The International JOURNAL of TECHNOLOGY, KNOWLEDGE & SOCIETY

Volume 4, Number 4

Executable Texts: Programs as Communications Devices and Their Use in Shaping High-Tech Culture

Stuart Mawler

COMMON GROUND

www.technology-journal.com

THE INTERNATIONAL JOURNAL OF TECHNOLOGY, KNOWLEDGE AND SOCIETY
http://www.Technology-Journal.com

First published in 2008 in Melbourne, Australia by Common Ground Publishing Pty Ltd
www.CommonGroundPublishing.com.

ISSN: 1832-3669
Publisher Site: http://www.Technology-Journal.com

THE INTERNATIONAL JOURNAL OF TECHNOLOGY, KNOWLEDGE AND SOCIETY is a peer refereed journal. Full papers submitted for publication are refereed by Associate Editors through anonymous referee processes.

Typeset in Common Ground Markup Language using CGCreator multichannel typesetting system
http://www.CommonGroundSoftware.com.

Executable Texts: Programs as Communications Devices and Their Use in Shaping High-Tech Culture

Stuart Mawler, Virginia Tech, UNITED STATES

Abstract: This paper takes a fresh look at software, treating it as a document, manuscript, corpus, or text to be consumed among communities of programmers and uncovering the social roles of these texts within two specific sub-communities and comparing them. In the paper, the social roles of the texts are placed within the context of the technical and cultural constraints and environments in which programs are written. Within that context, the social roles of the comments are emphasized, and are combined with the normative intentions for each comment, creating a dynamic relationship of form and function for both normative and identity-oriented purposes. The relationship of form and function is used as a unifying concept for a more detailed investigation of the construction of comments, including a look at a literary device that relies on the plural pronoun "we" as the subject. The comments used in this analysis are derived from within the source code of the Linux kernel and from a Corporate environment in the US.

Keywords: Programming, Programming Culture, Code Comments, Communications, Open-Source

Introduction

THIS PAPER REFOCUSES the view of software away from the utilitarian purposes of its completed form and toward its uses as both a communications medium and the central medium for discussing and defining values within computer culture. To reach its completed form, the vast majority of software is written and maintained in human-readable text (called source code), then converted by another piece of software (the compiler / interpreter) into machine-readable executable code (also called object code). In addition, software is usually not composed of a single program, but is made up of many small programs and other file types that serve specific purposes. These additional files are also written in human-readable formats and may be used in this format by the executable code. Taken together, I have dubbed any text-based file an "executable text," since these texts are intended to be used in the execution of a function or program, but are also intended to be written and read as a text. In this situation, a "text" is to be viewed with almost Biblical connotations, in the sense of a document that can be opened and read, a manuscript that is written "by hand" and pored over by other developers, or a corpus representing the shared knowledge of the group.

Contemporary programming style guides urge programmers to "code for human consumption" (Jones). Since these texts are to be read, we can conclude that the executable text is a communicative device, as would be expected of any text. When programmers look at source code, the code provides information about the program's functions; it acts as a repository of information. In addition, most programming languages have a built-in capability to include natural language comments, without impacting the operation of the running code (often being removed by the translation step from human to machine-readable). These comments are considered "documentation."

The normative structures and processes of programming encourage individual programmers to include comments within the body of the text. While the code is considered by many to speak for itself (Kernighan & Plauger, 151), practices encourage comments as a means to address issues of complexity and wider context that might make the code within one program difficult to understand. The comments illuminate specific modes of programming, complicated algorithms, history of changes to the text, or some collective information on the specific business or technical problem being addressed. Comments generally serve to make modifications, corrections, and enhancements simpler in the future—they "tell you (and any future developer) what the program is intended to do" (Jones) and provide this information beyond the view of most "ordinary users." However, comments serve many roles beyond the norms espoused by industry leaders.

The Source Archives

To complete this study, I have relied on two collections of executable texts from vastly different technological frames; the Linux kernel and a Corporate sample from a proprietary environment. Linux is an open-source operating system, meaning that the source code is available freely on the internet and

development occurs collaboratively across a wide, and voluntary, community. The kernel was selected for greater attention since it is a key portion of the operating system, handling interaction between the system and the hardware itself. Within the Linux kernel, I have focused on 12 specific executable texts of varying lengths.

