A paper presented at the 3rd Oekonux Conference, 20-23 May 2004, Vienna, Austria

Epistemologically **M**ultiple **A**ctor-**C**entred **S**ystem: or, **EMACS** at work!

Yuwei Lin

Copyright (c) 2004 Yuwei Lin
This paper is GFDL-ed. All rights reserved world-wide.

YL107 {at} york dot ac dot uk
Science and Technology Studies Unit
Department of Sociology
University of York, UK

Acknowledgement

My heartfelt thank goes to Professor Andrew Webster for his insightful comment and helpful proof-reading on the first version of this paper. Many thanks also go to the friends in the Debian community and the audience at the 3rd Oekonux conference for their constructive suggestions and encouragements.

Abstract

The paper begins with the story of EMACS (short for Editing MACroS), an editor programme 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 analyse 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 saw the origins of the free software movement and the subsequent development of a broader FLOSS social world. I will talk about how a variety of EMACSen (the plural form of EMACS) (e.g. GNU Emacs, XEmacs, MulticsEmacs etc.) are created, developed and employed/deployed in mundane programming

within an actor-centred network. Actors from different backgrounds contribute multiple ways of knowing, understanding and resolving problems that arise in the innovation process. A socio-technical perspective is employed to analyse how EMACSen are shaped by diverse actors, and at the same time also shape these actors and their practices.

To widen the scope of the paper in terms of its implication in a wider societal dimension, anchored in sociology of intellectual/knowledge, this paper also contributes to our understanding of the formation of knowledge in the Internet era, where information and knowledge flow fluidly and rapidly. The EMACS case denotes various key factors of forming cosmopolitan knowledge: how actors network together (e.g. shared interests), how they interact with one another (e.g. problem-solving process), and how local epistemologies and tacit knowledge being translated into cosmopolitan expertise in an in/tangible form (e.g. materiality of hardware or software). I believe this empirical enquiry will provide us with a means of retaining the holistic and meaningful characteristics of real-life events. Methodologically speaking, the contextual thickness makes a case study appropriate for "how" and "why" research questions because answering these questions deals with operational links needing to be traced over time. The detailed investigation of FLOSS phenomenon with attention to its context by using multiple sources of evidence and various methods of data collection will help to examine the innovation process by which new FLOSS technologies are created, arguing that this is ongoing and involves diverse groups who give the technology different meanings. This perspective also reflects an ongoing thinking in science and technology studies (STS) that technologies, no matter their designs, uses or applications, are not independent from social factors. Given the *history* of EMACS, one can see how the hacker ethics are emerged, developed, and followed in the innovation process. Based on the hacker ethics, EMACS, or a wide range of FLOSS, are not only a technological revolution, but also a social movement that operates largely in terms of symbol and meaning, both at the level of everyday life and at that of institutional operation.

1. Introduction

The paper begins with the story of EMACS, an editor programme 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 analyse how the innovation of EMACS took place over time as a socio-technical process. There are a number of reasons for why EMACS provides a valuable illustration of the heterogeneous and contingent FLOSS innovation system. Firstly, EMACS signifies the commencement of the General Public License (GPL), one of the most important licences upholding FLOSS innovation. EMACS's connection with GPL dated back to the TECO EMACS written by Stallman and colleagues at the MIT AI Lab in the 70s. It had been

designed to be used and developed via an explicit social contract – a sort of 'innovation contract' - between users and developers for their mutual benefits, enabling those involved to notify each other about the modifications they proposed to elements of the system. Secondly, (GNU) EMACS had been in use at more than a hundred sites (Stallman 1998: 16) and had a number of related editor software in place. Hence, EMACS will, on the one hand, illustrate the emerging innovation process from invention, through implementation, to diffusion, and on the other hand, show at work the complex web of classification and standardisation that began to shape and define the system as it became established and embedded in the wider FLOSS context. It will also allow us to explore the affordances of EMACS and how this diverse potentiality is exploited in different ways such as gaming, web browsing, editing, compiling and testing programmes, to name just a few. Furthermore, EMACS, an editor programme that can edit source code for crafting other programmes, functions as one of the essential artefacts in mundane programming. Studying EMACS therefore provides a rich site through which we can investigate relationships and interactions between actants (e.g. software and code) and actors (e.g. users and developers) in the FLOSS innovation system.

