EMACS, `grep`, and UNIX: authorship, invention and translation in software

Christopher Kelty, Rice University

March 20, 2006

Abstract

The UNIX operating system environment has achieved a ubiquity with no parallel in the software world. It began as a computer science experiment in the early 1970s, spread to universities around the world in the late 70s, became the darling of the computer industry in the 1980s, and (in its incarnation as GNU/Linux) helped launch the explosive growth of the Internet in the 1990s. More than simply being a story of the spread of a technology, UNIX has enabled the creation, growth and refinement of a mode of coding—designing, composing, writing and documenting of software—that has become a kind of *lingua franca* for geeks, hackers, scientists, engineers, and hobbyists alike. This paper explores two historical cases of software authorship related to UNIX which have implications for the meaning and practice of modern software coding: the case of the EMACS editor and the GNU General Public License and the case of the programming tool known as `grep`, which searches text using "regular expressions." Both cases offer insight into the nature of contemporary invention and authorship on the realm of software programming.

1 Introduction

The UNIX operating system environment has achieved a ubiquity with no parallel in the software world. It began as a computer science experiment in the early 1970s, spread to universities around the world in the late 70s, became the darling of the computer industry in the 1980s, and (in its incarnation as GNU/Linux) helped launch the explosive growth of the Internet in the 1990s. More than simply being a story of the spread of a technology, UNIX has enabled the creation, growth and refinement of a mode of coding—designing, composing, writing and documenting of software—that has become a kind of *lingua franca* for geeks, hackers, scientists, engineers, and hobbyists alike. This paper explores two historical cases of software authorship related to UNIX which have implications for the meaning and practice of modern software coding.

The first concerns the EMACS editor—one of the most widely used and revered tools for editing text—and for coding software in particular. The creation of this tool in the 1970s literally changed the practice of coding by allowing writers to interact real-time with the code they created—something that has become so utterly familiar that its absence now seems almost impossible. EMACS gave programmers the tools to create tools: ways to extend and to customize their own"environment" for authorship. Even more central to the practice of coding today, EMACS was the subject (in 1983-4) of one of the first real controversies over the copyrightability of software code–one that never went to court, but that instead gave us the first "copyleft" or Free Software license, the GNU General Public License. The controversy raises issues of authorship, ethics and propriety that intersect with the legal and technical milieu in which it occurred.

The other example concerns the creation of the program `grep` a ubiquitous tool for searching through text files for specified patterns. `grep` started life as a theory in biomedical physics (McCulloch and Pitts), was translated to recursive algebra (Stephen Kleene), became a patent for pattern-matching (Ken Thompson), and ultimately ended up as a widely used software tool (written and re-written by numerous UNIX coders). These translations imply diverse forms of writing, and different norms of authorship and ownership that attempt to fix the "idea" or "invention" in different forms, and suggest that concepts which appear in scientific

and technical contexts can have their own agendas and trajectories, despite the ownership over them by individuals and institutions.

What exactly is being invented, however? What distinctions should one draw between the act of writing—in the world of alphabetic literacy that we are generally most familiar with—and the act of inventing, in that classical Edisonian American register of creative engineering and re-configuration of machines, material and money? Software creation—especially software intended for the purpose of writing more software—is something that is intuitively experienced by an increasingly huge number of people, from the creators of programming and scripting languages to the everyday use of HTML. It is a practice that is rapidly becoming as naturalized as writing on paper has been in the past, and hence one that deserves deeper attention as it does so.

In some ways, the stories in this article could appear somewhat retrograde in the obsessive detailing of the practices of individuals. Rather than looking at complex objects created by the mediation of software and multiple authors—objects like databases that contain a huge amount of material and whose authorship is thereby more complicated, or the example of the Linux Kernel or multi-authored scientific papers)—this paper looks at the solitary practice of invention. In software history, there is a reason to do this: the demand that software production either has already, or will at some point transcend the "artistic" to become a science or a form of engineering is one that has haunted software engineers, architects and programmers since at least 1968, if not since the origins of programming. The desire to separate the activity of design (the detailed intellectual labor of writing) from the "coding" (the ostensibly manual practice of filling in the details) has long been debated in software engineering circles.

However, as has been made clear by numerous studies in the sociology of technology, the implementation of a design never coincides perfectly with the design, and always reveals new problems, leaks, and eventually tropes the development of the design. The classic confessional handbook of just this problem is Frederick Brooks *The Mythical Man Month*[Brooks, 1975]. The persistence of a discourse of software programming as an art (as in Donald Knuth's monumental work *The Art of Computer Programming* [Knuth, 1997]) has led generations of programmers to focus closely on the details of the tools they use to do the kinds of tasks

they do, and so the focus on the solitary individual creation of software tools—tools for writing software—is an area of special resonance for some of the questions of authorship and ownership. Similarly, the everyday activities of people far beyond the arcane realms of software programming are increasingly becoming more and more expert at using tools derived from, based on, or similar to the kinds of tools that software programmers have created—whether that means text-editors for HTML (such as Macromedia's Dreamweaver) or the increasingly common use of versioning (as in something like Wikipedia, or even "track changes" in Microsoft Word) or the complex scripting tools of video and audio editing systems. The very material questions of what tools (and how much control) plagues not only software programmers, but writers, artists and inventors in diverse realms.

The individual is also the focus here not because he or she posesses autonomy and creative independence, but because he/she is a particularly unpredictable kind of medium—a point of translation between one text and another, and it is the goal of the second example to show how the problem of invention is also a problem of tracking the emergence and peregrination of concepts, as they are instantiated in theories, in texts, in machines, and however temporarily, in humans.

Part I

EMACS and the GNU GPL

In lecturing on Liberalism in 1935, John Dewey said the following of Jeremy Bentham: "He was, we might say, the first great muck-raker in the field of law... but he was more than that, whenever he saw a defect, he proposed a remedy. He was an inventor in law and administration, as much so as any contemporary in mechanical production."[1] Dewey's point was that the liberal reforms attributed to Bentham came not so much from his theories as from his direct involvement in administrative and legal reform—his experimentation, his hacking. Whether or not Bentham's influence is best understood this way, it nonetheless captures an important component of liberal reform in Europe and America that is also a key component

[1] John Dewey, *Liberalism and Social Action*, New York, G. P. Putnams Sons [c1935] p 14.

in the story of Free Software: that the route to achieving change is through direct experiment with the system of law.

The same might be said of Richard Stallman, the hacker hero and founder of the Free Software Foundation, creator of the GNU Operating system, EMACS and the GNU General Public License—the most widely used and tested "free software" in the world. Stallman is routinely abused for holding what many perceive to be "dogmatic" or "intractable" ideological positions about freedom and the right of individuals to do what they please with software. While it is no doubt quite true that his speeches and writings clearly betray this dogmatism, it is a mistake to assume that his speeches, ideas or belligerent usage demands (free software vs. open source; GNU/Linux vs. Linux) constitute the real substance of his reform. In fact it is the software he has created and the licenses he has written and re-written which are the key to his Bentham-like inventiveness. Unlike Bentham, however, Stallman is not a creator of law and administrative structure–but a *hacker.*

Stallman's GNU General Public License, it is often pointed out, is a "hack" of the system of federal copyright law to achieve the social benefits of widely shared software. Hacks (after which hackers are named) are clever solutions to problems or shortcomings in technology. Hacks are workarounds, clever shortest-path solutions that take advantage of the characteristics of the system that may or may not have been obvious to the people who designed it. Hacks range from purely utilitarian to mischievously pointless. In the case of free software license, the hack relies on the existence of features of the US Copyright statutes (USC17). Free software licenses are in a sense, immanent to copyright laws—there is nothing illegal or even legally arcane about what they accomplish—but there is nonetheless a kind of lingering sense that this particular use of copyright was not how the law was intended to function. Free Software licenses begin with strong copyright: like all software since 1980, it is copyrighted the instant it is written, and the author (or the employer of the author) are thereby granted a number of strong rights over the dispensation of what has been written.[2] But the free software license immediately makes use of the rights of a copyright holder to curtail or abrogate

[2]Copyright Act of 1976, Pub. L. No. 94-553, 90 Stat. 2541, enacted October 19, 1976 and Copyright Amendments, Pub. L. No. 96-517, 94 Stat. 3015, 3028 (amending 101 and 117, title 17, United States Code, regarding computer programs), enacted December 12, 1980. All amendments since 1976, listed here http://www.copyright.gov/title17/92preface.html

the rights he or she has been given—thus granting licensees rights to copy, distribute, modify and use the copyrighted software, subject to a few (extremely hotly debated) restrictions. Thus, while statutory law suggests that individuals need strong rights, and grants them, free software licenses effectively annul them in favor of other activities, such as sharing, porting and forking software. It is for this reason that they have earned the name *copyleft.*

This is a convenient *ex post facto* description, however. Neither Stallman nor anyone else started out with the intention of "hacking" copyright law. It was not an obvious consequence of the vaunted hacker ethic—but a prosaic response to complex controversy over another invention–an inventive invention–EMACS. The story of the controversy is well known amongst hackers and geeks, but not often told, and not in any rich detail outside of these small circles.[3]

The first, and still the most colorful and compelling version of the story was told in Steven Levy's *Hackers* [Levy, 1984]. It is a story not so much of the details of copyright law, as the "ethical" motivations for the creation of free software. It is, more or less exclusively the story of Richard Stallman, and of the MIT AI Lab in the 1970s, but it is a story of community destruction—a community that had few members, but was, according to the stories, dramatically and intensely open. The AI Lab was a place in which the machines were maintained and customized by the hackers who were employed there by the scientists (and not by either the computer corporations providing the hardware or the AI research faculty). In the AI lab, the Incompatible Time-Sharing System (ITS) was an operating system largely built and maintained by hackers. ITS was radically open: all of the core components were visible to all of the users; there were no passwords or system security and no one who was singly responsible for such issues; it could be (and was) used to program and test theories and experiments in artificial intelligence—and especially, as Stallman understood it, to learn about, program, test the limits of and extend the software and hardware in a context where there was no secrecy and everyone was devoted, according to Stallman, to the same "constructive cooperation." Levy, among others, makes much of the idea of a "hacker ethic" in place at the AI lab (and elsewhere) that essentially supported this version of the visibility

[3]It is told in parts in Peter Wayner, *Free for All: How LINUX and the Free Software Movement Undercut the High-Tech Titans* Harper Business, 2000; Glyn Moody *Rebel Code: Inside Linux and the Open Source Revolution,* Perseus Books 2001; and Sam Williams *Free as in Freedom: Richard Stallman's Crusade fro Free Sotfware* O'Reilly 2002.

and openness of the machines and the source code that ran on them.

Usually, stories of this openness are short on details and long on ethos—producing a myth that is avowedly Edenic, in which hackers live in a world of uncontested freedom and collegial competition something like a writers commune without the alcohol or the brawling. But there is no sense in which Stallmans story represents a lost "golden era" of software programming in which all source code was free and all machines were open—rather, it represents the beginning of something that Stallman would actively bring into being through his narration of this moment, as well as his superhuman creation of software and his activism in the 1980s. The details of the creation of the first Free Software license show it to be something that took great effort to produce, out of changing circumstances that reached far beyond this small-scale community. There is no sense in which the GPL can be understood as a nostalgic return to a golden-age of small scale communities freed from the dominating logic of bureaucratic modernity—it is instead a fascinating, reflexive and powerful response to a changed political economy of computers and networks.

2 EMACS

EMACS is a text editor; it is also something like a religion. As one of the two most famous text editors it is frequently lauded by its devoted users, and attacked by detractors who prefer it's competitor (Bill Joy's `vi`, created in the late 1970s). Emacs is more than just a tool for writing text—for many programmers it was the principle interface to the operating system in general. It allowed a programmer not only to write a program, but to debug it, to compile it, to run it, to email it to another user, all from within EMACS. What's more, it allowed users to quickly and easily write extensions that automated frequent tasks. It can do almost anything, but it can also frustrate almost anyone. The name itself is taken from its much admired extensibility: Emacs stands from "editing macos" because it allows programmers to quickly record a series of commands and bundle them into a macro that can be called with a simple key combination. In fact, it was one of the first editors (if not the first) to take advantage of keys like CTRL and META, as in the now ubiquitous CTRL-S familiar to users of non-free word processors like Word™.