The Corporate sample comes from an internal IT department supporting the consumer customer database of a large US-based corporation. The seven texts in the set are all COBOL modules, intended to be executed on the mainframe, where the texts were originally composed and all subsequent maintenance occurs. The centrality of the customer database to business goals, coupled with the age of the system makes the comparison with Linux kernel relevant.

Constructing Comments

While the normative intent and purpose of comments is a factor in their construction, there are specific considerations driven by a combination of the language used (C versus COBOL), the environment (PC-based versus mainframe-based), and the culture (open-source versus proprietary) in which it is used.

Language

Though almost every language retains the ability to include natural language comments along with the symbolic logic of the program code, the format of the comments is different in each. In C, comments all begin with a "forward slash" ("/"), followed by an asterisk ("*"), and are concluded by the reverse—an asterisk ("*"), followed by another forward slash ("/"). All information between those two sets of marks will not be included in the operation of the program. While the compiler (the program that converts the symbolic logic in the text from source code to object code) only requires the starting and ending marks, individual developers may choose to use other characters as purely visual devices, making human identification of the comment quicker and easier.

Comments within COBOL are signified by an asterisk ("*") at the start of the line (which will be numbered sequentially by the environment, unlike C, which has no embedded line numbers). All information on that line to the right of the asterisk will not be included in the operation of the program. Only one asterisk is needed to remove a line from the production run of the program. As with C, beyond the characters required for a comment, all other characters are purely for human convenience, style, or information.

In addition to creating natural language comments, comment characters (e.g., the asterisk) can also be used to prevent execution of a line of otherwise machine-ready code. Programmers refer to these as lines of code that have been "commented out."

Environment

Creating and maintaining the C source code for Linux is done in a wide variety of ways, according to the preferences of the individual developers, who each select their own "graphical" editor to manipulate the code. These graphical editors can be configured to highlight comments (among other items) in different colors, helping to visually separate them from the symbolic code. In addition, graphical editors can be configured to display many lines on the screen, simply by changing window size or screen resolution.

In contrast, creating and maintaining COBOL programs purely within the mainframe environment offers programmers an interface paradigm (in the Kuhnian sense) that has been abandoned by almost all other disciplines. In a pure mainframe environment (as in the Corporate sampling), there are no graphical user interface tools beyond monochrome text on a monochrome background, meaning that comments, section headings, and any other notable items must be highlighted with text characters set off from the code by an asterisk at the start of the line. However, the mainframe environment presents an additional technology-specific constraint: screen "real estate." A visually arresting comment must be balanced against occupying many lines on a screen that only displays approximately 30 lines at a time, depending on user configuration. This tension between information and screen real estate results in some extremely terse comment styles. In one representative example, a single line contains the programmer's identity, the date, the project spawning the change, the release that included the change, and the fact that the line was the start of a multi-line edit (Corporate Source Code, Program-1).

Culture

In the Linux kernel, the most important influence on the form and tone of comments is the collaborative and voluntary nature of open-source development. Unlike a proprietary environment, Linux developers take on a task because it is of ideological or artistic importance to them, rather than just being a job. As a result, there is a generally congenial tone within the comments, even where there are debates about the appropriate method to code a particular passage. Out of the set of programs, I found no instances of overtly antagonistic language, while such language was common in the Corporate sample.

Another culturally specific practice is the use of comments to mark the start and/or end of a change to the symbolic code. Following this practice, a programmer would create a comment immediately above

the line(s) of code to be changed and possibly one directly below the last line of code to be changed. In the Corporate sample I have found this practice quite common, but it does not exist within the Linux kernel.

In the Corporate sample, it is the size and scope of work, together with both the hierarchical nature of the organization and the linguistic and technical environment, which most strongly influences the form and content of the comments. As in the example cited earlier, terseness is valued as a result of the screen constraints, but further identification of developers beyond initials is largely unnecessary since the staff is close-knit and relatively static over time.