The following account of the EMACS innovation process will illustrate how problems were defined, redefined and solved by a variety of actors interacting with artefacts to develop a software product/project. But I also show how the establishing of EMACS turned out to have quite unexpected outcomes in the longer term.

2. EMACS (Background Information)

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 mini-computer. In the text-based pre-graphical world that existed before the Apple Macintosh and Microsoft Windows, (and yet still in today's Unix world), the editor was a programme 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 programmes, which provided a more immediate onscreen feedback for users. TECO had already had the 'WYSIWYG' (What You See Is What You Get) feature named Control-R, written by Carl Mikkelson, 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, make them more powerful and more general, and then the collections of user-redefinitions gradually became system programmes 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 programmes.

However, a new problem emerged along with this new feature. While the new full-screen capabilities were embraced vigorously, various customised 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, recognised this problem and sought to address it. He firstly gathered together four different macro packages and began assembling a chart that he believed identified and organised 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 programme 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 of 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 those again. In so doing, Stallman strengthened the functionality of EMACS, making programming with macros more standardised through creating a reciprocal understanding of the written source code 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.

3. Affordance and EMACS: Extensibility and Customisation

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 her/his editor while s/he was editing. EMACS resembled a system that was useful for things other than programming, and yet one could program it while s/he was using it. It was claimed to be the first editor that could operate in this way (Stallman 2002: 1). Parallel EMACS-like editors were written for other programming languages or for different machines in the few years following its first release in 1976. For example, EINE (EINE Is Not EMACS) was written in Lisp language (the first editor written in Lisp) in 1976, the same year EMACS was released; Multics Emacs was written in MacLisp language in 1978; Gosling Emacs was written in C language in 1981; Hemlock was written in Spice Lisp language in 1983. These various developments inspired Stallman to write a new version of EMACS in 1985, named GNU EMACS, a part of his GNU project, meant to be a clone UNIX operating system that would be distributed for free.

GNU EMACS, XEMACS (formerly known as Lucid EMACS) and their sort are well regarded today in the social world of FLOSS or in the wider software world. This popularity does not come overnight. The wider development of EMACS has undergone a process of socio-technical construction. The socio-technical network of EMACS has expanded mainly because of the affordance it allows. The features of customisation and extensibility have provided mutual benefits to both users and developers. For example, the extensibility of EMACS enables one to go beyond simple customisation and write entirely new commands for programs in the Lisp language to be run by EMACS' own Lisp interpreter. In so doing, once the new commands are written, they will be stored in the library so that all can access to it and implement it.

Consequently, the socio-technical networks of EMACSen have been expanded. GNU Emacs for example, is now 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 look-alikes. These systems on the other hand also have 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 artefacts in the FLOSS social world.

The pattern of affordances linked to EMACS software is not, however, the only factor affecting its social network. There are some specific features of the system that appear important in explaining the

popularity of EMACS. EMACS appears to be a system whose material features enable it to have this 'close to hand' quality for users. Problems and the way they are ordered, managed and classified are not to be defined independently from a specific time or locale, rather, they are defined through, to paraphrase Bowker and Star (1999), 'a process of assembling materials close to hand and using them with others in specific contexts.' (Bowker & Star 1999: 288; see also Lave 1988). Hence, users' preference for EMACS reflects the way in which users see it providing specific types of answers to specific types of problems or needs that they have, as shown in the following quotes:

> I prefer using emacs for complicated editing/coding jobs because of it's features. But I prefer it for simple and quick jobs like config file editing, hacking scripts, etc So it horses for courses.
>
> (JJ110104)

> Because my fingers are friendly with all the commands I need!! I picked up EMACS first and haven't seen the need to change, I especially like the way it handles copy/paste/delete of rectangles within a text file.
>
> (GH120104)

In meeting these specific needs, certain functions of EMACSen become increasingly polished honed, and developed as users engage with and reshape the software.

4. Problems and Solutions: the *Sine Qua Non* of Innovation

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. As Borgman (2003) notes, the professional or the experienced often demonstrate better abilities in addressing well-defined problems because technical terminology, information resources and material resources and demands can be better identified.