It wasn't always this way. Before the UNIX-dominated minicomputer era, there were very few programs for directly manipulating text on a display. The ability to conceive of source code independently of a program running on a machine first meant conceiving of it as typed, printed, or hand-scrawled code which programmers would scrutinize in its more tangible paper-based form. Editors that allowed programmers to display the code in front of them on a screen, manipulate it directly, and save changes to those files were an innovation of the mid to late 1960s, and were not widespread until the mid-1970s (and this only for bleeding edge academics and computer corporations). Along with a few early editors, such as QED (originally created by Butler Lampson and Peter Deutsch, and re-written for UNIX by Ken Thompson), one of the most famous of these was TECO (Text editor and corrector) written by Dan Murphy for DEC's PDP-1 computer in 1962-3. Over the years, TECO was transformed (ported and extended) to a wide variety of different machines, including machines at Berkeley and MIT, and other DEC hardware and operating systems. By the early 1970s, there was a version of TECO running on ITS, the system in use at the AI Lab, and it formed the basis for EMACS, (thus EMACS was itself conceived of as a series of Macros for a separate editor: "Editing MACroS for TECO"). Like all projects on ITS at the AI Lab, many people contributed to the extension and maintenance of EMACS (including Guy Steele, Dave Moon, Richard Greenblatt, Charles Frankston and others), but there is a clear recognition that Stallman made it what it was. The earliest AI Lab Memo on EMACS, written by Eugene Ciccarelli (no 447) says:

> Finally, of all the people who have contributed to the development of EMACS, and the TECO behind it, special mention and appreciation go to Richard M. Stallman. He not only gave TECO the power and generality it has, but brought together the good ideas of many different Teco-function packages, added a tremendous amount of new ideas and environments, and created EMACS. Personally one of the joys of my avocational life has been writing Teco/EMACS functions; what makes this fun and not painful is the rich set of tools to work with, all but a few of which have an "RMS" chiseled somewhere on them.[4]

At this point, in 1978, EMACS lived largely on ITS, but its reputation soon spread, and it was ported to DEC's "Tops-20" (aka Twenex) operating system, and re-written for

[4]Eugene Ciccarelli "An Introduction to the EMACS Editor" MIT Artificial Intelligence Laboratory AI Memo No. 447 1978, pg 2.

Multics and the MIT's LISP Machine (on which it was called EINE—Eine Is Not Emacs, and which was followed by ZWEI—Zwei Was Eine Initially). Stallman himself was simultaneously pleased with the acceptance of EMACS and frustrated by this proliferation, since it meant that the work fragmented into different projects, each of then EMACS-like, rather than building on one core project (much like the proliferation of UNIX that was happening at the same time), and in the 1981 report he said:

> The proliferation of such superficial facsimiles of EMACS has an unfortunate confusing effect: their users, knowing that they are using an imitation of EMACS and never having seen EMACS itself, are let to believe they are enjoying all the advantages of EMACS. Since any real-time display editor is a tremendous improvement over what they probably had before, they believe this readily. To prevent such confusion, we urge everyone to refer to a nonextensible imitation of EMACS as an "ersatz EMACS."[5]

Thus, while EMACS in its specific form on ITS was a creation of Stallman, the *idea* of EMACS, or of any "real-time display editor" was proliferating in different forms and on different machines. The porting of EMACS, like the porting of UNIX, was facilitated by both its conceptual design integrity, and its widespread availability.

The phrase "non-extensible imitation" captures the combination of design philosophy and political philosophy that EMACS represented. Extensibility was not just a useful feature for the individual computer user—it was a way to make the improvements of each new user easily available equally to all by providing a standard way for users to add extensions and to learn how to use new extensions that were added (the "self-documenting" feature of the system). The program had a kind of conceptual integrity that was compromised when it was copied imperfectly; a modular, extensible design that by its very nature invited users to contribute to it and to extend it and to make it perform all manner of tasks—to literally copy it instead of imitating it. For Stallman this was not only a fantastic design for a text editor—but an expression of the way he had always done things in the small-scale setting of the AI Lab. It is unclear whether Stallman was aware of the kind of cut-throat competition to

[5] Richard Stallman, "EMACS: The Extensible, Customizable Self-Documenting Display Editor" MIT Artificial Intelligence Laboratory, AI Lab Memo 519a, 26 March 1981, p. 19. Also published as Richard M. Stallman, "EMACS the extensible, customizable self-documenting display editor", *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, p.147-156, June 8-10, 1981.

differentiate the *hardware* which had dominated the computer industry almost from day one, but it is clear that he fundamentally rejected such competitive differentiation in the *software* world, precisely because it seemed bound to lead to the most profound inefficiency—and as Levy recounts, to the destruction of a "community." Needless to say, not everyone shared Stallman's sense of communal order.

So in order to facilitate this kind of sharing, with respect to EMACS in particular, Stallman started something he called the "EMACS Commune". At the end of the same report "EMACS: The Extensible, Customizable Self-Documenting Display Editor" from 26 March 1981, he explained the terms of distribution for EMACS: "It is distributed on a basis of communal sharing, which means that all improvements must be given back to me to be incorporated and distributed. Those who are interested should contact me. Further information about how EMACS works is available in the same way."[6]

Later in the year another report intended as a user's manual for EMACS (AI Lab Memo 554, 22 October 1981) has similar more detailed and slightly more colorful instructions:

> EMACS does not cost anything; instead, you are joining the EMACS software-sharing commune. The conditions of membership are that you must send back any improvements you make to EMACS, including any libraries you write, and that you must not redistribute the system except exactly as you got it, complete. (You can also distribute your customizations, *separately.*) Please do not attempt to get a copy of EMACS, for yourself or anyone else, by dumping it off of your local system. It is almost certain to be incomplete or inconsistent. It is pathetic to hear from sites that received incomplete copies lacking the sources [source code], asking me years later whether sources are available. (All sources are distributed, and should be on line at every site so that users can read them and copy code from them). If you wish to give away a copy of EMACS, copy a distribution tape from MIT, or mail me a tape and get a new one.[7]

Because EMACS was so widely admired and respected, it gave Stallman a certain amount of power over this commune. If it had been an obscure non-extensible tool useful for a single purpose, no one would have heeded such demands—but because EMACS is by nature the kind of tool that is both useful for all kinds of tasks and customizable for specific ones, Stallman was not the only person who benefited from this communal arrangement. Two disparate sites may

[6] Ibid. pg 24.

[7] Richard M. Stallman "EMACS Manual for ITS Users" MIT Artificial Intelligence Laboratory, AI Memo 554 22 October 1981, pg 163.

well have needed the same macro extension, and so many could easily see the social benefit in returning extensions for inclusion—as well as the idea of becoming a kind of co-developer of such a powerful system. As a result, the demands of the EMACS commune, while unusual, and somewhat totalitarian, were of obvious value to the flock.

The terms of EMACS distribution agreement were not quite legally binding; nothing except Stallman's reputation, his hectoring or a desire to contribute compelled people to participate. On the one hand Stallman had not yet chosen to, or been forced to, understand the details of the legal system, and so the EMACS commune was the next best thing; on the other hand, the state of intellectual property law was in great flux at the time, and it was not clear to anyone, whether corporate or academic, exactly what kind of legal arrangements would be legitimate. Stallman's "agreement" was more a set of informal rules that expressed the general sense of mutual aid that was a feature of both the design of the system, and Stallman's own experience at the AI Lab. It was an expression of the way Stallman *expected* others to behave, and his attempts to punish or shame people, a kind of informal enforcement of these expectations. The small scale of the community worked in Stallman's favor.

Stallman's experiment with the EMACS commune is the beginning of what is now a two decade-long experiment with copyright licenses. And whereas there was often little to no discussion amongst UNIX companies in the 1980s about the relationship of design (and a standard UNIX) to those of intellectual property (even though they continued to speak of "open vs. proprietary" systems), the debates around EMACS and Richard Stallman were absolutely saturated in legal discussion. While UNIX vendors left intellectual property rules unchallenged, and simply assumed that they were the essential ground rules of debate, Stallman made them the substance of his experiment, like Bentham, he became something of a legal muckraker as a result.

The evolution from the EMACS Commune to the fully-fledged GNU General Public License (that now serves as one of the *de facto* free software licenses) lasted from 1983 until about 1989 (with the release of the GNU GPL Version 1.0). The early history of this is both frequently repeated, and somewhat mysterious—it is a story told by Stallman himself, and in various biographical accounts and interviews, but it is also a story that unfolded more or less publicly on what was then a very new medium: USENET (and via gateways to and from

ARPAnet).[8]

Despite Stallman's request that imitators refer to their versions of EMACS as "ersatz EMACS"—few did. What is perhaps even more surprising, however, is that few people took the core ideas of EMACS, implemented them in imitation, and then called it something else (such as the examples of EINE and ZWEI). As in the case of UNIX, the proliferation of "forked" versions of the software did not render them any less UNIX, even when AT&T insisted on ownership of the trademarked name. EMACS, on the other hand was not a trademarked name—nor was the ownership of the original version clear. As time went on, and EMACS was ported, forked, re-written, copied, or imitated on different operating systems and different computer architectures in universities and corporations around the world; within five or six years, a number of different versions of EMACS were in wide use.

Stallman's EMACS for Twenex (TOPS-20 on DEC machines) was the original (not including it's origins in ITS and TECO, listed above). Perhaps the second most famous implementation of EMACS was the version written by James Gosling at Carnegie Mellon University called GOSMACS. GOSMACS was written in C for UNIX while Gosling was a graduate student at CMU.

GOSMACS was distributed by Gosling, and was the most well-known version for UNIX; Gosling even ran an electronic mailing list—which was an unusual innovation at the time. The list allowed users of GOSMACS to ask questions and make suggestions about the program. Gosling single-handedly filled orders and sent out updates and answered queries about his own program. Apparently, he wanted it that way, as he asked people *not to re-distribute the program*, but to come back to him (or send interested parties to him directly) for new versions. As a result of this practice, GOSMACS found a large following in a manner different from Stallman's EMACS had, with Gosling firmly at the center. Again, for the first few years, the small scale of users worked in Gosling's favor, and his program spread and improved with the help of others. But, by 1982, the popularity of EMACS, like that of UNIX, was putting pressure on both Stallman and Gosling, and the realities of maintaining the software, as well

[8]This story is told through the detailed analysis of the USEnet and ARPAnet archives of the controversy. All of the messages referenced here are cited by their "Message-ID" which should allow anyone interested to access the original messages through, for instance, Google Groups (groups.google.com) which maintains the historical archives for the time being.

as the commitment to its availability would create the controversy at the heart of this story.

3 The controversy

In brief the controversy is this: Gosling decided in 1983 to sell GOSMACS to a commercial software vendor, Unipress. His decision was based on his own inability to effectively maintain the product and keep giving support to users—a task he consider the proper role of a corporation. Stallman, who remained committed to a non-commercial "commune" of EMACS, considered this "software sabotage." Even though Gosling had been substantially responsible for writing GOSMACS, Stallman felt some propriety for this "ersatz" version—or at least, was irked that there now existed no UNIX version of EMACS. So Stallman wrote one himself (as part of a project he announced around the same time, called GNU "GNU's not UNIX", to create a complete non-AT&T version of UNIX). He called his version GNU EMACS and released it to whoever wanted it. The crux of the debate hinges on the fact that Stallman used (ostensibly with permission) a small piece of Gosling's code in his new version of EMACS—a fact that led numerous people, including the new commercial suppliers of EMACS to cry foul. Recriminations and legal threats ensued and the controversy was eventually resolved by Stallman rewriting the offending code, thus creating an entirely "Gosling-free" version that went on to become the standard UNIX version of EMACS.

The story raises several questions with respect to the legal context in particular, questions about the copyrightability of software, the definition of what counts as software and what doesn't, and the meaning of infringement. While the controversy did not resolve any of these issues (the first two would be resolved by Congress and the courts, the third is still somewhat murky), it did clarify the legal issues for Stallman sufficiently that he could proceed from the informal EMACS Commune to create the first version of a free software license, the GNU General Public License.