Mapping Comments: Form/Function Grid

Network interoperability conferences have been said to "highlight the performative nature of standards" (Slaton & Abbate, 135). Similarly, comments have standards or norms of form and function, but they also serve a performative or identity-oriented function in both the Linux and Corporate environments. In each, the individual writes for the consumption of the group, staging his/her own performance with each comment, and how the individual chooses to stage that performance reflects on the group dynamic in which that performance occurs.

In order to better visualize the many roles a single comment may play within an executable text, I have conceptualized each comment as consisting of both form and function, which I have then mapped on a grid (see Figure 1).

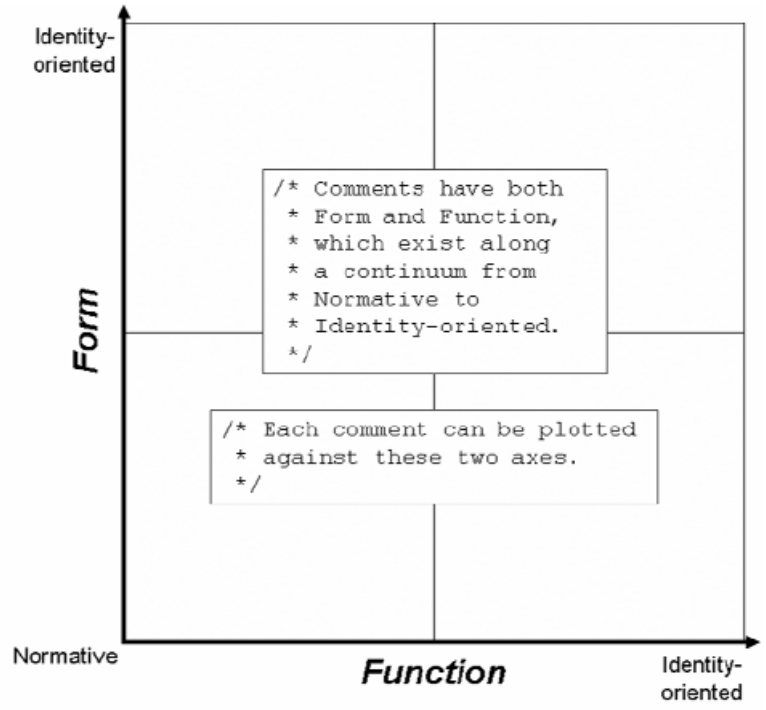

Figure 1: Form & Function Framework for Code Comments

The grid is in the form of an x-y axis, where the origin represents both normative form and function. Traveling to the right, along the x-axis, the function becomes increasingly identity-oriented. Traveling up, along the y-axis, the form becomes increasingly identity-oriented. Hence, a completely normative ("good") comment (in both form and function) would be plotted at or adjacent to the origin (as in the case of the inline COBOL comment referenced above), while a completely identity-oriented comment would be plotted in the far upper-right corner.

Rarely is a comment completely devoid of both normative form and function, placing it in the upper right corner of the grid, but some come very close, as in the following example.

```
*  "The futexes are also cursed."
*  "But they come in a choice of three flavours!"
(Linux Source Code, "futex.c").
```

The futex comment conveys functional information only in as much as the reader is intended to under-

stand that a futex is a complicated item that can be tricky to use. Instead, the reader is presented with arcane and ironic humor.

The Form/Function Grid is intended as a qualitative tool for visualization only. Few comments are ever devoid of normative information, and no comment is ever devoid of group identity, since the basic form is determined by the language, environment, and culture in which the comment is written. What seems purely normative in one setting seems highly identity-oriented in another, highlighting the wider community-related identity work performed by the comments and their structure, down to the presentation on the screen.

"Good" Comments

Looking directly at the usefulness of specific comments, one author asserts that "the availability of prose explanation of [an] algorithm will have a much larger influence on the speed with which the programmer understands the program than variations in the structure of the program" (Brooks, 200). Hence a "good" comment simply aids in the understanding of a program.

