programming-related communities, where such communities come together around a particular technology system. Such behavior requires a faction that defends the technology system by establishing orthodoxies regarding any and all aspects of that tool and the identification of partisans to defend the established orthodoxies. However, the behavior also requires an opposing faction driven by heretics and apostates, with heretics openly questioning the entire structure of orthodoxies and apostates who introduce concepts that do not fit into established orthodoxies.

Technosectarian battles are waged in a wide variety of locations, including written interactions within the technology-focused communities. Typically, these interactions occur in places that are visited almost entirely by other highly-technical participants. These written interactions are normatively believed to be objective, identity-neutral, and purpose-oriented by the community members. Contrary to this belief, technology-focused communities rely on these interactions to become the crusaders' banners in the technosectarian conflicts.

Identification of technosectarianism provides a label for what is often considered typical behavior of dedicated technologies, and it also allows a greater understanding of technology-focused communities. While such communities possess a shared lexicon, the foundational structure for the community is the technology system. The technology system itself is comprised of languages and lexicons, to which a near sacred status is applied.

The adoption of religious metaphors, while intuitively appealing, is also supported philosophically by [1] where he discusses his concept of the alterity relation to technology. This relationship highlights the similarity between "a contemporary form of anthropomorphism" and the sacred role of objects "in ancient or non-Western cultures".

Within the programming field, anthropomorphism is at least as commonplace as it is in the rest of contemporary society. Applying the concept to any programming community, the sacred object is the language and/or its syntax. Paraphrasing [1], a programming language does not "simply 'represent' some absent power but is endowed with the sacred". In a very real sense, programming languages no longer represent power, but actually contain that power, which makes them "sacred" to their practitioners. Continuing with [1], programmers will "defend, sacrifice to, and care for the sacred artifact. Each of these illustrations contains the seeds of an alterity relation". If the uninitiated assails the sacred language, the language is defended. Specific syntactical arrangements are also defended.

To expand upon the notion of technosectarianism, this paper looks at two different interaction types: code comments and newsgroup posts / threads. For the code comments, programs were selected from the Linux kernel and a real-time messaging user-interface called Pidgin. For the newsgroups, a timeframe was selected for analysis within comp.lang.c++ and comp.lang.lisp. First, however, it is useful to look at programming languages more generally, as these are the core sacred object.

2 PROGRAMMING LANGUAGES

There is an incredible diversity of programming languages currently in active use, and programmers make a choice among programming languages for as many different reasons as the languages are originally created. All other things being equal, the free selection of a given programming language says more about the programmer than it does about the task for which it will be used. A programming language may be a metaphor for the social organization of a community of programmers, and a shared language is one of the foundations of an established community and the programming language is the most basic shared language of a given programming community.

[1] Virginia Tech, Falls Church, Virginia, US, Department of Science and Technology Studies, Virginia Tech, Falls Church, VA, US. Email: smawler@gmail.com.

Importantly, as [3] said, “just as natural languages constrain exposition and discourse, so programming languages constrain what can and cannot easily be expressed, and have both profound and subtle influence over what the programmer can *think*”(emphasis in orig). Sherry Turkle agrees in [4], asserting that “different computer languages and architectures suggested different ways of thinking”. From a purely technological standpoint, the language (and, to a large degree the environment around the language) determines what can be thought and programmed, but the culture that is built up around the language also limits what can be thought of and executed within that language.

Finally, while programming languages are metaphorical, the metaphors also apply in the other direction, with the language itself becoming a metaphor for the communities that use them, proving that languages have yet to “retreat to the background”, as suggested by [5], and remain a central part of the programming discourse. In some senses, arguments over language and semantics are both the root of the discourse and a smokescreen around other substantive issues (or lack thereof). The programming language becomes a metaphor for the community that uses the language, as in “C programmers are just so arcane and have very little grasp of interpersonal relations,” or “UNIX programmers just cannot GREP the solution,” or “BASIC programmers are just a series of GOTO statements,” or “COBOL programmers are just a series of MOVE statements,” and many other statements that have formed a part of my own personal experience.[2]

There are other ways that languages are used as metaphors that help establish community. Each language “entails different styles of programming and suggests different modes for conceptualization,”, as [5] indicates, and may also suggest entirely different ways of forming communities around different metaphors embodied in the language(s) used in that community, with a language’s cultural implications, based in the technology and syntax of the language, and sometimes even the name itself.

3 PROGRAMMER COMMENTS

Comments are natural language footnotes or in-text commentaries that appear within individual programs. Comments are for the benefit only of human readers and have no bearing on the program execution.