By 1983, Gosling's mailing list for GOSMACS was being forwarded to a USENET group called net.emacs (as well as to ARPAnet via a gateway maintained by John Gilmore at Sun Microsystems). The bulk of the messages on the list were messages concerning the use and modification of GOSMACS, but in January of 1983, a slightly different kind of message

appeared: an announcement by Steve Zimmerman that the company he worked for, Computer Corporation of America, had created a commercial version of EMACS called CCA EMACS.[9] Zimmerman had not written the version entirely, but he had taken a version written by Warren Montgomery at Bell Labs (written for UNIX on PDP-11s) and created the version called CCA EMACS that was being used by programmers at CCA. Zimmerman had apparently distributed it by FTP at first,but when CCA determined that it might be worth something, they decided to exploit it commercially, rather than letting Zimmerman distribute it "freely".[10]

By Zimmerman's own account, this whole procedure required ensuring that there was nothing left of the original code by Warren Montgomery that Bell Labs owned.[11]

Following Zimmerman's announcement, some readers of net.emacs complained that it was meant for discussion of GOSMACS and not any other versions. Others seemed fine with the idea of any EMACS being suitable for discussion, and over the months, a number of people asked for comparisons between CCA EMACS and Gosling EMACS. Although it is difficult to say from these postings alone, this development may well have influenced James Gosling to make his own significant announcement in April of 1983:

> Message-ID: bnews.sri-arpa.865
> Newsgroups: net.emacs
> X-Path: utzoo!decvax!decwrl!sun!megatest!fortune!hpda!hplabs!sri -unix!James.Gosl...@CMU-CS-VLSI.ARPA
> From: James.Gosl...@CMU-CS-VLSI.ARPA
> Date: Tue Apr 12 04:51:12 1983
> Subject: Emacs goes commercial
> Posted: Sat Apr 9 13:49:00 1983
> Received: Tue Apr 12 04:51:12 1983

[9] Message-ID: bnews.cca.4248

[10] Message-ID: 385@yetti.UUCP. Just to complete the circle, Montgomery also posted a note to net.emacs describing his own version, and discussing its potential licensing by Bell Telephone labs. See Message-ID: bnews.ihnss.1395

[11] June 26, 1985 net.emacs. The details are even more arcane: AT&T was less concerned about copyright infringement than they were about the status of their trade secrets. Since trade secrets are secrets only so far as the corporation holding the secret makes a concerted effort to keep it secret, AT&T needed to demonstrate that it was doing so; Zimmerman quotes a statement that he claims indicates this "Beginning with CCA EMACS version 162.36z, CCA EMACS no longer contained any of the code from Mr. Montgomery's EMACS, or any methods or concepts which would be known only by programmers familiar with BTL EMACS of any version." The statement does not mention copyright, but implies that the CCA version does not contain any AT&T trade secrets, thus maintaining their secret status. The fact that EMACS as a conceptual design (a particular kind of interface, a LISP interpreter, and extensibility) was quickly becoming the most imitated piece of basic development software in the world was apparently irrelevant to this issue.

> The version of Emacs that I wrote is now available commercially through a company called Unipress who can be reached at (201) 985-8000. They will be doing development, maintenance and will be producing a real manual. Emacs will be available on many machines (it already runs on VAXen under Unix and VMS, SUNs, codatas, and Microsoft Xenix). Along with this, I regret to say that I will no longer be distributing it.
>
> This is a hard step to take, but I feel that it is necessary. I can no longer look after it properly, there are too many demands on my time. Emacs has grown to be completely unmanageable. Its popularity has made it impossible to distribute free: just the task of writing tapes and stuffing them into envelopes is more than I can handle.
>
> The alternative of abandoning it to the public domain is unacceptable. Too many other programs have been destroyed that way.
>
> Please support these folks. The effort that they can afford to put into looking after Emacs is directly related to the support they get. Their prices are reasonable.
>
> James.

The message is worth paying careful attention to: Gosling's work of distributing the tapes had become "unmanageable"—and the work of maintenance, upkeep, and porting (making it available on multiple architectures) is something he clearly believes should be done by a commercial enterprise. Gosling, it is clear, did not understand his effort in creating and maintaining EMACS to have emerged from a communal sharing of bits of code—even if he had done a Sisyphean amount of work to incorporate all the changes and suggestions his users had made—but he did long have a commitment to distributing it for free; a commitment that resulted in many people contributing bits and pieces to Gosling's EMACS.

"Free" however, did not mean "public domain"—as is clear from his statement that "abandoning it" to the public domain would destroy it. The distinction is an important one that was lost on many sophisticated members of net.unix as it has continued to be over the years: Here "free" means without charge—but Gosling had no intention of letting that imply that he was not the author, owner, maintainer, distributor, and sole beneficiary of whatever value Gosling EMACS had. His decision to sell it to Unipress was a decision to transfer these rights to a company who would then charge for all the labor he had provided for no charge (for "free") prior to that point. Such a distinction was not clear to everyone—many people considered the fact that GOSMACS was free to imply that it was in the public domain.[12]

[12] In particular, a thread discussing this in detail starts at Message-ID: 172@encore.UUCP and includes Message-ID: 137@osu-eddie.UUCP , Message-ID: 1127@godot.UUCP , Message-ID: 148@osu-eddie.UUCP .

Not least of these was Richard Stallman, who referred to Gosling's act as "software sabotage" and urged people to avoid using the "semi-ersatz" Unipress version.[13]

Up to around this time, Stallman had been deeply involved in a fight at the AI Lab over the commercialization of the LISP Machine (this is the story told in (Levy 1984 [Levy, 1984]). The advancing commercialization of EMACS would naturally have been a frustrating state of affairs, and no doubt contributed to his plan, which he revealed in the midst of this controversy, to create a completely new, non-commercial, non-AT&T operating system that he would give away free to anyone who could use it; he announced this on Sept 27, 1983.[14]

> Free Unix!
>
> 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.

His justifications were simple:

> Why I Must Write GNU
>
> I consider that the golden rule requires that if I like a program I must share it with other people who like it. I cannot in good conscience sign a nondisclosure agreement or a software license agreement.
>
> So that I can continue to use computers without violating my principles, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free.[15]

The message makes no mention of "public domain" software—but it does use the troublesome word "free." However, the meaning of free was not nearly as clearly defined by Stallman as it would later become, and there is no real mention during this period of the necessity of having a license when the goal is to give it away freely. And yet this does not quite mean that Stallman intends to "abandon it" to the public domain, as Gosling suggested. Instead, Stallman likely intended to require the same "Emacs commune" rules to apply—rules that he

[13] Message-ID: bnews.sri-arpa.988

[14] Message-ID: 771@mit-eddie.UUCP announced on net.unix-wizards and net.usoft

[15] Message-ID: 771@mit-eddie.UUCP
Date: Tue, 27-Sep-83 13:35:59 EDT

would be able to control largely by overseeing (in a non-legal sense) who was sent or sold what, and by demanding (in the form of messages attached to the software) that any modifications or improvements come in the form of donations.

Stallman's original plan for GNU was to start with the core operating system, the kernel, but his extensive work on EMACS (and the sudden need for a free EMACS that would work on a UNIX system, in order to bootstrap a free version of the OS) led him to start there. So in 1984, and into 1985, he and others began work on GNU EMACS, which was a Unix version of the EMACS he had been working on since the 1970s. All the while CCA EMACS and Unipress EMACS (Gosling's version) were being sold to Unix users (while DEC users continued to use the original free version created by Stallman). Zimmerman left CCA in Aug of 1984, Gosling had gone on to Sun Microsystems, where he has since stayed. The market in UNIX operating systems had exploded with the success of Sun, and the debates about "open systems" were well underway.

By March of 1985, Stallman had a complete version (version 15) of GNU EMACS running on the BSD4.2 Version of UNIX (the version Bill Joy had helped create, and had taken with him to form the core of Sun's version of UNIX), running on Dec's VAX Computers. Stallman announced this software in a characteristically flamboyant manner by publishing an article in the computer programmers' monthly magazine *Dr. Dobbs*, entitled the "GNU Manifesto."[16]

Stallman's announcement that a free version of Unix EMACS was available caused some concern amongst the commercial distributors. The main such concern was that GNU EMACS 15.34 contained code marked "Copyright (c) James Gosling"—code used to make EMACS display on screen. The "discovery" (not so difficult, since Stallman always distributed the source code along with the binary) that this code had been re-used by Stallman led to one of the earliest extended discussions of copyright law and software archived on USENET. The discussion is fascinating in historical terms, because it contains most of the issues that emerge repeatedly in the context of the creation of software over the years.

It is also a story of the structure of rumor on the USENET—a bit like the child's game of Chinese Whispers, except that the translations are all archived. The earliest mention of

[16] Richard Stallman, "The GNU Manifesto" *Dr. Dobbs Journal,* March 10.3 ? 1985 http://www.gnu.org/gnu/manifesto.html

this issue comes not in net.emacs, but on fa.info-vax, a newsgroup devoted to discussions of the Vax computer systems (fa = from Arpanet) and included a discussion (Ron Natalie and Marty Sasaki) labeled "GNU EMACS: How Public Domain?":

> FOO, don't expect that GNU EMACS is really in the public domain. UNIPRESS seems rather annoyed that there are large portions of it that are marked copyright James Gosling.[17]

This message was reprinted on June 4, 1985 on net.emacs with the addendum:

> RMS's work is based on a version of Gosling code that existed before Unipress got it. Gosling had put that code into the public domain. Any work taking off from the early Gosling code is therefore also public domain.[18]

The addendum was then followed by an extensive reply from Steve Zimmerman, the author of CCA EMACS (which had been based on Warren Montgomery's Bell Labs EMACS, but re-written to avoid re-using the code—hence his understanding of the issue seems to have been both deep, and troubling for him):

> This is completely contrary to Gosling's public statements. Before he made his arrangements with Unipress, Gosling's policy was that he would send a free copy of his Emacs to anyone who asked, but he did not (publicly, at least) give anyone else permission to make copies. Once Unipress started selling Gosling's Emacs, Gosling stopped distributing free copies and still did not grant anyone else permission to make them; instead, he suggested that people buy Emacs from Unipress. All versions of Gosling's Emacs distributed by him carry his copyright notice, and therefore none of them are in the public domain. Removing copyright notices without the author's permission is, of course, illegal. Now, a quick check of my GNU Emacs sources shows that sure enough, a number of files have Gosling's copyright notice in them. What this all means is that unless RMS got written permission from Gosling to distribute his code, all copies of GNU Emacs constitute violations of the copyright law. All those people making such copies, including those people who allow them to be copied off their machines, could each be liable for large sums of money. I think that RMS had better tell us if he has Gosling's written permission to make these copies. If so, why has he not stated this earlier (preferably in the distribution itself) and thereby cleared up a potentially major point of confusion? If not, why has he gone ahead and made many, many people

[17] Message-ID: 7773@ucbvax.ARPA
[18] Message-ID: 11400007@inmet.UUCP

> liable for criminal prosecution by recommending that they distribute this code without even warning them of their liability? (People who distribute this code would be liable even if they claim that they didn't see Gosling's notices; the fact that the notices are there is sufficient. "Ignorance of the law is no excuse.")
>
> Now, I have nothing against free software; it's a free country and people can do what they want. It's just that people who do distribute free software had better be sure that they have the legal right to do so, or be prepared to face the consequences. (Jun 9, 1985).[19]

Stallman replied the next day:

> Nobody has any reason to be afraid to use or distribute GNU Emacs. It is well known that I do not believe any software is anyone's property. However, for the GNU project, I decided it was necessary to obey the law. I have refused to look at code I did not have permission to distribute. About 5% of GNU Emacs is close to (though quite a bit changed from) an old version of Gosling Emacs. I am distributing it for Fen Labalme, who received permission from Gosling to distribute it. It is therefore legal for me to do so. To be scrupulously legal, I put statements at the front of the files concerned, describing this situation.
>
> I don't see anything I should warn people about--except that Zimmerman is going to try to browbeat them.[20]

Stallman's original defense for using Gosling's code was that he had permission to do so. According to him, Fen Labalme had received written permission (to make use of, or to redistribute is not clear) the display code that was included in GNU EMACS 15.34. According to Stallman, versions of Labalme's version of Gosling's version of EMACS were in use in various places (including Labalme's employer, Megatest), and that they considered this a legally defensible position.

Over the next two weeks, a slough of messages attempted to pick apart and understand the issues of copyright, ownership, distribution and authorship. Gosling wrote to clarify that GOSMACS was never in the public domain, but that "Unfortunately, two moves have left my records in a shambles" and he is therefore silent on the question of whether he granted permission.[21] Zimmerman sent around an anonymized message suggesting that some lawyers somewhere found the "third hand redistribution" argument was legally "all wet".[22]

[19] Message-ID: 717@masscomp.UUCP
[20] Message-ID: 4421@mit-eddie.UUCP
[21] Message-ID: 2334@sun.uucp
[22] Message-ID: 732@masscomp.UUCP

Stallman's biggest concern was not so much the legality of his own actions as the danger that people would choose not to use the software because of legal threats (even if they were issued only as rumors by ex-employees of companies that distribute software they had written). Stallman wanted users not only to feel safe using his software—but to adopt his view that software exists to be shared and improved, and that anything that hinders this is a loss for everyone. Several years later, in a speech he made in 1986 to the Swedish Royal Technical Institute, Stallman expressed in this way:

> Sometimes I think that perhaps one of the best things I could do with my life is find a gigantic pile of proprietary software that was a trade secret, and start handing out copies on a street corner so it wouldn't be a trade secret any more. Perhaps that would be a much more efficient way for me to give people new free software than actually writing it myself; but everyone is too cowardly to even take it.[23]

Stallman's own generosity, and desire to see his work used and built on, is often unbalanced by his own disdain for people who fear what he considers to be unjust law. Indeed, Stallman's legal grounds for using Gosling's code may or may not have been sound. Zimmerman did his best throughout to explain in detail what kind of permission Stallman and Labalme would have needed, drawing on his own experience with the CCA Lawyers and AT&T Bell Labs, all the while berating Stallman for not creating the display code himself. Meanwhile Unipress posted an official message that said "UniPress wants to inform the community that portions of the GNU Emacs program are most definitely not public domain, and that use and/or distribution of the GNU Emacs program is not necessarily proper."[24] The admittedly vague tone of the message left most people wondering what that meant—and whether they intended to sue anyone.