Where "good" comments begin to occupy other spaces is in the descriptive content. Discussing maintenance, one writer noted that a developer, "having exhausted the resources of local myth and legend, has no alternative but to actually read the code he is required to fix" (Deimel, 5). However, the comments often rely on just this local myth and legend as part of the explanation process, as the inclusion of project names in the comments indicates (as in the Corporate sample).

However, one writer acknowledges that comments, no matter how well-intentioned, can contain suspect information, saying, "The comments do not affect the meaning of the text source but they *may* help readers to discover the intended meaning" (Arab, 42, emphasis mine). Despite problems associated with comments, writers continue to believe in their importance.

The primary problem with comments, good or bad, is that information placed in a comment is not required to be maintained by any enforcement mechanism, unlike the code itself. With the symbolic logic of the code, a compiler enforces at least minimal syntactic correctness, and compiled code will generally be tested and used.

Since comments are ignored by the compiler and therefore cannot be tested by either the compiler or the users themselves, the only reason a comment will stay in synch with the logic is if the developers believe it is important to their own understanding. This problem of code and comment synchronization is a source of concern for theorists, based on the seemingly contradictory stances they take regarding the use of comments. One classic book on programming style asserts, "The only reliable documentation of a computer program is the code itself. […] Only by reading the code can the programmer know for sure what the program does" (Kernighan & Plauger, 141). However, the same authors go on to devote thirteen pages "to style in commenting" (Kernighan & Plauger, 141), further stating, "[…] An excellent program […] is thoroughly commented and neatly formatted" (Kernighan & Plauger, 150). Clearly, comments are a critical element of good programming.

Identity-Orientation

Of course, that programmers include comments at all is a social convention, as much a practice as it is information conveyance. Social conventions cover a range of purposes that far outweigh the official goals of comments within programs. Social conventions surround the very structure of the language in the comments, which is often used to further particular social or psychological goals, though these efforts are different in each sub-community of programmers.

One social goal is the establishment and maintenance of power, and providing inaccurate comments may effectively provide a developer with power. This power may be social and intangible or directly monetary, since ambiguity in the comments may "facilitate having a lucrative business of selling expensive service contracts and consulting services for the software" ("Documentation Definition").

Comments can also serve as conversations with the wider community. These conversations highlight areas where there is uncertainty about the code function (primarily in the Corporate sample) or where future fixes would ideally be made (found in both samples).

"We" Construction

While there is definitely evidence that comments are formed and have functions outside the normative practices of programming, there are far more subtle literary forms at work within the comments. Within the Linux kernel, there exists a consistent use of the subject "we" within the comments.

The "we" literary form found in the comments performs three functions. Firstly, the literary form allows elevation of the programmer (and his/her programming discipline) from technician to teacher, associating programming discourse with academic discourse. Secondly, the literary form helps maintain boundaries with outside individuals, organizations, and specifically users. Finally, the literary form expands the group identity to include not just the pro-

grammers, but the machines and software systems they create and manage as a cyborg community (see Haraway; and Downey, Dumit, & Williams). Notably, the Corporate sample contains very different language that occupies the space of group elevation, but nothing in the latter two categories.

In the Linux kernel, lines containing the "we" construction represent 12.92% of the total lines of comments. In contrast, the Corporate sample of just seven files contains only 36 lines with the "we" construction, representing just 0.57% of the total lines of comments (see Table 1).

Table 1: Comparison of Comment Dostribution

	Linux Sample:	**Corporate Sample:**
Total Files	52	7
Total Lines	41, 505	28,304
Lines with Comments	5711	6294
Precentage of Comments to Total Lines	13.75%	22.24%
Lines with "we" Construction	738	36
Percentage of total lines with "we"	1.78%	0.13%
Percentage of Comment lines with "we"	12.92%	0.57%

The statistics serve to support several assumptions. First, COBOL programs are often assumed to be much longer than C programs, though this is not a technical requirement. However, the Linux sample is not even one and a half (1.5) times larger than the Corporate sample is terms of lines of code, despite being more than seven (7) times larger in terms of the number of files. Second is an assumption that developers do not like to write comments or explain themselves, believing that good code will simply speak for itself. Despite having both a larger number of files and more lines of code, the Linux sample has 583 fewer lines with comments than the Corporate sample, hinting that developers may comment more when the activity is required. Third, the greater production life of the Corporate sample provides more opportunity for comments to be written. Fourth, the practice of commenting out obsolete lines of code in the Corporate sample results in a much higher percentage of comments to total lines, even though the resulting comments lack explanatory power and may contribute to overall confusion regarding the function of the particular program.

