

Epistemologically Multiple Actor-Centered Systems: or, EMACS at work!

*History shows how communal sharing and problem-solving
strengthen software functionality and innovation.*

By Yuwei Lin

Actors from different backgrounds contribute multiple ways of knowing, understanding and resolving problems that arise in the innovation process. This paper employs a socio-technical perspective to analyze how editing macros are shaped by diverse actors, and at the same time also shape these actors and their practices.

Introduction

The paper begins with the story of EMACS (short for Editing MACroS), an editor program originally written for TECO (Text Editor and Corrector) language and PDP-10 machines in the MIT AI Lab by Richard Stallman, from which various more sophisticated versions have been developed. I analyze how the innovation of EMACS took place over time as a socio-technical process. The EMACS story serves to illustrate how the innovation process in the FLOSS (Free/Libre Open Source Software) community occurred, but one that is then adopted and deployed in other social contexts, including the commercial sector. The analysis of EMACS is especially useful since it spans the period that witnessed the origin of the free software movement and the subsequent development of a broader FLOSS social world.

EMACS

In 1976 Richard Stallman, an employee at MIT AI Lab, and his colleagues, wrote the editor EMACS to upgrade the previous editor TECO on an ITS (Incompatible Time-Sharing System), which was the software running on the AI Lab's Digital PDP-10 minicomputer. In the text-based pre-graphical world that existed before the Apple Macintosh and Microsoft Windows, the editor was a program crucially important for creating and manipulating text (Moody 2001: 16). Instead of typing commands when editing texts, the TECO editor enabled users to employ macros, command strings for a cluster of TECO programs, which provided a more immediate onscreen feedback for users.

TECO already had the WYSIWYG (What You See Is What You Get) feature named Control-R, written by Carl Mikkelsen, which allowed users to enter macros (command strings) and discarded them after entering them. Borrowing the idea from another WYSIWYG editor named "E", Stallman then brought additional functionality to TECO to make it possible to save macro shortcuts on file and call them up at will. It is said that this improvement was subtle but significant in that this raised TECO to the level of a user-programmable WYSIWYG editor, which later on enabled innovation at another meta level that became the progenitor of FLOSS (Williams 2002: 82).

The amended macro function in TECO permitted users to redefine their own screen-editor commands, pass them around and improve them, and make them more powerful and more general. The collections of user-redefinitions gradually became system programs in their own right (ibid.). In so doing, users extended the original TECO system

by adding or replacing functions based on their self-defined definitions of 'the problem'. Users were not, then, limited by the decisions made or problem-solving approaches taken by the original innovators (Stallman 1998: 2). The extensibility made TECO more flexible for use and in turn attracted a larger number of users to incorporate the macro function into their TECO programs.

However, a new problem emerged along with this new feature. While the new full-screen capabilities were embraced vigorously, various customized visions of TECO also led to over-complexity. The most obvious example was that one had to spend more time than before figuring out what macro commands did what in terms of an individual's self-definition of "the problem" when improving each other's work. Guy Steele, a colleague of Stallman's at the AI Lab, recognized this problem and sought to address it. He first gathered together four different macro packages and began assembling a chart that he believed identified and organized the most useful macro commands (Williams 2002: 83). In the course of implementing the design specified by the chart, Steele's work attracted Stallman's attention and led to the latter's participation in this renovation project.

Together with David Moon and Dan Weinreib, the four tried on the one hand, to develop a standard system of macro commands, and on the other hand, to still keep the command set open-ended to enable ongoing programmability/extensibility by others. The program was named EMACS.

The distribution of EMACS marked another milestone in the software history. In response to the prevalence and technical opacity associated with the practice of entirely self-defined commands, and to endorse the hacker tenet of sharing information, Stallman set the terms on which EMACS could be used in the statement linked to the source code when distributing the editor.

EMACS, as noted in Stallman's biography, served as a social contract that rendered communal sharing the basis of its distribution. Users, on the one hand, were able to modify and redistribute the code; on the other hand, they were asked to report back the extensions they might have made to Stallman so that he could incorporate and distribute them. In so doing, Stallman strengthened the functionality of EMACS, making programming with macros more standardized through creating a reciprocal understanding of the written codes through sharing problem solutions, as well keeping the extensibility that macros afforded.

Consequently, a library was created for users to load new or redefined functions and to publish and share their extensions. "By this route, many people can contribute to the development of the system, for the most part without interfering with each other. This has led the EMACS system to become more powerful than any previous editor." (Stallman 1998: 2). Since then, an archetype of EMACS had been established.