The situation became curiously more complex when Fen Labalme wrote a message to net.emacs, which although it did not clarify the legal status of Gosling's code (Lebalme was also unable to find his "permission" from Gosling), it did raise a related issue: the fact that he and others had made significant contributions to Gosling EMACS, which Gosling had incorporated into his version, and then sold to Unipress without their permission:

[23] http://www.gnu.org/philosophy/stallman-kth.html
[24] Message-ID: 103@unipress.uucp

> As one of the "others" who helped to bring EMACS up to speed, I was distressed when Jim sold the editor to UniPress. This seemed to be a direct violation of the trust that I and others had placed in Jim as we sent him our improvements, modifications, and bug fixes. I am especially bothered by the general mercenary attitude surrounding EMACS which has taken over from the once proud "hacker" ethic -- EMACS is a tool that can make all of our lives better. Let's help it to grow![25]

Lebalme's implication here (though he may not even have realized this himself) is that Gosling may have infringed on the rights of others in selling the code to Unipress. Indeed the general irony of the complicated situation was certainly not as evident as it might be given the emotional tone of the debates: Stallman was using code from Gosling, based on permission Gosling had given to Lebalme; but Lebalme had written code for Gosling which he had commercialized without telling Lebalme; and all of them were creating software that had been originally conceived in substantial part by Stallman, who was now busy rewriting the very software Gosling had rewritten for UNIX.

All of the legal discussion (including several other equally arcane concerns about a piece of software called dbx) was made moot, however, when Stallman announced that he would simply rewrite the display code in EMACS.

> I have decided to replace the Gosling code in GNU Emacs, even though I still believe Fen and I have permission to distribute that code, in order to keep people's confidence in the GNU project.
>
> I came to this decision when I found, this night, that I saw how to rewrite the parts that had seemed hard. I expect to have the job done by the weekend.[26]

And indeed, By July 4, independence day, Stallman was able to send out a message that said:

> Celebrate our independence from Unipress!
> Emacs version 16, 100% Gosling-free, is now being tested at several places. It appears to work solidly on Vaxes, but some other machines have not been tested yet.[27]

[25] Message-ID: 18@megatest
[26] Message-ID: 4559@mit-eddie.UUCP
[27] Message-ID: 4605@mit-eddie.UUCP

The fact that it only took one week to create the code is a testament to Stallman's widely recognized skills in creating great software—it doesn't necessarily represent any sense of threat or urgency. Unipress also seems to have been concerned about their own reputation, and the implication made by Stallman that they had forced this issue to happen. A month later, Mike Gallaher wrote to insist, somewhat after the fact, that Unipress had no intention of suing anyone—as long as they were using the Gosling-free EMACS, version 16 and higher.

> UniPress has no quarrel with the Gnu project. It bothers me that people seem to think we are trying to hinder it. In fact, we hardly did or said much at all, except to point out that the Gnumacs code had James Gosling's copyright in it. We have not done anything to keep anyone from using Gnumacs, nor do we intend to now that it is "Gosling-free" (version 16.56).
>
> You can consider this to be an official statement from UniPress: There is nothing in Gnumacs version 16.56 that could possibly cause UniPress to get upset. If you were afraid to use Gnumacs because you thought we would hassle you, don't be, on the basis of version 16.56.[28]

Both Stallman and Unipress received various attacks and defenses from observers of the controversy—many people pointed out that Stallman should get credit for "inventing" EMACS, and that the issue of him infringing on his own invention was therefore ironic—others proclaimed the innocence and moral character of Unipress, who, it was claimed, were providing more of a service (support for EMACS) than the program itself.

Some readers interpreted the fact that Stallman had rewritten the display code (whether under pressure from Unipress or not) as confirmation of the ideas expressed in the GNU Manifesto—namely that commercial software stifles innovation. The logic went: precisely because Stallman was forced to re-write the code, rather than build on something that he himself assumed he had permission to do, there was no innovation, only fear-induced caution. On the other hand, latent within this discussion is a deep sense of propriety about what people had created—many people contributed to making EMACS what it was, not only Stallman and Gosling and Zimmerman—and most people had done so under the assumption (legally correct or not) that it would not be taken away from them, or worse that others might profit by it.

[28] Message-ID: 104@unipress.uucp

This unresolved sense of common stewardship is only partially answered in Gosling's (and others') distinction between creating the software and maintaining it. Maintaining might include the incorporation of the work of others (whether that means small or large changes) or it might mean simply distributing and "supporting" the project. The distinction (or the difficulty of making one) is one that runs all the way through the history of Free Software, and re-emerges over and over again as a question of business models, questions of legitimate ownership, and of different styles of project management in Free Software as well. One programmer's support is another programmer's fundamental contribution.

4 The context of copyright

Legal concerns play an important, but non-obvious part in the details of this story: whether it is the legal threats of Zimmerman, the uncertain status of the Gosling copyright, or the unstable meaning of "permission" there is a persistent attempt to come to terms with not only the moral and ethical tensions that emerged in the EMACS controversy, but the legal realities as well. The fact that US copyright law had just gone through its biggest changes since 1909 confused the situation beyond what any of the actors might have realized, by shifting the ground of recent practice in software and computer engineering, and by introducing a new set of uncertainties.

The 1976 Copyright Act introduced a number of changes that had been some 10 years in the making, and had come into existence with new concerns over copy machines, home audio-taping, and the new video-cassette recorders. But the changes proposed for 1976 made no mention of software in particular—and this was an oversight that frustrated many in the computer industry, in particular the young software industry. Pursuant to this oversight, the presidential Commission on New Technological Uses of Copyright (CONTU) was charged with making suggestions for changes to the law with respect to software. In 1980, congress implemented these changes, and explicitly added software to Chapter 17 of the US Copyright Statute as something that could be considered copyrightable by law.[29]

As is clear from the story, many software programmers had been doing so already (Gosling

[29] CONTU Report http://digital-law-online.info/CONTU/contu1.html (visited Jul 8 2005).

had marked all of his code with a copyright notice) but most corporations had been relying primarily on trade secret status to protect their software up to that point, given the uncertainties in the law. Trade secret law, implemented state by state, could be relied upon to protect the intellectual property of a corporation so long as the material was actually kept secret—and hence an elaborate practice of maintaining trade secrets developed in the computer industry, of which we have seen examples both in the case of AT&T's attempts to maintain trade secret status for UNIX, and in the case of Zimmerman's experience of the same with respect to the BTL version of EMACS.

Software patenting, as well, was in the midst of uncertainty, with congress, the USPTO and the practices of corporations sending mixed messages—a number of what computer scientists considered "fundamental algorithms" had been granted patents—but it is not until the late 1980s and 1990s that the number of patents and the number of patent-infringement lawsuits on software started to skyrocket.[30]

The 1980 amendment to the copyright law answered one of three lingering questions about the copyrightability of software, the simple question of whether it was copyrightable material at all: yes. It did not however, designate what "software" meant. During the 1980s, a series of court cases helped specify what counted as "software" including source code, object code (binaries), screen display/output, look and feel, and microcode/firmware.[31] The final question concerns how much similarity constitutes and infringement in each of these cases, and this is something the courts are still faced with adjudicating.

The story of Gosling/Unipress EMACS confronts all of these questions. Stallman's initial creation of EMACS was accomplished under conditions where it was unclear whether copyright would apply (and furthermore, assumed by many that it did not), but that did not stop Gosling

[30] *Diamond V. Diehr*, 450 U.S. 175 (1981) 450 U.S. 175 (Supreme court decision forcing P.T.O. to grant patents on software).

[31] Copyright cases: *Whelan Associates Inc. v. Jaslow Dental Laboratory, Inc., et al*, U.S. Court of Appeals, Third Circuit August 4, 1986797 F.2d 1222, 230 USPQ 481 (Affirming that "structure (or sequence or organization)" of software is copyrightable, not only the literal software code); *Computer Associates International, Inc. v. Altai, Inc.* U.S. Court of Appeals, Second Circuit June 22, 1992 982 F.2d 693, 23 USPQ2d 1241 (Arguing that the structure test in Whelan was not sufficient to determine infringement, and thus proposing three part "abstraction-filiation-comparison" test); *Apple Computer, inc. V. Microsoft Corp* U.S. 9th Circuit Court of Appeals, 35 F.3d 1435 (9th Cir. 1994) 35 F.3d 1435 (finding that the "desktop metaphor" used in Macintosh and Windows was not identical, and thus did not costitute infringement); *Lotus Development Corporation v. Borland International, Inc.* (94-2003), 513 U.S. 233 (1996) (Finding that the "look and feel" of a menu interface was not copyrightable.

from asserting copyright over his version. The 1976 changes to the statute did away with the necessity of registering copyrights, which meant that everything written after that point would have been automatically copyrighted, were it not uncertain whether software was *writing* or not. With the 1980 clarification, suddenly everything in code was copyrighted automatically, according to the law. Such a change was not obvious to participants however—especially participants who had grown accustomed to thinking of some kinds of objects being in the "public domain"—and the "openness" of both EMACS and GOSMACS goes some distance towards explaining this confusion.

Since neither Stallman nor Gosling sought to keep the program secret in any form, either by licensing it, or by requiring users to keep it secret—there could be no claims of trade secret status on either program. A customary assumption would then have been that the program was therefore "in the public domain"—something neither Stallman nor Gosling were entirely happy with, since both of them continued to maintain the software they had created, and neither had relinquished any propriety for having created it—only certain rights to use and modify it. Nor had either Stallman nor Gosling attempted to patent any of the algorithms (which was presumably, anathema to both of them).

By contrast, the version of EMACS that Warren Montgomery had created at Bell Labs (and on which Zimmerman's CCA version would be based) *was* the subject of trade secret protection by AT&T, who were at the time, still a year away from divestiture, and thus unable to engage in the commercial exploitation of the software. When Zimmerman's company CCA sought to commercialize the version of UNIX he had based on Montgomery's, it became necessary to remove any AT&T code, in order to avoid violating their trade-secrets. CCA in turn distributed their EMACS as either binary or as source (the former costing about $1000, the latter as much as $7000), and hence relied on copyright rather than trade secret protection, to protect their version from unauthorized uses.

Thus the question of whether software was copyrightable—or copyrighted—was being answered differently in different cases: AT&T relying on trade secret status, Gosling, Unipress and CCA negotiating over copyrighted material, and Stallman experimenting with his "commune". Although the uncertainty was legally answered by the 1980 Amendment, it was not the case that everyone instantly understood this new fact, or changed their practices based

on it. There is ample evidence throughout the USENET archive that the 1976 changes were poorly understood (for example, there is much discussion of the procedures by which one can register a copyright at the Library of Congress, even though such a registration was rendered optional by the 1976 changes). That is to say, despite the fact that the law changed in 1980, practices changed more slowly, and justifications crystallized in the context of experiments like that of GNU EMACS.

The question of what constitutes "software" is also at issue here. The most obvious tension is between the meaning of "source code" and the meaning of "software". On the one hand, the question of whether the source code or the binary code is copyrightable, and on the other, the definition of the *boundaries* of software in a context where all software relies on other software in order to run at all. Thus arise arcane questions about static and dynamic linking to libraries, operating system calls, and running software from within other software as part of the discussion.

For instance, EMACS was originally built on top of TECO (written in 1962) which was referred to both as an editor and as a programming language; hence even seemingly obvious distinctions such as that between an application and a programming language were not necessarily always clear. If TECO is a programming language, and EMACS is an application written in TECO, then it would seem that EMACS should have its own copyright, distinct from any other program written in TECO. But if EMACS is an extension or modification of TECO the *editor*, then it would seem EMACS is a derivative work, and would require the explicit permission of the copyright holder.

Further, each version of EMACS, in order to be EMACS, needed a LISP interpreter in order to make the extensible interface similar across all versions. But not all versions used the same LISP interpreter. Gosling's used an interpreter called MOCKLISP, for instance. The question of whether the LISP interpreter was a core component of the "software" or an "environment" needed in order to extend the application, was thus also uncertain, and unspecified in the law. While both might be treated as software suitable for copyright protection, both might also be understood as necessary components out of which copyrightable software would be built.[32]

[32] Later in this story, another issue of the definition of software will trouble people: the output of a parser-

What's more, the copyright status of source and binary was not addressed by the copyright law. While all the versions of EMACS were distributed in binary, Stallman and Gosling both insisted that the source be included, for slightly different reasons—Stallman because he saw it as necessary for extension and modification; Gosling, presumably, to lessen the support needs of his users, who might need to change some minor things here and there to get it to run. The threshold between modifying software for oneself, and copyright infringement was not yet clear—and hung on the meaning of "re-distribution" as well—something that could be tricky for users of a networked, time-sharing computer.[33]July 19, 1985), and this would much later become a different and trickier issue that would result in the GNU Free Documentation License.