Most importantly, however, the differences between the two samples shed light on the different community values. The lower frequency of the "we" construction within the Corporate sample signifies less emphasis on group identity and boundaries and more emphasis on individual identity through sarcasm and other devices. Further, in the Corporate sample, the use of imperative verbs dominate, making the comments with a lecture-like tone take on a more hierarchical sense of an order being issued or perhaps complaints and diatribes against those not "following the rules." Even where the Linux and Corporate samples have similar identity-oriented purposes, the comments follow the pattern of group emphasis in the Linux sample and individualism and imperative verbs in the Corporate sample, standing somewhat in contrast with the relative stability of the corporate team over time.

Group Elevation

In some ways, group elevation is the most subtle but pervasive of the identity-oriented uses of comments. Comments serving this purpose are highly normative in function and yet identity-oriented in form, though often falling along the boundary between the two left quadrants of the form/function grid. The form of these comments gives weight and responsibility to the programmers, making them more than mere technicians.

```
/*
 * If we're in an interrupt or softirq, we're done
 * (this also catches softirq-disabled code). We will
 * actually run the softirq once we return from
 * the irq or softirq.
 *
 * Otherwise we wake up ksoftirqd to make sure we
 * schedule the softirq soon.
 */
(Linux Source Code, "softirq.c").
```

The form of the above example is that of a lecture in the sense of an academic situation, where the authority figure once typically used "we" extensively. I describe this as a grammar of justification, following Mulkay's "vocabularies of justification" (Mulkay, 637-656), where the grammar justifies elevation of programmer status to that of professor or leader.

Educational comments also appear throughout the Corporate sample, however, the use of the "we" construction is nearly absent. In one example, a comment provides developers with basic information about the structure of all COBOL programs, which is neither specific to the environment (cultural or technical) nor unique to the implementation or methodology; it simply educates later developers (Corporate Source Code, Program-7). This is not a "good" comment in the normative sense, but serves to reinforce the personal identity of the developer who is capable of and driven to provide such programming guidance.

While there is a lecturing or educational tone to many of the comments in the Linux sample, there are few, if any, outside references to other "knowledge domains" (see Brooks) or embedded programming style guides, as there are in the Corporate sample. This disparity between the samples is reflective of the different make-up of each community. Unlike in the Corporate sample, programmers in the Linux sample are expected to be proficient in order to be able to contribute, making overt style guides and programming recommendations embedded within comments unnecessary.

Boundaries

Importantly, the literary device of the "we" construction brings out the boundary work that programmers, like other disciplines, perform on a regular basis (Gieryn, 792-93). The comments in the Linux kernel highlight the tensions and trust issues between programmers and their users, at one point directly saying, "We only trust the superuser with rebooting the system" (Linux Souce Code, "sys.c"). This statement places general users on the outside and leaves out the implied assumption that a "superuser" is a highly technical user and very likely a programmer.

```
/* For safety, we require "magic" arguments. */
if (magic1 != LINUX_REBOOT_MAGIC1 ||
    (magic2 != LINUX_REBOOT_MAGIC2 &&
                magic2 != LINUX_REBOOT_MAGIC2A &&
                magic2 != LINUX_REBOOT_MAGIC2B &&
                magic2 != LINUX_REBOOT_MAGIC2C))
        return -EINVAL;
(Linux Source Code, "sys.c").
```

Note also that the entities needed to reboot the entire system are not "keys," or "control characters." Instead, these are "magic arguments," which constructs a position for programmers as magician or wizard, who only allow "superusers" access to the "magic arguments."