This problem-solving orientation is said to be a specific feature of open source innovation. In defining

and solving a problem, one employs those materials that are available and negotiate or cooperate with actors within or across the boundary set by the problem itself. There is no standardised path to be followed; each resolution is a result of situated practices and knowledge. This is evident in many areas of everyday life: Lave (1988) observed that instead of applying conventional mathematical algorithms found in textbooks, people 'perform highly abstract, creative mathematical problem solving' when shopping in a supermarket (Bowker & Star 1999: 288). Another legend about how ancient Taiwanese calculated prices when making a business deal also illustrates how materials at hand make the problem-solving process practical and versatile[1]. In this regard, each episode of problem-solving is materially-grounded, textured and situated.

4.1 Diverse Readings of Efficiency

Efficiency was an important parameter in the process of defining software problems. A problem would not exist if users did not recognise 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' programme 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 were undertaking. Stallman's macros 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 programme.

It is worth noting that Stallman did not sit down and write the editor system programme 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

1 The story is about how ancient Taiwanese put pebbles on the ground in a pyramid shape and remove them following certain rule to perform calculation. The answer was always correct. No one knew how did this method came up so far. It is a legend.

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.

4.2 Access to Problems and Social Network of Innovation

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 to appear, innovation options likely to be more pre-defined, and innovators sharing a consensus on 'what needs to be done'. On the contrary, I want to argue that in a heterogeneous field where diverse innovator actors are found, more problems arise or are triggered. If the boundary of the social group centring 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 multivocality 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 (source code peculiarly). There was a so-called 'collaborative hacker' approach, sharing knowledge and improving each other's work, that encouraged programmers to continually to 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).

However, the generation of too many problems may become counterproductive to innovation. The ability to solve problems is key to innovation. The more a problem is accessible, as noted above, the more diverse actors will be invited to participate in the innovation group centring on the problem. As there is no single perfect solution for a problem, multiple voices and silences should always be welcome (Bowker & Star, 1999: 41). If a problem is presented in an intelligible/perceivable/accessible way, it will encourage more participants to craft solutions (though there may be nothing wrong with

asking a ‘dumb’ question, as the dumbest question can sometimes produce the best answers). As noted by a number of commentaries, well-defined problems in which the given information, operations, and goal are clearly specified will more likely to have solutions than ill-defined problems (Glass *et al.*, 1979; Borgman, 2003). Furthermore, such sources argue that an expert can articulate the queries more specifically and completely than could someone new to the domain. This ability to articulate problems becomes one of the parameters that define expertise, which will be discussed later in this paper.

4.3 Defining and Redefining Problems

As I noted above, while Stallman’s innovation was celebrated because of its extensibility and flexibility, it did however create new problems. One that many saw was the sense of a 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 Weinreib 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 an 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.

5. Innovation elements

Hitherto, I have shown how problems are constructed and how solutions are crafted when a social network of expertise for problem-solving is framed. In the following, I will examine relationships and interactions between actors in a social network of innovation in light of the story of EMACS.

5.1 Teamwork and Communication

Programming or writing codes can be done at an individual level, as one of the features empowered by software technologies. However, teamwork is crucial for producing software of good quality.

Working together with colleagues face to face was the normal routine in the 1970s when EMACS was invented. Though the first e-mail message was generated in 1972, e-mail did not operate at a global level until 1983 when TCP/IP was invented to afford a global speech. Given this material limitation, the way Steele, Stallman and their colleagues communicated in the 70s was unlike what it would be today. At that time, the most practical manner of sharing knowledge and improving peer's works was sitting down at a programmer's terminal, looking into his/her machine and opening up a his/her work to make comments and modifications directly to the machine. That was also how Stallman realised what Steele was doing in his sorting out the confusion associated with the diversity of macros/commands—he recalls passing by his desk by chance and watching what he was doing. Such physical contact in a proximal space (of the office or lab) was the main site of interaction between programmers. When the members in the EMACS team wanted to share work, they either saved programmes to discs (still then rather huge in size) and swapped them, or they simply sat down in front of the other's machines to do hands-on discussion. The personal relationships in the office became one of the main factors influencing the development of the software.