Finally, the question of what constitutes infringement was at the heart of this controversy, and was not resolved by law or by legal adjudication, but simply by re-writing the code to avoid the question. Stallman's use of Gosling's code, his claim of third-hand permission, the presence or absence of written permission, the sale of GOSMACS to Unipress when it most likely contained code not written by Gosling, but copyrighted in his name—all of these issues complicate the question of infringement to the point where the only feasible option for Stallman to continue creating software, was to avoid using anyone else's code at all. Indeed, Stallman's decision to use Gosling's code (which he claims to have changed in significant portions) might have come to nothing if he had unethically and illegally chosen not to include the copyright notice at all (under the theory that it was original to Stallman, or an imitation, rather than a chunk of Gosling's work). However, Stallman was most likely concerned to obey the law, and give credit where credit was due, and therefore left the copyright notice attached.

In the context of the decline of trade secret and the rise of copyright, Stallman also appears to have been concerned about not only copyright, but the trade secret status of software. His

generator (sometimes called a compiler compiler). As with the case of binary vs. source material, a parser-generator takes a general description of a program (often written in a notation called Backus-Naur Form) and outputs the source code for a C (or other language) program. Is the output of this program the source or the binary? If it includes a large chunk of C code that is copyrighted by someone, does running the parser-generator constitute copyright infringement? This debate runs all the way up to the present, and one of the most colorful examples of this is the DeCSS Case. See Chapter 1 of Gabriella Coleman *The Social Production of Creative Freedom,* PhD Dissertation, University of Chicago, 2005; and the DeCSS Gallery at http://www.cs.cmu.edu/~dst/DeCSS/Gallery/

[33] An interesting addendum here is that the Manual for EMACS was also released at around the same time as EMACS version 16, and was available as a TeX file. Stallman also attempted to deal with the paper document in the same fashion (see Message-ID: 4734@mit-eddie.UUCP

assertion that he will not "look at" any code he does not have permission to distribute, seems to suggest that he understood that action in terms of trade secret law, where a violation could occur if it can be proven that someone has seen a competitor's process or product, even if the resulting product is significantly different. Trade secret law would have dictated that, if he had seen an implementation of trade secret software, then written his own version, that might still constitute violation, whereas copyright law makes no such distinction—infringement is infringement regardless of how it occurs.[34] Stallman's concerns about infringement (and his experience with the LISP Machine fights), drove him to re-invent everything without looking at anyone else's code. This was indeed a heroic feat, and helps explain his paranoia and putatively extreme micro-management style—but it is only necessary to avoid the violation of trade secrets, while a copyright infringement can occur even in the case of independent invention.

Naturally the issue of using another person's code posed a problem for future GNU projects, and it was something that must have registered in Stallman's understanding very early as a result of the controversy. How would Stallman avoid the future possibility of his own volunteers and contributors later asserting that he had infringed on their copyright? By 1986, it seems Stallman was sending out a letter that recorded the formal transfer of copyright to the Free Software Foundation, with equal rights to non-exclusive use of the software.[35] While such a demand for the expropriation of copyright might seem contrary to the aims of the GNU project, in the context of the unfolding copyright law, it made perfect sense. Having been accused himself of not having proper permission to use someone else's copyrighted material in his free version of GNU EMACS, Stallman took steps to forestall such an event in the future.[36]

[34] A standard practice well into the 1980s and even later is the creation of so-called "clean room" versions of software, in which new programmers and designers who have not seen the offending code are hired to re-implement it in order to avoid the appearance of trade secret violation. Copyright law is a strict liability law, meaning that ignorance does not absolve the infringer—but the practice of "clean room engineering" seems not to have been as successful in the case of copyright, as the meaning of infringement remains murky.

[35] Message-ID: 8605202356.AA12789@ucbvax.Berkeley.EDU
Date: Tue, 20-May-86 12:36:23 EDT

[36] The issue was not resolved there—the recent case brought by SCO against IBM, accuses IBM of inserting copyrighted code belonging to SCO into a version of the Linux kernel that was subsequently incorporated into the main code base maintained by Linus Torvalds and others. The response of Torvalds and others was exactly the same as Stallman: remove the infringing code and re-write it. However, this case has been complicated by SCO's refusal to reveal which code is infringing. In this case, whatever code is allegedly infringing does not simply contain a copyright notice with SCO's name on it, because SCO purchased the code from Novell, who

In addition to the variety of legal questions raised by this controversy, it also managed to raise several practical issues of fear, intimidation, and common-sense understandings of intellectual property law. Zimmerman's veiled threats of legal liability were not directed only at Stallman, but at anyone who was using the program Stallman had written. Whether or not such an interpretation of the law was correct, it did reveal the mechanisms whereby a generally low level of detailed knowledge about the law, and a law in flux at that (not to mention a climate of litigation that forms the reputation of the American legal system worldwide) often seemed to justify a sense that buying software was a less risky option than taking it for free, because, it was assumed, businesses would be liable for such infringements, not customers. By the same token, the sudden concern of software programmers (rather than lawyers) with the detailed mechanics of copyright law meant that a very large number of people found themselves asserting common sense notions only to be involved a flame war over what the copyright law "actually says"—even though the interpretation of that law was anything but clear at the time. Such discussion has continued and grown exponentially over the last 20 years, to the point that Free Software hackers must be nearly as deeply educated about IP law as they are about software code.[37]

Another core feature of this uncertainty was precisely the lack of organizational coherence to the emerging milieu of shared "public domain" software. While it was easy to identify corporations, research labs, and universities as responsible actors for some things, much of what was beginning to circulate came from no identifiable organization, even if it could not have been made without the material support of machines, offices, network connections etc. A significant reason for this is the growth of USEnet and ARPAnet, and the use of mailing lists and newsgroups to distribute code. Any brief scan of the earliest archives of USEnet technical groups, will reveal not only lines upon lines of software code, but also frequent requests for "public domain" versions of software that might be circulating. Generally, what people meant by "public domain" was "free" (as in beer), but just as often what was needed

purchased from AT&T in the early 1990s—it is thus code that has gone through three transfers of ownership, and it may not be clear at all who originally wrote it... or if it was produced by another piece of software. See http://www.grokline.netfor more details (than you can shake a sticky bit at) on the case.

[37] see Coleman *Social Production of Creative Freedom*, Chapter 6 on the Debian New Maintainer Process, for an example of how induction into a free software project stresses the legal as much as the technical, if not more.

was software unencumbered by corporate or organizational restrictions for the purposes of learning, re-writing, or extending something for a particular purpose.

This changing organizational ecology was also clear to Stallman, who in 1985 officially resigned from the AI Lab to start the non-profit Free Software Foundation for the creation and distribution of the software he had created. The putative purpose was to devote his energy to the GNU project—but the added protection of doing so under the auspices of an independent organization allowed him to forestall at least some potential threat that MIT (which still provided him with office space, equipment and network connection) might decide to claim ownership over his work.

One Man's Fight for Free Software

By JOHN MARKOFF

Richard M. Stallman is a computer programmer obsessed with a mission. He wants to bring back the good old days when programming was a communal activity and those toiling at the craft freely shared their ideas — and their source code, the internal instructions that tell the computer what to do.

Mr. Stallman, known among his colleagues as "The Last Hacker," has spent the last decade battling a computer software industry that increasingly builds ownership walls around intellectual property. He believes that computer software should be freely shared and devotes himself to creating sophisticated programs that he gives away.

He spends his days and nights in a cramped office at the Massachusetts Institute of Technology's Artificial Intelligence Laboratory working to spread his philosophy that software is different from other physical commodities since it can be copied at virtually no cost. He believes there should be no restrictions on freely copying and distributing it.

Gains for 'Open' Software

Mr. Stallman's ideas have gained increasing importance of late because the computer industry has been moving toward "open" software that will run on many different brands of computers. Consortiums of computer companies have formed to champion their version of the open software, based on the popular Unix computer software operating system.

But Mr. Stallman carries the idea further. Not only should the software run on different computers, but it should be free. He is doing nothing illegal, but his argument raises bitter objections from many programmers and companies. They counter that protecting intellectual property is vital to encouraging innovation.

During the last two decades intellectual property protection has become the foundation of the modern software industry. Mr. Stallman asserts, however, that what he calls "the use of human knowledge for personal gain" has had a negative impact because information is no longer widely shared.

"It's impossible to do anything without copying something that has come before," he said. "People talk about the bad effects of government secrecy in Russia. The U.S. is heading for the same place in terms of commercial software."

In a manifesto that outlines his philosophy, Mr. Stallman says that software sellers want to divide the users and conquer them by making each agree not to share with others.

"I have decided to put together a

Continued on Page D7

The New York Times/Rick Friedman

Richard M. Stallman, who is known among his colleagues as "The Last Hacker," believes software should be freely shared and spends his time at his M.I.T. office developing sophisticated programs he gives away.

Figure 1: *New York Times* Wednesday, January 11, 1989

Between 1986 and 1990, the Free Software Foundation and its software became extremely well known amongst geeks. Much of this had to do with the wealth of software that they produced and distributed via USENET and Arpanet. And as the software circulated and was refined, so were the new legal constraints and the process of teaching users to understand what they could and could not do with the software—and why it was not in the public domain.

Each time a new piece of software was released, it was accompanied by one or more text files which explained what its legal status was. At first, there was a file called DISTRIB which contained an explanation of the rights the new owner had to modify and re-distribute the software.[38] DISTRIB referenced a file called COPYING containing the "GNU Emacs copying permission notice" also known as the "GNU Emacs General Public License." The first of these licenses listed the copyright holder as Richard Stallman (in 1985); but by 1986 they all referred to the Free Software Foundation as the copyright holder.

As the Free Software Foundation released other pieces of software, the license was renamed—there was a GNU CC GPL, a GNU Bison GPL, A GNU GDB GPL etc. all of which were essentially the same terms, in a file called COPYING, that was meant to be distributed along with the software. In 1988, after the software and the licenses had become considerably more widely available, Stallman made a few changes to the license that relaxed some of the terms, and specified others.[39] This new version would become the GNU GPL v. 1.0. By the time Free Software emerged into the public consciousness, the GPL had reached v. 2.0, and the Free Software Foundation had its own legal staff.

If the creation of the GNU GPL can be understood as a kind of hack—it is one that clearly emerges out of deep and emotionally intense engagements with intellectual property law, ownership and propriety and complicated technical issues known only to very few participants. But it is also something that must be understood not just as a fix, or a workaround, but as a means of creating something entirely novel: a privately ordered legal "commune"—a space thoroughly independent of, but insinuated into the existing bedrock of rules and practices of the world of corporate and university software, and carved out of the slippery, changing substance of intellectual property statutes. At a time when the giants of the industry are fighting to create a different kind of "openness"—one that preserves and would even strengthen existing relations of intellectual property—this hack represents a radical alternative that emphasizes the sovereignty not of a national or corporate status quo, but the sovereignty of self-fashioning individuals who seek to opt out of that national/corporate unity. The creation of the GNU General public license is not a return to a golden age of small-scale communities

[38] See this message for an example: Andrew Purshottam, Mar 24, 1985 Message-ID: 5745@ucbvax.ARPA)
[39] See Message-ID: 8803031948.AA01085@venus.Berkeley.EDU

freed from the dominating structures of bureaucratic modernity—it is the creation of something new out of those structures. It relies on, and emphasizes not their destruction, but their stability—at least until they are no longer necessary.

Furthermore, the creation of the Free Software Foundation represents an interesting concession. While the image of self-determining free individuals can seem to demand the total absence of organizations, such a pure individuality was essentially impossible for Stallman (and remains impossible today). Stallmans work relied on the largesse of MITs AI lab—without it he would have no office, no computer, no connection to the network, and indeed for a while, no home. Yet the FSF represents a recognition on his part that independence must also come in the form of a legally and bureaucratically recognizable entity, set apart from MIT and responsible only to itself. Yet the creation of such an entity, and the evolution and innovation that the FSF has gone through, as well as that of other Free Software and Open Source organisctions, represent the most concentrated threat to the ideals of individual freedom out of which Free Software grows; the need for organization, and the increasing need to manage complexity makes of Free Software projects something very, very similar to corporations in practice.