These examples of boundary work still provide information about the source code with which they appear, though it could be argued that the informational function is limited. What seems clear, however, is the identity oriented form in each, placing them in between the top two quadrants of the form/function grid.

Cyborg Community

In the final usage of the "we" construction, the boundaries maintained by the programmers are expanded to include the machine and the software the programmers create to run on it, creating a cyborg community. To be able to extend that boundary, it might be argued that the machine needs to be on the same plane as the programmer.

```
/* Some compilers disobey section attribute on statics when not
   initialized -- RR */
```
(Linux Source Code, "softirq.c").

The programmer in the above example notes that the compilers "disobey" in some situations. While the behavior of the compiler is dependent upon the instructions given to it (and how well it was programmed in the first place), the computer has been raised up to a level where the programmer can consider it to be within his/her boundary.

In terms of the form/function grid, the above comment serves a highly normative function, however, the form is highly identity-oriented, since the function is merely implied and the computer so heavily anthropomorphized.

However, the language of the comments moves from merely casting human attributes on the system to an association with the system, being most thoroughly realized in this example, which enhances cyborg identity:

```
/*
 * We're trying to get all the cpus to the average_load, so we don't
 * want to push ourselves above the average load, nor do we wish to
 * reduce the max loaded cpu below the average load, as either of these
 * actions would just result in more rebalancing later, and ping-pong
 * tasks around. Thus we look for the minimum possible imbalance.
 * Negative imbalances (*we* are more loaded than anyone else) will
 * be counted as no imbalance for these purposes -- we can't fix that
 * by pulling tasks to us.  Be careful of negative numbers as they'll
 * appear as very large values with unsigned longs.
 */
```
(Linux Source Code, "sched.c").

The literary "we" embedded in this paragraph is powerful in its linking of the author and his/her creation. In particular, notice the parenthetical statement in the third sentence, where "we" technically refers to a particular CPU, but the author clearly associates him/herself with that hardware and embeds him/herself in the software as though directing the function in real-time.

While Turkle positions her work as the opportunity to "see the computer as partner in a great diversity of relationships" (Turkle, 20), the "we" construction seems to speak to significantly different relation all together. In this case, rather than being a partner, the computer is part of the unit, creating the cyborg relationship, not as partner, but as part and constitutive element. Unlike Turkle's version of programmer anthropormorphizing, the Linux "we" construction does not encourage programmers to think of themselves "like" a machine, where the machine is a model for cognition. Rather, the "we" construction creates the Haraway-like cyborg model, where the programmer is actually part of the machine.

The conceptual plotting of this comment on the form/function grid also highlights an interesting phenomenon. The function is largely informative, however, the form, including the use of the "we" metaphor is highly identity-oriented, creating a tension between a form that serves identity-oriented goals and a function serving largely normative goals. The complexity of this last example forms a micro case study of how multiple uses and goals for comments can exist within the context of a particular technological frame (or set of frames), without taking any power away from the "official" normative goals.

Corporate Identity

Instead of establishing boundaries or cyborg identity, the Corporate sample seems to emphasize individual identity using mild humor. In one example, an unidentified developer concluded three different but related edits with the same tag line:

```
04931 * THAT CONCLUDES THE STRUCTURED COBOL PORTION OF THIS SECTION...
04932 * RETURN TO SPAGHETTI CODE!
04933      GO TO 3015-END-OF-CT-EDITS.
```
(Corporate Source Code, Program-1).

While the comments are all unsigned, we know the identity of the problem that caused the change, so within the day-to-day workings of the office, the identity of the developer would likely be well known.

The humor is also clearly evident, with fun poked at several areas, including the corporate direction on technology, the ability for IT to keep track of user requests for new capabilities, and the programming prowess of prior programmers.

Conclusion

Comments have an "official" or normative purpose—explain how the program works—and a close reading of various programs in radically different settings shows this to be a valid and "real" use. As has been written regarding standards, the basic tools of a discipline are perceived as "knowledge rather than practice" (Slaton & Abbate, 124). Comments are considered to include information about a program and even an organization or community. As the items above show, there is indeed information (knowledge), but the comments are clearly a practice and a performance.