So why include comments, when the code that actually performs the operations is immediately available? At least one author, [6], asserts that a readily available natural language explanation of the code “will have a much larger influence on the speed with which the programmer understands the program than variations in the structure of the program.”[3] One of the most direct guides for creating comments comes from [7], where comment authors are requested to “put a comment on each function saying what the function does”, including its arguments and anything non-standard or unexpected.

Therefore, the normative purpose of comments is to explain the code. Comments conforming to this purpose are normatively “good” in that they support the stated practices of the profession.

Beyond the normative definition of “good”, comments perform many additional functions, including as a means to prevent execution of a line of otherwise machine-ready code by turning the code itself into a comment. The practice is a part of the orthodoxy of given communities – it is accepted in some and anathema in others.

“Commenting-out” code means that old code is visible to later programmers, but the compiler does not convert it into object code for the computer to use. Commented code can, in theory, simply be “uncommented” to be resurrected and adds a sense of history to the text of the program. However, the concept has been parodied as a counter-norm in [8], where the author writes that programmers should “be sure to comment out unused code instead of deleting it and relying on version control to bring it back if necessary. In no way document whether the new code was intended to supplement or completely replace the old code, or whether the old code worked at all, what was wrong with it, why it was replaced etc.” The parody here takes the position that the commented code is merely chaff that makes a subsequent developer’s job more like that of an anthropologist or archaeologist.

In the Linux kernel, unlike the Pidgin source code, there are absolutely no examples of code commented out, which does not indicate a seamless development path free from re-writes. Rather, the lack of code that has been commented out points to a stylistic and ideological decision that helps form the orthodoxy of the community. The old code no longer represents knowledge for the Linux kernel contributors and, to them, its presence would simply add confusion. To view the changes over time, a person would

[2] I have personally worked in the IT field, alongside programmers using various language and platforms for more than twenty (20) years. In that time, I have heard statements like this so often that they become an unacknowledged element of the culture.

[3] This assertion seems to conflict directly with another, much more well-cited study. Weinberg cites an IBM sponsored “experiment in which several versions of the same code are produced, one with correct comments, one with one or two incorrect comments and one with perhaps no comments at all” (Okimoto, 1970, in Weinberg, p. 164), concluding that “for certain types of code, at least, correct interpretation of what the program does can be obtained more reliably and faster without any comments at all” (Weinberg, p. 164). This same 1970 IBM study continues to be cited into the second half of the 1990s, with authors using it as a basis to conclude, similar to Weinberg, that “studies of short programs show that comments in code interfere with the process of understanding, [and] if not up to date, can be misleading and cause errors in the semantic representation of the code” (Rugaber).

need to compare previous versions of the same file, or use the tools of version control, as implied in the quip about unmaintainable code. While removing the old code does tend to lend clarity, it also might create a false sense of inevitability, as though the edits needed to bring the code to its current state were less extensive than they might actually have been.

The text of one comment within [9] is illustrative of the boundary defense and the group identification being performed by a programmer within the context of the code. The text reads, without original formatting:

setuid() is implemented like SysV with SAVED_IDS

Note that SAVED_ID's is deficient in that a setuid root program like sendmail, for example, cannot set its uid to be a normal user and then switch back, because if you're root, setuid() sets the saved uid too. If you don't like this, blame the bright people in the POSIX committee and/or USG. Note that the BSD-style setreuid() will allow a root program to temporarily drop privileges and be able to regain them by swapping the real and effective uid.

In the actual comment presented above, a programmer (unidentified in the context of the comment) provides an explanation of how the code works in the first line. That first line is short-hand intended to let later readers understand the intention of the code, should there be changes or problems that might require changes.

The subsequent lines of text (seven in the original) do not actually explain the code to which they refer, instead performing several additional, counter-normative functions. First, the author provides background to his/her argument regarding the effectiveness of the solution, while explaining possible downfalls of the solution. Second, the author takes aim at those s/he considers to be the driving factors behind the change, clearly implying that they do not have sufficient knowledge to shape the solution in this way. Thirdly, the author provides an alternative solution, explaining the source of the solution, and its benefits. Finally, and most importantly, the author uses the comment to distance him/herself from the solution – noting that s/he considers the solution “deficient”, placing “blame” onto an outside party, and claiming preference for a completely different approach fostered by a different brand of Linux with which our author is clearly intimately familiar. The implication of this distancing is that other programmers, whose opinions would matter to the author, might question the change and how it was implemented – the comment lets these other readers know that the author understands the situation within the code and beyond and would make different (and better) choices if s/he could.