Affordance and EMACS: Extensibility and Customization

The earlier generation of EMACS had been successful because it provided flexible use with an embedded programming language, TECO. Because of this feature, editing commands could be written in that programming language and users could load new commands into their editors while editing. EMACS resembled a system that was useful for things other than programming, and yet one could program it while using it. It was claimed to be the first editor that could operate in this way (Stallman 2002: 1).

Consequently, the socio-technical networks of EMACS have been expanded. GNU Emacs for example, is available for Unix, VMS, GNU/Linux, FreeBSD, NetBSD, OpenBSD, MS Windows, MS-DOS, and other systems. GNU Emacs has been re-configured more than 30 times as part of other systems. Other variants include GOSMACS, CCA Emacs, UniPress Emacs, Montgomery Emacs, and XEmacs. Jove, Epsilon, and MicroEmacs are limited lookalikes. These systems have additional requirements, needs, and visions that differ from the original GNU Emacs. This phenomenon widens the range of what we might see as "digital epistemologies" (ways of ordering and knowing software) and their expression through software artifacts in the FLOSS social world.

Problems and solutions: the sine qua non of the software innovation process

In light of the EMACS account above, problems play a crucial role in the innovation process. Triggering a problem or perceiving a problem and dealing with it are important tasks for scientists and engineers, and through their resolution help generate innovation. A problem thus can be seen as the inauguration of an innovation. The confronting of a problem provides an opportunity of coming up with something new or different. A problem de facto denotes one's perception of the situation, one's knowledge, and skills. Accordingly a problem signifies one's identity as an expert or a novice, for example.

Efficiency was an important parameter in the process of defining software problems. A problem would not exist if users did not recognize the existing practice (e.g. composing code with printing terminal editors in the 70s) as inefficient. As efficiency is regarded as key within the field of engineering; engineers were and are still taught to design efficient technologies.

Efficiency is not however, self-evident: Stallman reports that he did not have a strong sense of the need for a real-time display editor until he encountered the "E" program when he visited the Stanford Artificial Intelligence Lab in 1976. He was inspired by the function E afforded and sought to expand TECO's functionality in the same way and helped form the group that was to work on a real-time display system. Meanwhile, there were other parallel groups providing solutions for real-time display systems, and their work could become complementary to the work that Stallman's group was undertaking. Stallman's macro improvement to TECO enhanced Mikkelson's earlier WYSIWYG feature for TECO. As a result, with this greater affordance and functionality, TECO became more popular. The TECO socio-technical network expanded when more people accepted the macro innovations and incorporated them into their own versions of the TECO program.

It is worth noting that Stallman did not sit down and write the editor system program immediately after his encounter with E. Instead, he looked up the database and found that Mikkelson had made a WYSIWYG feature for TECO. He then integrated his idea into that. If Mikkelson's work had not existed, we may have seen a different technical option taken, as the "problem" may have been defined differently. This points to the contingency of the innovation process. This process is reflected in many FLOSS developers' own reflections on their systems as seen in Stallman's and Torvald's biographies (e.g. Torvald said he probably would not have started the Linux kernel project if the GNU Hurd had already existed; Stallman said he would not have started to write the GCC compiler if Tanenbaum had agreed to share his work).

Hence, looking for existing material and tools is another common practice in solving problems in software innovation. Software engineering, as in other fields, is built on existing technologies. Programmers typically explore existing databases and see whether any tool is available; if not, they are likely to try to create one to solve the problem.

Access to problems is another issue that needs to be discussed in light of the data from my fieldwork. The accessibility of problems measures the relative ease with which problems can be understood. If one problem entails an opportunity for innovation, an active innovation field should welcome more problems. In a less open innovation system, problems are less accessible, and the boundary is relatively impermeable to new entrants and new ways through which the system can be enhanced. In such an innovation system, problems are less likely to be seen, innovation options likely to be more pre-defined, and innovators more likely to share a consensus on "what needs to be done".

On the contrary, I argue that in a heterogeneous field where diverse actors are found, more problems arise or are triggered. If the boundary of the social group centering on the problem is soft, more diverse actors will be included in the circle. There is a positive correlation between the elasticity of the boundary of an innovation field and the momentum behind the pace of innovation because the more accessible the problem is, the higher the level of multi-vocality existing in the innovation system.