Finally, the legal context has also changedbut remains deeply entwined with the outcomes of the 1976 and 1980 changes. The 1996 Telecommunications Act, the Sony Bono Copyright Term Extension Act and the 1998 Digital Millenium Copyright Act, as well as court cases from Napster and SCO vs. IBM to the supreme court Eldred and MGM v. Grokster cases are also poised to change the meaning of the practices that Free Software participants engage in, and have all transformed the legal context in ways that are not yet entirely clear.

The EMACS text editor is still extremely widely used, currently in version 21.4, and ported to just about every conceivable operating system. The controversy with Unipress has faded into the distance, as newer and more intense controversies have faced Stallman and Free Software, but the GPL has become the most widely used and most finely scrutinized of the legal licenses. More importantly, the controversy over EMACS is by no means the only such controversy to have erupted in the lives of software programmers; indeed, by now it is virtually a rite of passage for young geeks to be involved in such a controversy, because it is the only way in which the technical details and the legal details that confront geeks to

be explored in requisite detail. Not all such controversies end in the complete rewriting of source code—and today many of them concern the attempt to convince, or evangelize, for releasing code under a free software license in order to forestall infringement, or facilitate collaboration. The EMACS controversy is in some ways a primal scene—a traumatic one for sure—that has determined the outcome of many subsequent fights by giving form to the Free Software License and its uses.

Part II

Grep

If the first part of this article details the negotiation and detailed practical struggles over invention and authorship of a particular, highly successful piece of software, the second part broadens this debate beyond software to the "concepts" that are translated, or trasnduced across particular instantiations—from text to human mind to computer algorithm, to running computational process, and so on. What remains the same in the translation of concepts across media? How do the processes of authorization and legitimation allow for the stabilization of concepts in particular forms. Adrian Johns, for instance, has shown in detail how the book—the authoritative vehicle for concepts and ideas for at least the last 300 years—has come to be the kind of entity it has through cultural, social and technical processes of stabilization [Johns, 1998]. This section starts from a ubiquitous software tool called `grep`, and shows the processes of translation, authorization and re-creation that stretch back in to the early 20th century. At the end of the section, I propose a simple model for re-thinking the process of translation and authorization.

5 Regular Expressions

Part two of this article focuses on a tool that users of EMACS will be intimately familiar with. The arcane-looking title of this article (M-x search-backwards-regexp) is the EMACS command to search through some text, from end to beginning, for specific patterns described

by something known as a "regular expression."

In simplest form, regular expressions are tools for searching through text. For example, if you've ever used a word processor to search for every instance of "dissertation," and replace it with the word "book"—then you've come into contact with regular expressions. More specifically, a regular expression is a means of specifying an abstract pattern that will match any number of real instances described by the pattern. There is in fact a whole elaborate language one can learn if you wanted to do really sophisticated searching and replacing (see Fig. X).

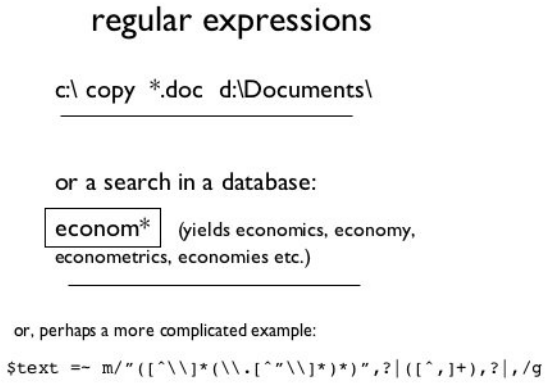


Figure 2: Species of Regular Experssionsx. Top: A familiar command in the DOS operating system; Middle: A familiar example from searching a database; Bottom: A complicated example for matching comma-separated values in a text file (from *Mastering Regular Expressions*, pg 231.)

Most normal users of editing software probably don't want to know this language, but many of us need it, and use it, anyways. Consider spam: one simple way to stop at least some spam is to feed every line of every email into a set of regular expressions that search for specific addresses, words, combinations of words, or statistically significant compositions of words, letters, phrases. Eliminate all email that has 'Viagra' in the subject line, for instance. This is a most excellent use of regular expressions. Until of course it's time to replace Viagra with Cialis or Levitra.

One shortcoming of regular expressions is that they match *only* syntactical or lexical features—there is no way to designate the class of "Erectile Dysfunction Drug names" in regexs—unless they all happen to have some kind of syntactic or lexical commonality.

Regular expressions are thus extremely useful for certain limited tasks, for instance, if one

were to write a complex document like a book (in EMACS, of course), in which it might be necessary to do a lot of searching and replacing of relatively highly structured text (especially if it were written, for example in HTML, LaTex or another text-based editing system). This is all the more true of writing a complex piece of software that includes a large number of files— imagine needing to change a variable's name, by one letter, across 700 files, and you have a good reason to use regular expressions.

In fact, once one starts to dig just a little, it becomes clear that regular expressions absolutely permeate the software infrastructure that make the internet function. This is true for two reasons. First, underneath the not particularly responsive user-interfaces that most of us are familiar with, the internet is in fact entirely made of text—strings of ordered and structured words and numbers and symbols. These conventions are both firm and for the most part, unique—making regular expressions a perfect tools for what is generally known as "pattern matching" in text.

A second reason they are so ubiquitous is due to serendipity: because a very, very large part of the internet runs on UNIX and UNIX-like operating systems, and because these operating systems come standard with a number of oddly named tools— `awk`, `sed`, and `tcl`, and of course `grep`, regular expressions are available as standard equipment on nearly every operating system or server or device a user might come into contact with, allowing programmers to create tools to search and process data and make the never-ending stupid web forms we now fill out every day of our miserable lives.

Many writers and programmers have learned regular expressions from a book called *Mastering Regular Expressions*, in the hope that they might.

Here one can learn about all the great ways in which regular expressions can be used—to process stock quotes and news-feeds, to insert and remove comments from programming code, to parse logs of information that were automatically generated by other programs or websites.

Mastering Regular Expressions, as with many such books in the land of practical programming techniques, is filled with examples, uses and details, peppered with occasional fun facts and historical anecdotes. Such handbooks can often be the only extant source of detailed historical information about the development of particular kinds of software. Certainly no one has written a history of the technique, even given its seeming ubiquity. Often there are

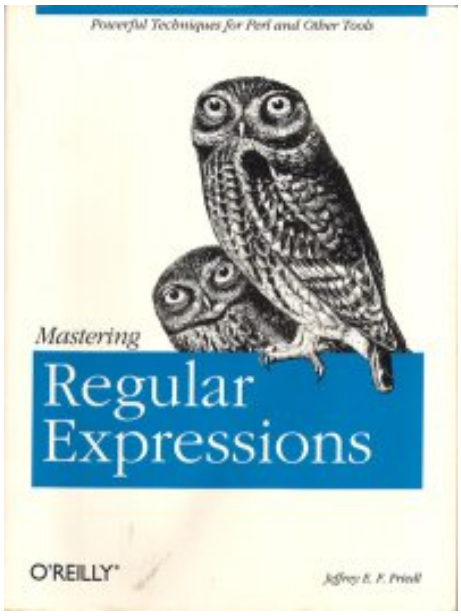


Figure 3: *Mastering Regular Expressions*, by Jeffrey Friedl [Friedl, 1997]

striking connections available to the historical analyst who might otherwise not have access to any given narrative of invention and development—and this is the case with *Mastering Regular Expressions*.

Regular Expressions are really just a set of conventions—they only work if there is an algorithm, or what is often called an "engine" behind them. In this case, the author, Jeffrey Friedl refers to these engines simply DFAs and NFAs. But then, all of a sudden in a section called, cryptically, "The Mysteries of Life Revealed" he says:

> The two basic technologies behind regular-expression engines have the somewhat imposing names *Nondeterministic Finite Automaton and Deterministic Finite Automaton.* With mouthfuls like this, you see why I stick to just "NFA" and "DFA". We won't be seeing these phrases spelled out again.

The appearance of these odd names should probably be more striking than Friedl is willing to give them credit for being: after all, what, exactly, does *automaton* mean here? Surely not android, robot, or Arnold Schwarzenegger. But the very word connotes a kind of materiality that seems radically out of place here. We are after all talking about searching for patterns in electronic text, and *automaton* seems to suggest something more like a doll, a music box, or Vaucanson's famous defecating duck. And why are they *finite*—what in the world would an

infinite robot be? Even Hollywood has to stop producing Terminator movies at some point. What exactly is the image, or model of an "automaton" at work here? From where does it issue and how does it find its way into this eminently prosaic handbook for the wage-slaves of the dot-com boom?

The short answer is that finite automata are abstract mathematical objects—they are a core feature of any computer scientist's training in writing tools that compile, parse, translate, or otherwise do systematic syntactic transformations on streams of symbols.

The long answer is the rest of this section. Ask a computer scientist about automata and he or she will probably first sigh at the thought of trying to explain it, then mumble something about a theory of machines, about computers and language and brains as if it were all just self-evident that one might speak of automata this way. But scientists and programmers steeped in this language are often to far inside it to be surprised by it, much less to explain it.

There is, however, a fascinating story of invention, translation (or perhaps "transduction") and creative appropriation to be told here—one that begins in the present with the ubiquitous use of regular expressions and the common program `grep` (so common geeks use the term as a verb—as in "I grepped for my socks, but found only this mis-matched pair") and ends up with the *Principia Mathematica* of Whitehead and Russell, after traversing neurology, the patent system, recursive algebra, the EDVAC computer, and the science of "automata theory." The peregrination of the concept is such that it raises curious and interesting questions about what it means for the "same" concept to traverse different media, machines, minds and disciplines, and remain somehow identical to itself accross all that. Rather than a story of the universality of a scientific idea (in this case, a particular kind of logic) it reveals the endlessly creative and productive power of ideas treated as objects for renewal and transformation. The geneaology I trace here looks roughly like (Fig X).

The origin of this investigation—the program called `grep`—can start on just about any UNIX-based computer, including today, the Macintosh operating system. One nice thing about the ubiquity of UNIX is that it actually makes it very easy to trace at least part of this geneaology back to about the mid 1980s simply by digging around on directories included with the Linux operating system. The widespread availability of Linux, and the openness and

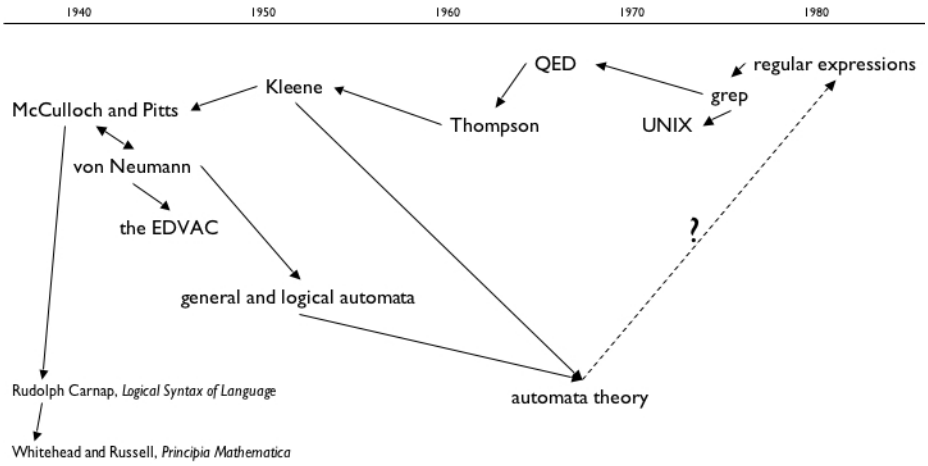


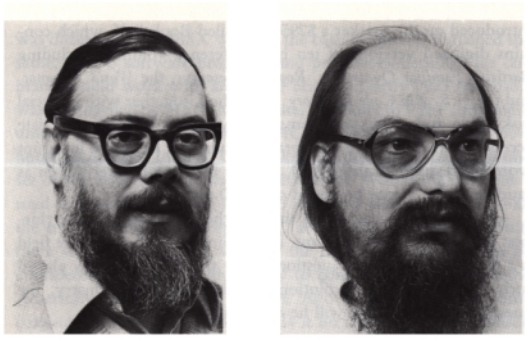
Figure 4: Geneaology of Regular Expressions

the relatively rigorous practice of retaining every change and every version in a public place makes such research delightfully easy.

Beyond a certain point, however, it can become much, much more difficult to trace the people, the tools, the technologies and their histories. Not all such tools begin life as academic projects, and so there is often no literature on them—and many tools, epecially commercial tools, simply disappear when the corporations that make them do so, leaving traces, but no documentation of their impact.

For a variety of reasons, the UNIX operating system is a major exception here. In its various forms, UNIX and derivatives are easily the most widely distributed and the most well documented (historically speaking) of systems. And a remarkable amount of it was initially created by academics and by a relatively short list of people at Bell Labs in the 1970s, including its two gloriously nerdy inventors, Dennis Ritchie and Ken Thompson (Fig X).