Here is an example of this, involving that Steele's cooperation with Stallman to improve EMACS' print function, which was originally designed by him with a keystroke-triggered feature that reformatted EMACS' source code so that it was both more readable and took up less space. As Steele recalled, 'We sat down one morning. I was at the keyboard, and he was at my elbow. He was perfectly willing to let me type, but he was also telling me what to type.' (Williams, 2002: 87). As Williams, who reported this, also notes, 'Steele was used to marathon coding sessions' which was different from Stallman's "intense coding style" that 'forced Steele to block out all external stimuli and focus his entire mental energies on the task at hand', though eventually after 10 hours they managed to write the print source code within 100 lines (ibid.). Such accounts suggest that when two persons work together, one has to adjust to another's working style or that they negotiated with each other. Agreements had to be reached to process the collaborative work, though this might become quite arduous: as Steele said "It was a great experience, very intense, [but one] I never wanted to do it again in my life." (ibid.). The partnership between Steele and Stallman did not however carry on, in part because they had different working styles. Apart from that, Steele's leaving to work for a commercial company was another reason. Steele's departure posed a risk and brought some uncertainty to the EMACS project.

Stallman's way of managing this uncertainty was to open things up—instead of downsizing the social group to reduce the uncertainty and risk. He released the EMACS source code with a condition of use

that requested feedback about any modification to the source code, while also allowing its redistribution at the same time. In so doing, Stallman actually redefined and broadened the boundary of the developing team and made the EMACS innovation more accessible. In issuing this social contract, on the one hand Stallman drew users' attention to the extensibility of EMACS, and on the other hand fulfilled his belief in the freedom of information. The condition he put on source code distribution therefore acted equally to engage users with a practical attitude as well as to promote his philosophy and to sustain the culture that he was used to living within the MIT AI Lab (its practice/pattern/habit). He attracted new users by saying that,

> Extensibility makes EMACS more flexible than any other editor. Users are not limited by the decisions made by the MACS implementers. What we decide is not worthwhile to add, the user can provide for himself. He can just as easily provide his own alternative to a feature if he does not like the way it works in the standard system.
>
> (Stallman 1998: 1)

Stallman changed the innovation environment in expressing his welcome for open contribution[2]:

> A coherent set of new and redefined functions can be bound into a library so that the user can load them together conveniently. Libraries enable users to publish and share their extensions, which then become effectively part of the basic system. 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.
>
> (ibid.)

> User customization helps in another, subtler way, by making the whole user community into a breeding and testing ground for new ideas. Users think of small changes, try them, and give them to other users—if an idea becomes popular, it can be incorporated into the core system. When we poll users on suggested changes, they can respond on the basis of actual experience rather than thought experiments.
>
> (ibid.)

The above articulations meant that more material and social resources were brought into play. Originally Stallman and colleagues merely dealt with the four copies of macro commands that Steele

2 Note that these narratives are quoted from the documentation written in 1998, almost 20 years after the original release of EMACS. This is to say that the apparatuses Stallman mentioned here are not developed in a day. Instead, their developments are shaped by Stallman's interactions with users and users themselves, as well as the given material environment.

collected. In bringing in more innovation sources, Stallman expanded the social network of EMACS innovation and was able to redistribute the power of the artefacts in the innovation system. As a self-proclaimed hacker, he sought to foster a philosophy of open innovation. As he said:

> [E]ven though there was no organized political thought relating the way we shared software to the design of Emacs, I'm convinced that there was a connection between them, an unconscious connection perhaps. I think that it's the nature of the way we lived at the AI Lab that led to Emacs and made it what it was.
>
> (Stallman 2002: 1)

Stallman's statement points to a key aspect of the social construction of EMACS software: the culture of sharing knowledge embedded in the programming practice in the 70s. Most programmes created were based on this daily practice. If 'culture' is defined as a way of living, which is invisible and embedded in our daily lives, sharing and exchanging programmes is such a programming culture embedded in everyday programming. Stallman adopted this position and tried to sustain it against the wave of commercialising software. The weight on 'the hacker culture', which constantly appears in Stallman's writings and speeches, serves to explain the way he designed software in an unconventional way.