The elasticity of the boundary can be manipulated through a range of educational, legal, political, economic, social and technical means. In the 1970s, software problems were more accessible in the sense that fewer regulations were applied to restrict programmers to access key materials (codes peculiarly). There was a so-called "collaborative hacker" approach, sharing knowledge and improving each other's work, that encouraged

programmers to continually redefine the boundaries of the problem (e.g. conducting reverse engineering to deconstruct a software to understand how a code was written). As a result, a wide range of software programming tools (languages, editors, compilers etc.) was created in the 1970s (Ceruzzi, 2003).

As noted, while Stallman's innovation was celebrated because of its extensibility and flexibility, it did however create new problems. One problem was the sense of growing confusion derived from the plethora of self-defined macro commands, which was seen to eventually work against a more efficient process of programming. In response, another member of the AI group, Steele, tried to provide a solution by charting the macros. Steele's solution encouraged Stallman, Moon and Weinreid to participate in a solution-crafting innovation group. These four who shared the same interest in this project formed a social network of expertise and began to fashion the digital tools that would be seen to provide the solution they were looking for: in this sense the path they took and tools they developed reflected a shared conceptual frame that was "not accidental, but constitutive." (Clarke, 1998; Bowker & Star, 1999: 36).

What made the process more intriguing is the relationships and interactions between the actors themselves and the actants (materials), and the transitions -- the boundary crossings -- that a problem so identified enabled. Boundary crossings can require both getting in (e.g. gaining access to the problems) and getting out (e.g. forging alternative solutions different from the original one). These boundaries are interpreted or constructed through the problem-solving process of software design. In the process, actors move across boundaries and shift their identities as outsiders or insiders to the core innovation group.

Shared interests and translation of interests

As seen in the story of EMACS, engaging actors in a network is the key to effective innovation in that the range of expertise in the network will affect a group's abilities to solve problems. Since everyone is an expert with regard to some things and a novice with regard to others, a problem/question in an open environment will be answered sooner or later. It is worth noting, however, actors' involvement in a network is not randomly assembled, but determined by shared interests. As Latour articulates,

"The first and easiest way to find people who will immediately believe the statement, invest in the project, or buy the prototype is to tailor the object in such a way that it caters to these people's explicit interests. As the name 'inter-esse' indicates, 'interests' are what lie in between actors and their goals, thus creating a tension that will make actors select only what, in their own eyes, helps them reach these goals amongst many possibilities." (Latour 1987: 108-9).

After Stallman left the MIT AI Lab and planned to write a Unix-like operating system for non-commercial distribution, a range of ICTs facilitated his individual action. Stallman's GNU project, as in many other FLOSS projects, precisely took advantage of ICTs to target peers to join his innovation network. These artifacts enabled him to maintain communication with other programmers and even helped to secure some innovation resources when he worked alone. On 27 September 1983, Stallman posted the message below onto the Usenet newsgroup net.unix-wizards in order to invite people who shared the same interest or parallel knowledge to join the discussion. Stallman posted the following message:

"Starting this Thanksgiving I am going to write a complete Unix-compatible software system called GNU (for GNU's Not Unix), and give it away free to everyone who can use it. Contributions of time, money, programs and equipment are greatly needed."

(Williams 2002, ch. 7: 1)

In this message, Stallman revealed his defined problem and the proposition for possible solutions -- a complete Unix-compatible software system. Because he had lost institutional support (financial and intellectual) from the MIT AI Lab, he was more likely to find peers interested in joining the social network of developing a new operating system and to engage their attention by posting messages onto the Unix newsgroup, where specific users/programmers share the same interest inhabit.

Without knowing who was going to get in touch with him, the message posted entailed uncertainty and risk in Stallman's project. When Stallman posted this message, he created a social network of crafting a new operating system. If Stallman was able to invite many programmers to join this project, the network would grow and the project would take off. Here, making decisions of which listserv or newsgroup a message should be posted to in order to attract as many actors as possible is another form of classification shaping the innovation process. Stallman reckoned the Unix group was the one where he would be most likely to find his target peers.

This tendency of looking for peers who share the same interests echoes my previous argument that a shared interest among peers is crucial for the continuation of collaboration. The common interests engage actors to work together, share knowledge and exchange information. The teamwork gets more complex with higher peer participation. If the management style stays in a democratic/open way, the boundary of

the team will remain soft. This type of innovation is more accessible because open debate within the team is more likely to produce multiple topics to attract actors. Here, the construction of a shared interest or a common topic is resonant with Latour's notion of inter-essant (1987), through which actors are enrolled to mobilize innovation networks.