So, with a bit of searching through the literature and the operating systems, it becomes clear that all of the various modern implementations of regular expressions can be traced back to implementations of the three programs `sed, awk,`and `grep`. `grep` is the earliest of these,



Dennis Ritchie Kenneth Thompson

Figure 5: The inventors of the UNIX Operating System Environment, at their peak.

and is a general purpose tool written very early on in UNIX development—probably in the early 1970s. The name itself derives from a set of commands that would have been issued in another program, a text-editor called QED. In QED, in order to search for a pattern you would type:

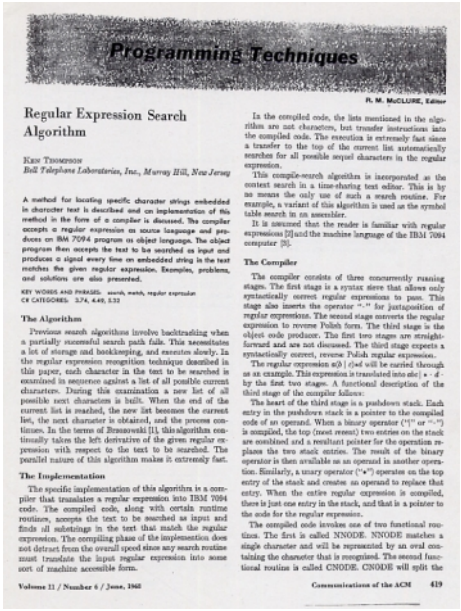
G/*re*/**P** (**G**lobally search for regular expression *re* and **P**rint it)

QED was written initially by Butler Lampson and Peter Deutsch, two computer engineers at UC Berkeley. The ability to "grep" for patterns was added by one of their students: the same Ken Thompson. At the time, Thompson had just started at Bell Labs, working on a famous computer project at MIT called, at first, the Compatible Time Sharing System (CTSS), and later called Multics. In about in 1967, Bell Labs pulled its support from the MIT project, leaving Ken Thompson and his buddy Dennis Ritchie with nothing to do and no computer to hack on.

Huddled in the attic of Bell Labs' Murray Hill, New Jersey headquarters, they scrounged around and came up with a cheap mini-computer, a DEC PDP-7. The PDP-7 was a very limited machine, but this did not stop them from creating the first version of what would become one of the most successful and revered operating systems ever, UNIX (which was a pun, meaning a castrated Multics).

In the course of all this, Thompson also managed to file for and receive a patent, #3568156, and write a paper documenting the algorithm he implemented in QED. The patent was filed in 1967(granted 1971) and the paper was published in the *Communications of the Association for*

Computing Machinery in June of 1968, and was called "Regular Expression Search Algorithm" [Thompson, 1968].



Programming Techniques

R. M. McCLURE, Editor

Regular Expression Search Algorithm

KEN THOMPSON
Bell Telephone Laboratories, Inc., Murray Hill, New Jersey

A method for locating specific character strings embedded in character text is described and an implementation of this method in the form of a compiler is discussed. The compiler accepts a regular expression as source language and produces an IBM 7094 program as object language. The object program then accepts the text to be searched as input and produces a signal every time an embedded string in the text matches the given regular expression. Examples, problems, and solutions are also presented.

KEY WORDS AND PHRASES: search, match, regular expression
CR CATEGORIES: 3.74, 4.49, 5.32

The Algorithm

Previous search algorithms involve backtracking when a partially successful search path fails. This necessitates a lot of storage and bookkeeping, and executes slowly. In the regular expression recognition technique described in this paper, each character in the text to be searched is examined in sequence against a list of all possible current characters. During this examination a new list of all possible next characters is built. When the end of the current list is reached, the new list becomes the current list, the next character is obtained, and the process continues. In the terms of Brzozowski [1], this algorithm continually takes the left derivative of the given regular expression with respect to the text to be searched. The parallel nature of this algorithm makes it extremely fast.

The Implementation

The specific implementation of this algorithm is a compiler that translates a regular expression into IBM 7094 code. The compiled code, along with certain runtime routines, accepts the text to be searched as input and finds all substrings in the text that match the regular expression. The compiling phase of the implementation does not detract from the overall speed since any search routine must translate the input regular expression into some sort of machine accessible form.

In the compiled code, the lists mentioned in the algorithm are not characters, but transfer instructions into the compiled code. The execution is extremely fast since a transfer to the top of the current list automatically searches for all possible sequel characters in the regular expression.

This compile-search algorithm is incorporated as the context search in a time-sharing text editor. This is by no means the only use of such a search routine. For example, a variant of this algorithm is used in the symbol table search in an assembler.

It is assumed that the reader is familiar with regular expressions [2] and the machine language of the IBM 7094 computer [3].

The Compiler

The compiler consists of three concurrently running stages. The first stage is a syntax sieve that allows only syntactically correct regular expressions to pass. This stage also inserts the operator "." for juxtaposition of regular expressions. The second stage converts the regular expression to reverse Polish form. The third stage is the object code producer. The first two stages are straightforward and are not discussed. The third stage expects a syntactically correct, reverse Polish regular expression.

The regular expression a(b | c)*d will be carried through as an example. This expression is translated into abc| * . d· by the first two stages. A functional description of the third stage of the compiler follows:

The heart of the third stage is a pushdown stack. Each entry in the pushdown stack is a pointer to the compiled code of an operand. When a binary operator ("|" or ".") is compiled, the top (most recent) two entries on the stack are combined and a resultant pointer for the operation replaces the two stack entries. The result of the binary operator is then available as an operand in another operation. Similarly, a unary operator ("*") operates on the top entry of the stack and creates an operand to replace that entry. When the entire regular expression is compiled, there is just one entry in the stack, and that is a pointer to the code for the regular expression.

The compiled code invokes one of two functional routines. The first is called NNODE. NNODE matches a single character and will be represented by an oval containing the character that is recognized. The second functional routine is called CNODE. CNODE will split the

Volume 11 / Number 6 / June, 1968 Communications of the ACM 419

Figure 6: Ken Thompson's ACM Paper on Regular Expressions

Thompson's paper is quite difficult to read—not only because it involves a complicated subject unfamiliar to non-computer scientists, but qbecause it is represented largely in assembly language for the IBM 7094 computer—a language and a computer last in production in about 1965.

Nonetheless, a bit of 7094 Assembly code later, it is still not quite clear why regular expressions in the creation of a program for matching patterns in text should be referred to by the terms "Non-deterministic and deterministic finite automatons." The key, however, is on one of the four brief citations to this short paper:

> Steven C. Kleene "Representations of events in Nerve Nets and finite Automata," in *Automata Studies*, Ann. Math. Stud. No. 34, Princeton University Press, 1956.

Here, at least, is the single reference in the paper to a "finite automaton"—and to make matters even more intriguing, it appears to be a paper on the subject of "nerve nets" and automata. This paper, it turns out, is a sort of hidden classic. It is a complicated text and one

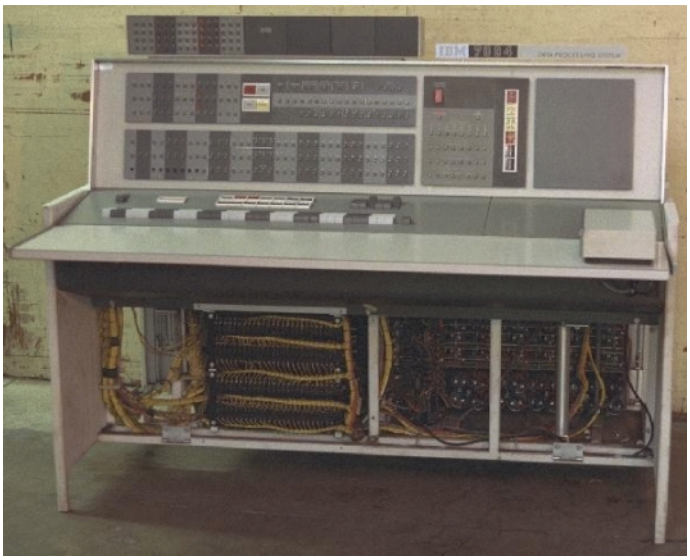


Figure 7: The IBM 7094

that represents a very important point of translation in the genealogy of software, algorithms and networks. Stephen Kleene was a graduate student at Princeton in the 1930s, which was an auspicious time to be a graduate student at Princeton, surrounded by the likes of Alan Turing, Kurt Goedel, Alonzo Church, not to mention Einstein and Von Neumann. He worked for years on the subject of recursive functions, which were an outgrowth of these tumultuous times.

Just for a moment, before exploring Kleene's paper, I want to dwell on what exactly Ken Thompson did with it. Kleene's paper is a mathematical paper: published in the Annals of Mathematical Studies, filled with definitions and theorems, lemmas and derivations. It is a paper that describes an algebra—that is a set of operators, and some theorems by which one can transform any given set of numbers (or symbols) into another. The paper says absolutely nothing about computers, programming, matching patterns in text or searching for text on a computer—the paper was not even composed on or near a computer, yet somehow Thompson recognized that this algebra was a tool for efficiently *matching patterns in text*. Thompson employs the algebra to create his software—one might say this is an obvious case of science being "applied" to create technology. The problem is that whatever it is Kleene's paper sets out, proleptically (so to speak) as a theory, is not what Thompson does with it. Rather he *translates* Kleene's algebra into an algorithm, a patent and a paper.

So what *is* Kleene's paper about? Well, the title claims it is about nerve nets and finite *automatons*. But, you might ask, why was a mathematician modeling neurons to begin with? For most of his life, Kleene taught at the University of Wisconsin, Madison, but in 1951, he spent a year at RAND, the defense department think-tank responsible for all manner of Cold War mathematical, economic, nuclear strategy and operations research. Merrill Flood, his sponsor at RAND, who had also been a grad student at Princeton, had given him a paper to read: this paper was called "A Logical Calculus of the Ideas Immanent in Nervous Activity " by Warren McCulloch and Walter Pitts[McCulloch and Pitts, 1943].

Upon reading this paper, Kleene's first inclination as a mathematician was to push further the model it contained—a model of the brain as a logical calculus (not a machine exactly). Kleene wanted to formalize it into an algebra that would mathematically describe the network of neurons that McCulloch and Pitts had proposed as a model. In a reflection from 1981, Kleene said:

> I found [McCulloch and Pitts'] model to be very interesting —an original contribution — but their analysis of it to fall quite short of what was possible. So I did what cried out to be done with it (as it seemed to me)... [Kleene, 1956, pg ?]

What "cried out to be done" however, was probably not what McCulloch and Pitts would have cried out for. So, here I want dwell for a moment on what exactly Kleene did with McCulloch and Pitt's paper:

> Our theoretical objective is not dependent on the assumptions [about the neurophysiological data] fitting exactly. It is a familiar stratagem of science, when faced with a body of data too complex to be mastered as a whole, to select some limited domain of experiences, some simple situations, and to undertake to construct a model to fit these at least approximately. *Having set up such a model, the next step is to seek a through understanding of the model itself.* [Kleene, 1956, pg ?]

Kleene was not interested in whether McCulloch and Pitt's model was accurate (i.e. whether it really represented the state, function or structure of a neuron or a brain)—what he takes from the paper is a *model* of a brain that was, for him, not yet fully explored in mathematical terms—and he spends the bulk of his paper doing so. For Kleene's purposes, it need not be a model of a brain—only a model.

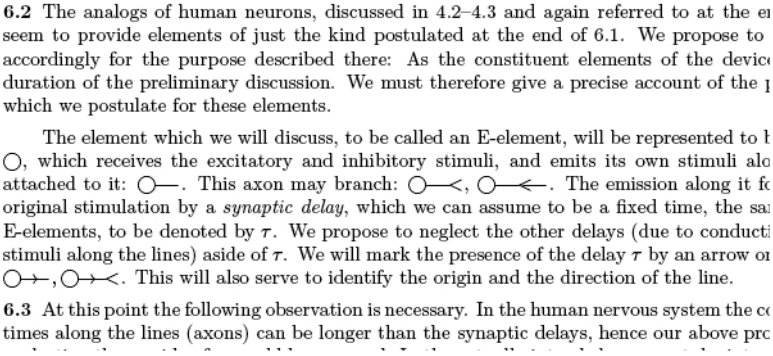
For those who do not recognize it, McCulloch and Pitts' paper is a very famous document. Often cited as the origin of the cybernetics movement, the origin of the field of cognitive science and as one of the core contributions to mathematical biology, it is at once a very technical paper, and a very speculative one. In the thirties, McCulloch was a doctor doing research on neurons—by the early 1940s he had begun the outlines of a theory of the brain in which neurons, connected together in the brain, could be said to each occupy one of only two states (either firing or not firing). Such a model certainly lent itself to a certain formalization—but it wasn't until he enlisted Walter Pitts—a student at the time of Rudolph Carnap (of the Vienna Circle Logical Positivists)—in order to actually work out and publish this logical calculus. The system they used to describe a connected net of neurons firing was Rudolph Carnap's *Logical Structure of Syntax* ("Language II") and the formal system of Russell and Whitehead's famous *Principia Mathematica*[Carnap, 1937] and [Whitehead and Russell, 1910]. These two formalisms by Carnap and by Russell and Whitehead were meant to capture, in the first case, the logical structure of language, and in the second, the logical structure of mathematics. Neither of them are attempts to describe the logical structure of the human brain (even if they might have been taken to imply this in a metaphysical sense).