The comment sets up an opposition between insider and outsider, through the use of a grammatical construction: the use of “you”. This form implies that some readers require education in the ways of the industry and that questioning the programming decisions made in this section is inappropriate and not something done by those on the inside—in the know. In this case, the outsider is a special case, since they can read the code. The outsider is likely a new or potential member of the Linux contributor community.

The author also invites the reader to come inside the core Linux boundary, as a means to deflect criticism from his/her “deficient” code. The author’s rhetoric can be seen as an attempt to create solidarity between him/herself and the reader (“you”) by speaking to an assumed distaste for an external bureaucratic enemy (“the POSIX committee and/or USG”), who exist beyond an additional boundary and who should be blamed for the problem. In a sense, the author invokes political savvy to redeem a technical shortcoming, using his/her expertise to shift the boundary, potentially deflecting criticism.

Technosectarianism in this case is realized as orthodoxy and partisanship. The author of the comment is strongly in “agreement with the doctrines, opinions, or practices currently held to be right or correct”, quoting [10] in application to this instance. The belief system is evident in the language directed at the POSIX committee specifically, where the committee’s beliefs are clearly seen to be incorrect / wrong. In the representation of the orthodoxy, the author is clearly “an adherent of a cause”, paraphrasing [11] in reference to the instance, where the cause is simultaneously systems programming and the Linux project itself.

4 PROGRAMMING NEWSGROUPS

A programming language is a lexicon; hence, any two people using the same language inherently share a lexicon, or repertoire refined through discussions. Many programmers come to a specific language by accident, but there are those who choose the language and thus become advocates – partisans – for the language. Shared vocabularies are literacies that can be used for behavioral monitoring and self-regulation, as [12] argue, and this is exactly the sort of activity that is found within technical community newsgroups. [4]

According to [13], USENET began in 1979 as a somewhat informal “news exchange system between Duke and the University of North Carolina, using dial-up connections”. USENET was divided up into separate newsgroups devoted to particular topics and, notably, “users could create newsgroups on any topic they wanted to discuss”. All newsgroups are oriented around threads, established when an individual submits a comment or question with a subject and all responses are organized under that original message. Computer users were early

[4] This study is based on analysis of two newsgroups: comp.lang.lisp and comp.lang.c++

adopters of USENET, forming newsgroups “focused on practical matters of using and operating computers”.

So why do people post to programming-related newsgroups? The normative use of a newsgroup post within most technical communities is to pose or answer vexing or intractable technical questions. These problems may be related to a general concept, a specific code snippet, or an interpretation of details within a standards document, but they are essentially problem-related. Within each interaction, participants can be assigned one of two roles, ‘original posters’ (“OP”) who initiate threads and respondents. Each role has different styles, genres, and tropes, following the work of [14]. In general, community members establish themselves nearly exclusively in only one of these roles, conforming to the normative purposes of the newsgroup post as an interaction event around problem-solving.

Questions in a newsgroup can take many forms and levels of complexity, from “how do I get started” to “how does one interpret this paragraph of a standard”. Correspondingly, solutions have different forms. Participants who provide answers express many different justifications for their participation. One particularly common justification, used in [15], says that those who provide solutions are “here for the newbies” (those new to the language).

However, the very assertion of these norms and attempts to adhere to them hide equally important examples of technosectarian behavior within these same interaction events. Besides simply posing and answering difficult technical questions, newsgroup posts determine what counts as knowledge within a particular programming community; they create and maintain group mythologies; they pass along shared histories; they validate and maintain programming practices; they maintain and strengthen boundaries with other languages and partisan feelings for the language at hand; and they establish the identity and credibility of individuals within the community. In addition, participants are also active in the groups because they feel strongly about the language to which the group is devoted – they are partisans.

Within [16], we have a potentially normative thread that quickly becomes counter-normative. From the normative perspective, this thread might be seen as a discussion of difficulties experienced by the OP in working with Lisp, the language to which the newsgroup is devoted. In this interpretation, the counter-normative aspect of the OP can be seen in the tone, which might be considered confrontational, if based only on the subject: “The Fundamental Problems of Lisp”.