The working environment at MIT AI Lab in the '70s was similar to this way. Most of the FLOSS projects that rely on virtual collaboration also meet the criteria of Latourian inter-esse/sant. In contrast, a closed or centralized direction of a project would reinforce innovation boundaries and restrict accessibility to the innovation process. In so doing, a project can be kept under control to eliminate risks and uncertainties generated by multi-vocality. Following this route, a project will approach closure eventually. Proprietary software is mostly managed in this way.

EMACS as a Boundary Object

Hitherto, I have investigated how a software innovation network is created and developed in terms of classifications of problems, identifications of solutions and common interests. Actors and actants are brought together, interact and negotiate with each other to solve problems. Boundaries of networks, which determine access to innovation systems, vary in different contexts. A successful innovation, as discussed, is the one that manages to bring in as many actors and actants as possible by translating their interests to extend and mobilise the network as well as handle the uncertainties and risks emerging during the process.

In reviewing the innovation story of EMACS, a variety of EMACS have been innovated/renovated by diverse actors for different purposes. The functions of EMACS

have been expanded and are still expanding. In other words, the affordance of EMACS is sustained. While diverse innovation repertoires are brought into the social network to tackle a joint problem, they also complicate the situation by defining and redefining EMACS' innovation concept over and over again. If different definitions of innovation concepts cannot be reconciled, a project diverges, as the development of many versions of EMACS show.

The reasons for the divergences vary in

-- social scope (e.g. disagreement on Stallman's social contract)

-- technical/material scope (e.g. original EMACS did not run on programming languages other than TECO, so new versions of EMACS were designed for other programming languages such as Lisp, EINE and ZWEI, developed by Weinreb, a fellow colleague working on the original version of EMACS)

-- or other contingent factors (e.g. experimental projects -- just for fun, perhaps).

These parallel processes demonstrate the dynamic in the innovation network of EMACS. Facing the challenge of heterogeneity, authors and maintainers of EMACS try to enhance their legitimacy and uniqueness in providing greater socio-technical functions.

Since EMACS are used in many different ways and denote various socio-cultural meanings, diverse projects have symbolized users' habits and preferences (socially and technically). In adopting specific tools and participating in specific projects, users are attached to the artifacts. These artifacts grow to be norms to demarcate boundaries. For

example, the users of GNU Emacs see themselves as different from those of XEmacs, while the broad range of users of Emacs distinguish themselves from users of other editor programs, such as vi. Holy wars happen often because these users want to fortify their boundaries against each other to show their identities. Accordingly, software programs containing symbolic contents should be seen not merely as algorithm codes but also as social codes. It would be interesting to see how these projects are symbolized as norms, how they are interpreted and used. That is, in the course of explaining the heterogeneous FLOSS social world, one can study the socio-technical meanings given to various projects to understand “FLOSS”. This actor-centered view may provide a distinctive research result from the prevailing structure-centered or essentialist approaches in the FLOSS studies.

Life in EMACS will continue to change, but the crucial presumption will be whether the boundary of the innovation network can be maintained to sustain the innovation. For instance, Gosling did not continue to share his codes, rather, he sold his EMACS version to a commercial company. His version of EMACS therefore did not continue to grow. This is not to say that selling software to a commercial company will kill the product. There are other reasons for the elimination of a software product and sometimes the involvement of commercial companies does provide other sources for sustaining or developing a product. But in the case of Gosling Emacs: 1) He thought selling the product to a firm might broaden the network. But the transaction symbolized Gosling's failure to continue the network expansion. 2) The commercial product was not successful. The firm failed to engage actors' interest in it.

Commercialized software forms a firmer boundary than FLOSS community projects to exclude outsiders of the developing team from the innovation. In that case, problems will

not be triggered that easily. If problems are the initiatives of innovation, a less diverse character in the team will not help the innovation at that point. When GNU Emacs was invented and still in an unstable stage, it did not put users off, instead, the existing problems invited peers to tackle them. GNU Emacs was able to engage actors in its social network of innovation. The network expanded and a variety of functions were developed. These functions become cornerstones to attract more actors/users in an ever-increasing snowball effect.

Unlike some proprietary software firms that try to lock users in by using proprietary document formats (and critics say they do this in order to dominate the market), EMACS engages users by presenting greater shared interests in socio-cultural or technical aspects. The story of EMACS sheds some light on FLOSS innovation, albeit the commercial impetus should be taken into account in order to understand the situation completely.