To summarise this section, EMACS was successful because it was able to meet the requirements of most users by being flexible (allowing users to define their own control keys). This feature of flexibility reflects EMACSen's affordance and enables more actors to move into the innovation process. Whereas TECMAC and TMACS (the first version of EMACS editors for TECO written in 1975) appeared to be a solution as the real-time display editor for TECO, new problems had emerged. It took programmers a considerable time to understand each other's definitions of commands before they could bring new order to the programme. The temporary equilibrium reached in the TECO innovation system wobbled again. Steele came up with the idea of a standard set of commands to solve the problem. He, Stallman and the others who shared the same view started to craft such a programme. For the sake of durable efficiency, Stallman reckoned the best way to avoid the derivation of new confusions was to have new-defined commands reported back to him. Hence he wrote the terms of use for EMACS to request feedback of new modifications. This social contract was an informal rule, on the one hand drawing on technical efficiency to sustain the function of EMACS, and on the other hand building up a social network to maintain the 'hacker culture', the daily practice of sharing knowledge that Stallman and other programmers were used to. These social and technical practise and expectations worked together to foster innovation within EMACS. Nevertheless, while the social norm linked to innovation gained greater weight given Stallman's later act, some users were reluctant to conform to the obligations. This is one of the reasons why Stallman's GNU EMACS is labelled as a

moral product regardless of its technical utility (see the discussion on *holy war* below). People who did not share Stallman's vision went on creating other editor programmes. Furthermore, new versions of EMACS were created through yet other problem-solving process (e.g. EINE, ZWEI, XEmacs etc. are derived from the need of porting EMACS with other programming languages).

5.2 Shared Interests and Trust

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 (1987) 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)

Given the proposition that innovation starts with classifying problems, presentations of problems that deliver degrees of problem definition (well-defined problem or ill-defined problem; see Reitman 1964; Simon 1973; Borgman 2003) become crucial for engaging actors in innovation networks. Presenting a well-defined problem will maximise the probabilities of getting useful information. A problem presented also signifies the problem addresser's level of expertise. An actor with higher expertise often phrases her/his questions in technical terms while a lay user often explains her/his problem loosely in everyday, non-technical language.

Hence, the re-presenting of problems, the ways in which they are framed, and redefined, appears to convey expert or a lay user identity and status within the network. This is particularly true in a virtual field. Without knowing each other in person, the identities of actors are represented by and configured through the questions or answers they provide. As a result, an intellectual stratification emerges. Mailing lists or newsgroups for example are typically hierarchical in this regard. In order to maximise the chance of finding peers sharing common interests, people post questions/problems in places where they reckon they will be most likely to find their level of expertise. Certain mailing lists or newsgroups

thus are specialised and reproduce specific levels of expertise. From my review of the mailing lists of local Linux user groups, it would appear that their expertise is normally considered as directed to immediate local interests. However, as observed, even experienced software developers cannot always answer the questions asked in local user group forums. Some questions are very specific to certain types of hardware and systems, and are not easily answered unless one has engaged with these specific topics before. This suggests that local discussion groups normally regarded as simply low-level technical discussions can be innovative (eg. in regard to addressing hardware or system configurations).

Conspicuously, newly developed ICTs are deeply embedded in the mundane practices of programming, peculiarly in the FLOSS innovation system (e.g. e-mail, listservs, newsgroups, repositories, wiki). These tools/techniques are applied to facilitate communications and collaborations (e.g. peer-review) between programmers. The virtual fields providing spaces for discussions on collective topics may be analysed as sites, which offer access to innovation by spanning the boundaries between different social networks. They represent a process of social innovation in a way as Liff & Steward (2003) puts it that "This heterotopian character offers a distinctive new learning context for users through the juxtaposition of practices which traditionally have been pursued at different sites." (Liff & Steward 2003: 331). Stallman's GNU project, like many other FLOSS projects, precisely took advantage of ICTs to target peers to join his innovation network. After Stallman left the MIT AI Lab and planned to write an Unix-like operating system for non-commercial distribution, a range of ICTs facilitated his individual action. These artefacts 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, the message posted entailed uncertainty and risk in

Stallman's project. While 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 participations. 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-esse* (1987), through which actors are enrolled to mobilise 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. In contrast, a closed or centralised 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 multivocality. Following this route, a project will approach closure eventually. Proprietary software is mostly managed in this way.