However, the OP makes clear that the post is not a question, but an assertion. The author asserts that the language is deficient in several key ways, including inherent flaws in the basic syntax of the language. These problems are defined by the author as “damages lispers has done to themselfs” (sic). The author spends much time comparing Lisp to a very different language and platform, to which there are other newsgroups devoted.[5]

The subsequent discussion results in the longest single thread in the newsgroup with nearly 120 responses. Despite the counter-normativity of the OP, many of the responses might be considered normative – the authors consider the assertions and provide reasoned responses with potential solutions and/or alternative views. However, the majority of the responses assail the OP. One suggests that the OP should consider using the language to which Lisp was most often compared. One branch of the discussion even changes the subject to “The Fundamental Confusion of Xah” (the OP).

Technosectarianism is realized in this case primarily as heresy and apostasy. The OP clearly maintains opinions that are “at variance with those generally accepted as authoritative” quoting [17], where participation in the newsgroup is a matter of choice, involving elevation of the language itself to a sacred object. These assertions, therefore, are simply heretical. Furthermore, by active participation in the Lisp newsgroup over an extended period of time, these controversial assertions act as a form of apostasy, where the OP has gone beyond heresy and appears to have “forsake[n] his allegiance”, applying [18].

5 CONCLUSION

The two instances above, code comments and programming-related newsgroup discussions, display, at the very least, that supposedly objective and purely technical interactions contain far more meaning and use than the community norms would indicate. Understanding these non-normative uses exposes new source materials for those researching technology practitioners. For practitioners, identifying the non-normative uses might result in different approaches to either creating or using unexamined tools, processes, or standards. Programming teams might change the way they create or use comments. Mentors, teachers, and professors might reflect on their own approaches to guiding teams or students by either avoiding or leveraging the non-normative uses of their various interactions.

Technosectarianism provides a means to identify a whole set of behaviors and beliefs that lend themselves particularly well to such religious metaphors, while helping those on the inside and on the outside of technology-oriented communities appreciate the degree to which a technology system is, itself, the foundational structure for such communities.

[5] The language of comparison is Mathematica.

REFERENCES

[1] D. Ihde, “A Phenomenology of Technics,” in *Philosophy of Technology: The Technological Condition: An Anthology*, R. C. Scharff and V. Dusek, Eds. Blackwell Publishing, 2003.

[2] E. Semino, J. Heywood, and M. Short, “Methodological problems in the analysis of metaphors in a corpus of conversations about cancer,” *J. Pragmat.*, vol. 36, pp. 1271–1294, 2004.

[3] M. L. Scott, *Programming Language Pragmatics*, 3rd ed. Burlington, MA: Morgan Kaufmann Publishers, 2009.

[4] S. Turkle, *The Second Self: Computers and the Human Spirit*, Twentieth Anniversary Edition. Cambridge, MA: MIT Press, 2005.

[5] J. Pflüger, “Language in Computing,” in *Experimenting in Tongues: Studies in Science and Language*, M. Dörries, Ed. Stanford, CA: Stanford University Press, 2002.

[6] R. Brooks, “Using a Behavioral Theory of Program Comprehension,” in *Proceedings of the 3rd International Conference on Software Engineering*, IEEE Press, 1978, pp. 196–201.

[7] R. Stallman, “GNU Coding Standards,” *Free Software Foundation*, 08-Feb-2006. [Online]. Available: http://www.gnu.org/prep/standards/. [Accessed: 16-Feb-2006].

[8] R. Green, “Unmaintainable Code,” *Canadian Mind Products*, 2006. [Online]. Available: http://mindprod.com/jgloss/unmain.html. [Accessed: 14-Apr-2006].

[9] Linux Kernel Organization, *sys.c*. 2006.

[10] “orthodoxy, n.,” *OED Online*. Oxford University Press, Jun-2012.

[11] “partisan, n.2 and adj.,” *OED Online*. Oxford University Press, Jun-2012.

[12] D. Barton and M. Hamilton, “Literacy, reification, and the dynamics of social interaction,” in *Beyond Communities of Practice: Language, Power, and Social Context*, D. Barton and K. Tusting, Eds. Cambridge, UK: Cambridge University Press, 2005.

[13] J. Abbate, *Inventing the Internet*. Cambridge, MA: MIT Press.

[14] N. Fairclough, *Analysing Discourse: Textual analysis for social research*. New York: Routledge, 2003.

[15] “comp.lang.lisp,” *comp.lang.lisp*. 01-Jul-2008.

[16] X. Lee, “The Fundamental Problems of Lisp,” *comp.lang.lisp*. 13-Jul-2008.

[17] “heresy, n.,” *OED Online*. Oxford University Press, Jun-2012.

[18] “apostasy, n.,” *OED Online*. Oxford University Press, Jun-2012.