References

- Borgman, C. L. 2003. "Designing Digital Libraries for Usability". In Ann P. Bishop, Nancy A. Van House & Barbara P. Battenfield (Eds) *Digital Library Use: Social Practice in Design and Evaluation*. London: The MIT Press.
- Bowker G. C. & Star, S. L. 1999. *Sorting Things Out: Classification and its Consequences*. London: The MIT Press.
- Ceruzzi, P. 2003. *A History of Modern Computing. 2nd edition*. The MIT press.
- Debian Documentation Team. 2003. *A Brief History of Debian*. URL (consulted 27 November 2003): <http://www.debian.org/doc/manuals/project-history/project-history.en.txt>
- Glass, A. L., K. J. Holyoak, and J. L. Santa. 1979. *Cognition*. Reading, MA: Addison-Wesley.
- Latour, B. 1987. *Science in Action: How to Follow Scientists and Engineers Through Society*. Cambridge, MA: Harvard University Press.

- Latour, B. 1993a. "On Technical Mediation: The Messenger Lectures on the Evolution of Civilization".
Cornell University, Institute of Economic Research: Working Papers Series.
- Latour, B. 1993b. *We Have Never Been Modern*. Hemel Hempstead: Harvester Wheatsheaf.
- Latour, B. 1996. *Aramis, or the Love of Technology*. Cambridge, MA: Harvard University Press.
- Lave, J. & Wenger, E. 1992. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press.
- Lave, J. 1988. *Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life*. Cambridge: Cambridge University Press.
- Levy, S. 1984. *Hackers: Heroes of the Computer Revolution*. Garden City NY: Anchor Press/Doubleday.
- Liff, S. and Steward, F. 2003. "Shaping e-access in the cybercafe: networks, boundaries and heterotopian innovation." *New Media & Society*, vol 5(3): 313-334.
- McKelvey, M. 2001. "Internet Entrepreneurship: Linux and the dynamics of open source software", CRIC Discussion Paper no. 44. Centre for Research on Innovation and Competition, The University of Manchester & UMIST.
- Moody, G. 2002. *Rebel Code: Linux and the Open Source Revolution*. London: Penguin Books.
- Reitman, W. 1964. Heuristic Decision Procedures, Open Constraints, and the Structure of Ill-Defined Problems. In M. W. Shelley and G. L. Bryan, (eds.), *Human Judgment and Optimality*. New York: Wiley.
- Simon, H. 1973. The Structures of Ill Structured Problems. *Artificial Intelligence*, 4, 181-201.
- Stallman, R. "The Emacs Full-Screen Editor". URL (consulted 27 November 2003):
<http://www.lysator.liu.se/history/garb/txt/87-1-emacs.txt>
- Stallman, R. 1995. *GNU Emacs Manual. 11th Edition*, Version 19.29. Boston: Free Software Foundation.
- Stallman, R. 1998. "EMACS: The Extensible, Customizable Display Editor", 11 February, URL (consulted November 2003): <http://www.gnu.org/software/emacs/emacs-paper.html>
- Stallman, R. 2002. "My Lisp Experiences and the Development of GNU Emacs" (transcript of Richard Stallman's Speech, 28 Oct 2002, at the International Lisp Conference). URL (consulted December 2003): <http://www.gnu.org/gnu/rms-lisp.html>
- Star, S. L., Bowker, G. C., Neumann L. J. 2003. "Transparency beyond the Individual Level of Scale: Convergence between Information Artifacts and Communities of Practice", in the book *Digital library Use: Social Practice in Design and Evaluation*, edited by Ann Peterson Bishop, Nancy A. Van House, and Barbara P. Buttenfield. The MIT press.

Williams, S. 2002. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. Sebastopol, CA: O'Reilly.

Zawinski, J. 2003. *Emacs Timeline*. URL (consulted 16 December 2003): <http://www.jwz.org/doc/emacs-timeline.html>

Yuwei Lin, Taiwanese, is a PhD Candidate based at the Science and Technology Studies Unit at the University of York, UK. Her research involves free/liber open source software (FLOSS) studies, hacker culture, and science and technologies studies (STS). More details about Yuwei can be found at <http://www-users.york.ac.uk/~yl107>

*Source: Ubiquity, Volume 5, Issue 1, Feb. 25 - Mar 2, 2004,
<http://www.acm.org/ubiquity/>*