However, that same reading also reveals much more. At the most fundamental level, the structure of comments is strongly influenced by the programming language, the technical environment, and the culture in which they are written. More importantly, while clearly performing normative functions, the form allows constructions that help elevate the programming community, establish and defend community boundaries, stretch the definition of programming community to become a cyborg community, including the systems the programmers create and the machines on which they run, and help individuals establish their own identity.

One specific lesson from the comments is that very specific types of community will be created by and reflected in the discussions and interactions in the source code comments. As Sherry Turkle writes, "A computer program is a reflection of its programmer's mind" (Turkle, 24), and the comments will occupy a similar mental space. The greater emphasis in the Linux sample on the "we" construction reflects a more collegial atmosphere, highlighting the voluntary nature of the project, which relies on the good will of the collected organization. The Corporate sample is more combative and directive in tone and structure, reflecting the hierarchical structure of the organizations in which they are constructed.

Importantly, what programmers establish and then reflect through these comments might be called "ownership" or "connection" with the product of the programming endeavor. Speaking of toys, Sherry Turkle says, "When people are asked to care for a computational creature and it thrives under their ministrations, they become attached, feel connection, and sometimes much more" (Turkle, 290), essentially describing how programmers relate to their work. Following the process of anthropomorphizing, the machines and systems used and built by the programmers become creatures that are nurtured and ministered to by the programmers. In some cases, the creatures are given life by the programmers. This is a feeling common to all programming, not just open source. Any significant amount of time spent caring for a program, or set of programs results in attachment and connection.

It would be tempting to assume that open source programmers have a greater degree of connection to their projects, but the writing of the programmers in their comments does not seem to support this, with, if anything, more passion expressed by the Corporate programmers than by the Linux programmers. The connection is likely the result of longevity, since both sets of code are relatively old (though the Corporate sample is older), but also specialization. Particular programmers specialize, either by choice, as in the Linux sample, or by a combination of choice and management directive, as in the Corporate example. This helps reinforce the sense of ownership inherent in programming, since, as Sherry Turkle says, "a large computer system is a complicated thing", leaving "plenty of room for territoriality" (Turkle, 198). This territoriality is alternatively to be seen as attachment and connection to the creatures spawned by the care given to the machine.

While some writers have attempted to view comments as explanatory only for the text at hand and to view the texts simply as programs to control a machine, I conclude that comments are a critical resource in the establishment and/or reinforcement of identity on both a group and personal level, through their role in the executable texts. The identity-oriented purposes of executable texts can only be removed from texts when "the theory is unconcerned with personality characteristics of programmers, with the effects of varying motivational conditions or with social interaction in programming groups" (Brooks, 200), which is a less-than-useful way to approach either a social group or a form of communication.

References

Bijker, Wiebe, E. "The Social Construction of Bakelite: Toward a Theory of Invention" in Bijker, W. E., T.P Hughes, and T.J, Pinch (eds.), *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology*, Cambridge, MA: MIT Press, 1987, pp 159-187.

Brooks, Ruven. "Using a Behavioral Theory of Program Comprehension." *Proceedings of the 3rd International Conference on Software Engineering*. IEEE Press, May 1978.

"Documentation Definition." unsigned. *Linux Information Project*, 23 February 2006; from: http://www.bellevuelinux.org/documentation.html, 2006/03/01.

Downey, Gary, Joseph Dumit, & Sarah Williams. "Cyborg Anthropology," in *Cultural Anthropology* 10(2) (1995): p. 264-269.

Gieryn, Thomas. "Boundary-Work and the Demarcation of Science from Non-Science: Strains and Interests in Professional Ideologies of Scientists" in *American Sociological Review* 48, pp. 781-795, 1983.

Haraway, Donna Jeane. *Simians, Cyborgs, and Women: The Reinvention of Nature*. New York: Routledge, 1991.

Jones, Duncan Edwards. "The seven secrets of successful programmers." The Code Project; from: http://www.codeproject.com/tips/7secrets.asp, 2006/03/01.