6. The Materiality of EMACS

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 handles the uncertainties and risks emerging during the process. To extend this analysis of dynamics of software innovations, future works should focus on programmers' mundane practices to explore how key innovation sources (both tangible and intangible) are employed. This account is in the tradition of material culture analysis, and is thereby' as much concerned with how subjects are constituted within material worlds as with how they understand and employ objects (Miller 1987), a perspective analogous to the writings of Latour (e.g. 1993[b], 1996) on science studies and technology' (Miller and Slater 2000: 3). This account has therefore to be multifaceted and not reduced to one dominant or homogeneous notion of either "hacker culture" or "business strategy". It is anticipated that software innovation processes in different contexts can be

understood in this way without falling into the dichotomy of moral/immoral, efficient/inefficient that the typical arguments about FLOSS and proprietary software usually have.

7. EMACS as a Boundary Object

In reviewing the innovation story of EMACS, a variety of EMACSen 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 other programming language than TECO, so new version of EMACS was designed for other programming language such as Lisp, such as EINE and ZWEI, developed by Weinreb, a fellow colleague working on the original version of EMACS as well), 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 the heterogeneity, authors and maintainers of EMACSen try to enhance their legitimacy and uniqueness in providing greater socio-technical functions.

In this account, EMACS, as an extensible editor that can be used flexibly, has served as a boundary object for diverse users. They interpret the role of EMACS in different ways according to their situated practices. A number of common practices of EMACS can be summarised. Some take EMACS as a pure tool for editing texts, programming, gaming, web browsing. Others contribute to the FLOSS community by reporting bugs of EMACS while using it, and still the others residing in the core of EMACS innovation system take EMACS as an artefact or even art work. Because EMACS denotes so many different meanings, the diverse interpretations and manipulations of EMACS can prevent it becoming a stereotyped editor, even though its material effects (as affordances) do give it a specific technical character. This is in contrast to other proprietary software products. For example, Microsoft Office Word software has enrolled a huge number of users for processing their Word documents. When mentioning "*Word*" people take *Microsoft Office Word* for granted. Accordingly, *Microsoft Office Word* is used not more than for typing and word processing. But for EMACS, its high affordance enables many roles to be played in mundane computing practices. Once EMACS is mentioned, people would like to know which version of EMACS is indicated (e.g. GNU Emacs, XEmacs—each indicates different usage habits, particular interests in programming languages, or even

political views), and for what purpose it is used. The potentiality of EMACS for different functions is resilient. Unlike with Microsoft Word, users can customise and configure EMACS to meet their requirements.

Since EMACSen are used in many different ways and denotes various socio-cultural meanings, diverse projects have symbolised users' habits and preferences (socially and technically). In adopting specific tools and participating in specific projects, users are attached to the artefacts. These artefacts grow to be norms to demarcate boundaries. For example, the users of GNU Emacs see themselves different from those of XEmacs, while the broad range of users of Emacs distinguish themselves from other users of other editor programmes, such as *vi*. Holy wars happen often because these users want to fortify their boundaries against each other to show their identities. Accordingly, software programmes 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 symbolised 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-centred view may provide a distinctive research result from the prevailing structure-centred 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 source code, 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 symbolised Gosling's failure to continue the network expansion. 2) The commercial product was not successful. The firm failed to engage actors' interest in it. The commercialised 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 just 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 as a snowball effect. Unlike some proprietary software firms try to lock users in by using proprietry document formats (and critics say they do this in order to dominate the market), EMACSen engage users by presenting greater shared interests in socio-cultural or technical aspects. The story of EMACS sheds some light on the FLOSS innovation, albeit the commercial impetus

should be taken into account in order to understand the situation completely.

Reference

Borgman, C. L. (2003) 'Designing Digital Libraries for Usability'. In Ann Peterson Bishop, Nancy A. Van House, and Barbara P. Buttenfield. 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.

Lave, J. (1988) Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life. Cambridge: Cambridge University Press.

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.

Miller, D. & Slater, D. (2000) The Internet: An Ethnographic Approach. Oxford: Berg.

Miller, D. (1987) *Material Culture and Mass Consumption*. Oxford: Blackwell.

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) GNUEmacs 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.

Biography

Yuwei Lin, Taiwanese, is currently a PhD candidate at the Science and Technology Studies Unit (SATSU), Department of Sociology, University of York, UK. She received her bachelor degree in economics and diploma in women's and gender studies from the National Taiwan University in 1999. Her current research interests centre on free/libre open source software (FLOSS) studies, digital culture (particularly hacker culture), virtual ethnography and sociology of science and technology.