Kernighan, Brian W. & P.J. Plauger. *The Elements of Programming Style*, 2nd Ed. McGraw-Hill, 1978.

Mulkay, Michael J. "Norms and ideology in science," in *Sociology of Science* 15 (4/5), pp. 637-656, 1976.

Slaton, Amy and Janet Abbate. "The Hidden Lives of Standards: Technical Prescriptions and the Transformation of Work in America". In *Technologies of Power: Essays in Honor of Thomas Parke Hughes and Agatha Chipley Hughes*. Michael Thad Allen and Gabrielle Hecht, eds. Cambridge, MA: MIT Press, 2001.

Turkle, Sherry. *The Second Self: Computers and the Human Spirit (Twentieth Anniversary Edition)*. Cambridge, MA: MIT Press, 2005.

Weinberg, Gerald M. *The Psychology of Computer Programming*. Litton Educational Publishing, Inc., 1971.

About the Author

Stuart Mawler

Stuart is an IT Consultant, bringing more than 15 years of IT experience, primarily as a software architect, to his research interests in programming culture and the Sociology of Technology.

The International JOURNAL of TECHNOLOGY, KNOWLEDGE & SOCIETY

COMMON GROUND

Please visit the Journal website at http://www.Technology-Journal.com
for further information about the Journal or to subscribe.

THE UNIVERSITY PRESS JOURNALS

International Journal of the Arts in Society
Creates a space for dialogue on innovative theories and practices in the arts, and their inter-relationships with society.
ISSN: 1833-1866
http://www.Arts-Journal.com

International Journal of the Book
Explores the past, present and future of books, publishing, libraries, information, literacy and learning in the information society. ISSN: 1447-9567
http://www.Book-Journal.com

Design Principles and Practices: An International Journal
Examines the meaning and purpose of 'design' while also speaking in grounded ways about the task of design and the use of designed artefacts and processes. ISSN: 1833-1874
http://www.Design-Journal.com

International Journal of Diversity in Organisations, Communities and Nations
Provides a forum for discussion and builds a body of knowledge on the forms and dynamics of difference and diversity.
ISSN: 1447-9583
http://www.Diversity-Journal.com

International Journal of Environmental, Cultural, Economic and Social Sustainability
Draws from the various fields and perspectives through which we can address fundamental questions of sustainability.
ISSN: 1832-2077
http://www.Sustainability-Journal.com

Global Studies Journal
Maps and interprets new trends and patterns in globalization. ISSN 1835-4432
http://www.GlobalStudiesJournal.com

International Journal of the Humanities
Discusses the role of the humanities in contemplating the future and the human, in an era otherwise dominated by scientific, technical and economic rationalisms. ISSN: 1447-9559
http://www.Humanities-Journal.com

International Journal of the Inclusive Museum
Addresses the key question: How can the institution of the museum become more inclusive? ISSN 1835-2014
http://www.Museum-Journal.com

International Journal of Interdisciplinary Social Sciences
Discusses disciplinary and interdisciplinary approaches to knowledge creation within and across the various social sciences and between the social, natural and applied sciences.
ISSN: 1833-1882
http://www.Socialsciences-Journal.com

International Journal of Knowledge, Culture and Change Management
Creates a space for discussion of the nature and future of organisations, in all their forms and manifestations.
ISSN: 1447-9575
http://www.Management-Journal.com

International Journal of Learning
Sets out to foster inquiry, invite dialogue and build a body of knowledge on the nature and future of learning.
ISSN: 1447-9540
http://www.Learning-Journal.com

International Journal of Technology, Knowledge and Society
Focuses on a range of critically important themes in the various fields that address the complex and subtle relationships between technology, knowledge and society. ISSN: 1832-3669
http://www.Technology-Journal.com

Journal of the World Universities Forum
Explores the meaning and purpose of the academy in times of striking social transformation.
ISSN 1835-2030
http://www.Universities-Journal.com

FOR SUBSCRIPTION INFORMATION, PLEASE CONTACT
subscriptions@commonground.com.au