So Kleene had translated McCulloch and Pitts brain into an algebra, which Thompson then translated into an algorithm, a patent, and a text editor. We could extend the series backwards even further by looking at what Carnap and Whitehead and Russell had hoped to achieve, and how McCulloch and Pitts transformed it into a brain—including the key intervening work by Turing. But let's stop here, and return to the question of the finite automaton. McCulloch and Pitts paper does not refer to the brain as an automaton—nor does it discuss computers, calculating machines, or even, at any point refer to the model they have created as machine-like, automatic or algorithmic—only as logical. Kleene's gets the connection from yet another important paper:

> I found [McCulloch and Pitts'] model to be very interesting —an original contribution — but their analysis of it to fall quite short of what was possible. So I did what cried out to be done with it (as it seemed to me), *having newly in mind also the idea of a finite automaton, which came to me that summer at RAND through reading in printer's proof (?) Von Neumann's Hixon Symposium lecture.* [Kleene, 1956, pg ?]

John Von Neumann's Hixon Symposium lecture of 1948 is better known as his "general and logical theory of automata"—a text that was not actually published until 1966 (as the question mark signifies), but was delivered at a famous symposium on cerebral mechanism—a founding moment of cybernetics as the participants included McCulloch, von Neumann, Norbert Wiener, Karl Lashley and others.

Von Neumann's paper is the first real attempt to deal with, as it says, a general theory of automata—both animals and machines—and it is filled with a mathematician's observations about the relationship of mathematical calculation to physical and mechanical constraints. Von Neumann's interest in the similarities and differences between living organisms and artificial automatons was profound—and perhaps the best evidence for this is yet another document he wrote (for which he is also justly famous) called "the First Draft of a Report on the EDVAC." This document—and it was indeed a draft, circulating only in mimeograph form for years—is the document generally understood to be the founding design document of what is now called the "Von Neumann architecture" of computers, in which the CPU is separated from the memory and the input and output channels. It was created in collaboration with (some say purloined from) Eckert and Mauchly—who had actually built a real, mostly functioning version of a computer called the ENIAC, at the Moore School in Pennsylvania. Such



6.2 The analogs of human neurons, discussed in 4.2–4.3 and again referred to at the e
seem to provide elements of just the kind postulated at the end of 6.1. We propose to
accordingly for the purpose described there: As the constituent elements of the devic
duration of the preliminary discussion. We must therefore give a precise account of the
which we postulate for these elements.

The element which we will discuss, to be called an E-element, will be represented to
○, which receives the excitatory and inhibitory stimuli, and emits its own stimuli alo
attached to it: ○—. This axon may branch: ○—≺, ○—≺—. The emission along it f
original stimulation by a *synaptic delay*, which we can assume to be a fixed time, the sa
E-elements, to be denoted by τ. We propose to neglect the other delays (due to conducti
stimuli along the lines) aside of τ. We will mark the presence of the delay τ by an arrow o
○→—, ○→≺. This will also serve to identify the origin and the direction of the line.

6.3 At this point the following observation is necessary. In the human nervous system the c
times along the lines (axons) can be longer than the synaptic delays, hence our above pro

John von Neumann, First Draft of a Report on the EDVAC, 1945

Figure 8:

a document might fairly be expected to be boring. Dry. Schematic. It is indeed all of these

things, but it is also fascinating: instead of simply describing the design of a computer in terms of vacuum tubes, wires, plugboards and relays—the report uses McCulloch and Pitts diagrams of *neurons* as its formalism. Von Neumman directly imported the concept of a Logical calculus of nerve nets into the design of the EDVAC. There is no necessary reason why he needs to do this—but it does give an otherwise strictly technical document a certain air of madness. The EDVAC formed the basis for the first real, universal computing machines—and the beginning of decades of comparisons of human brains and computers.

What comes of this is not artificial intelligience however, but a general logic-based, mathematical theory of automata—machines that execute algorithms. This theory becomes automata theory—the basis of a great deal of work in computer science, but it does so only via this series of odd, media-hopping translations. Between 1950 and 1958, there was as a result largely of Von Neumann's synthesis, an explosion of interest in the mathematical models that were being called *automata.* For instance, the book in which Kleene's paper appeared was called *Automata Studies*, and it contained contributions from Von Neumann, Claude Shannon, Marvin Minsky, Ross Ashby and others. About the same time, Von Neumann published a popularization called "The Computer and the Brain."

This fascinating moment is also the point at which *linguistics* gets involved in the person of Noam Chomsky, who as a result of work on finite automata, is somewhat dimly recognized as one of the fathers of computer science, as well as linguistics. Chomsky worked also with George Miller, one of the patriarchs of modern cognitive science, and there are other well known names—Benoit Mandelbroit, Marcel Schutzenberger, Willard Van Ormand Quine, Yehoshua Bar-Hillel, Roman Jakobson etc. In the end, however, the study of automata as a model of human brains diminished in almost inverse relationship to the growth of interest in the characteristics of the abstract model itself: so that today, the term, and the concept of a finite automata is easily as complex, baroque and entertaining (at least to math geeks) as the mechanical toy automata of the 18^{th} and 19^{th} century.

Here, then, is the story of regular expressions: McCulloch and Pitts translated Carnap, Whitehead and Russell into a brain, Kleene translated McCulloch and Pitts into an algebra, Thompson translated Kleene into a patent, a paper and an text-searching tool in an editor called QED which spawned `grep`, which has proliferated regular expressions to the point of

being nearly ubiquitous today. QED.

6 Conclusion

This set of translations is a fascinating, if obscure story. It is not the *only* story of such translations however: one could pick other examples—databases for instance, or email, or networked computing itself, and the details of the story will reveal translations between philosophies, models, mathematics, algorithms, programs and machines.

My interest in these stories, however, is not simply in the history of machines or disciplines or sciences. It is all too easy to cordon off computers and software as inscrutable machines with highly complex and specific histories. Computers and software, however, are simultaneously machines and media. Like Xerox machines, phonographs, cameras, typerwriters, pencils and presses before them, they transform the practice of *composition*: from abstraction to creation, from translation to translation, these media make things mean differently.

In the social sciences and humanities (and in particular in the field of the history and social study of science) our modes of analyzing the relationship of media to meaning is terribly impoverished. For most scholars, the story of regular expressions is, at worst, a story of "metaphors" and at best, a story of creative appropriation. As with longstanding debates about the relationship of information theory and genetics ([Doyle, 1997, Kay, 2000, Keller, 2000], this story seems to be about whether the replacement, for example, of a logical calculus with a model of a neuron is "motivated" or somehow itself logical. A certain kind of belligerently naive realism seems to prevent scholars from recognizing that when McCulloch and Pitts describe human neurons as a calculating machines, *they are not being metaphorical.* It is not a mis-interpretation, a divergence, a decline, or a demonstration of their naive ideological commitment to realism. What this story shows, by contrast, is an incorrigible *productivity*— a set of translations through which the innovation that was proposed in one text is ignored in favor of a very different innovation in the next: start with an attempt to describe human brains as logical systems, but end with a nearly ubiquitous technology that does not at all require one to subscribe to the theory that a brain is logical. I can search through text using regular expressions until my eyes melt—and I never have to even entertain

the thought that I am using any brain except my own to do so.

How then, should we describe these translations? Consider a familiar notion from media theory: what Marshall McLuhan and Elizabeth Eisenstein refer to as the *fixity* of printed texts [McLuhan, 1962, Eisenstein, 1979]. This is the notion that the printing press introduced a new kind of stability into texts—giving them a power and authority as stable entities that they did not have prior to the invention of the printing press, when manuscripts were ostensibly "corruptible" through repeated hand copying. Whether this stability is understood as a feature of the technical characteristics of the medium, or of a more general social and cultural transformation depends very much on your reading of these scholars. Nonetheless, the point remains: it is *texts*, they assert, that have authority—very rarely is an individual's memory of a text going to take precedence over a written and fixed version.

But one can challenge this notion of fixity in at least two ways. One is to focus in detail, historically, on the social and cultural process by which a particular text becomes authoritative—who reads it, who uses it, how does it become a standard or a canonical text. Adrian Johns work *The Nature of the Book*, which is a fantastic source for precisely this kind of critique, lays out how western European printed texts were made to be trustworthy and credible—and it was not without considerable energy and effort by a pretty diverse set of interested people[Johns, 1998]. The fact that **this** book, or any book, is perceived as reliable or authoritative is a *social, cultural, economic and political* achievement, not only a *technical* one.

A second way to challenge the supposed fixity of books, however, is to focus less on the relationship of media to text, or text to context, and more on the temporal and rhetorical (in the classical sense) process of composition, and the temporal and social process of authorization.

Take, for example, the work of Mary Carruthers as a guide in this respect[Carruthers, 1998]. Carruthers is a medieval historian who has a remarkably clear understanding of the process through which texts are not only *written*, but exist as memorized compositions and as *spoken* and *recited* objects. For instance, the relationship of the written copy of Augustine's *Confessions* to the one in Augustine's memory is different than it would be in the era of reliable printed books, and yet again in the era of electronic texts. It is true, the text—as in the literal sequence of letters—may end up being exactly the same in all three eras (though this

is highly unlikely), but what changes however, is the determination of *which version* (and at which time) is considered most authoritative—or of how the text can be inserted into other authors' compositions and thereby circulated, discussed, innovated, transformed.

Similarly, as I've shown here, algorithms are not simply written in books or papers, but are communicated as concepts, implemented in various programs, learned by analyzing programs, (not to mention licensed from patent or copyright holders) and they are also memorized (that is, we would say today, *understood*) by mathematicians, scientists, programmers and others who can speak of them with relative ease. The fact that such objects have lives outside of the text means that we should probably re-conceive of their relationship to both humans and media.

I've tried to diagram what I'm getting at here, so consider this image of *fixity:*

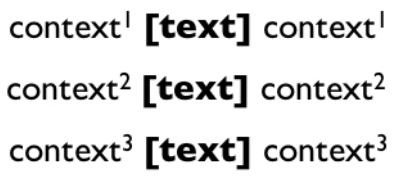


Figure 9: Texts and their contexts

Here a single text is inserted into several different contexts: these may be temporal or spatial or disciplinary, etc. but the text remains the same: think of all the wonderful studies, for instance, of the reception of Homer or Virgil in various places and epochs or the receptions and stagings of Shakespeare plays in different eras. Here the text is treated as, if not stable, at least identical with itself; a text against which multiple meanings and effects are charted. Texts become, in this case, kind of like monuments, which in turn leads to the emergence of things like oeuvres, masterworks, inventions and, for Foucault, even authors. Meanings may change, different effects may ensue, but the text remains the same.

Opposite this, however, one might pose a different notion of the relation between humans and their media— one in which the temporal process through which a text might be

transformed is rendered more visible through a social and cultural and technical process of translation

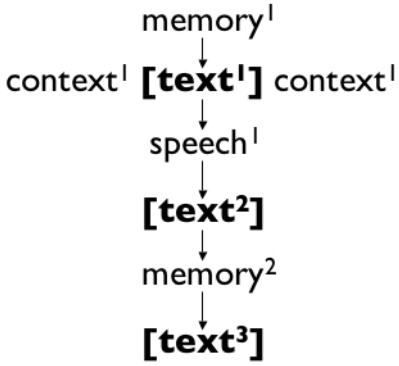


Figure 10: Expansion of the concept of "context"

One might think of this as the expansion of the notion of context, either in space or time. Instead of a stable environment that sits ready to accomodate (or reject) a text, this diagram demonstrates the path from one context to the next—and sets up the question of whether or not it is the context or the text that stabilizes the meaning and identity of concepts and ideas. Indeed, it might be extraordinarily difficult to speak of text^1 and text^2, for instance, being the "same" text. The translation could be a translation from one language to the next, a re-writing of the text, a recording or the text, or a natural language algorithms' analysis of the text. And yet there is a process by which one leads to the next, without which nothing at all would get written. The stability of the text is not what accounts for this process of translation—indeed, these texts may be fixed or immortal in some kind of material or institutional sense (the work of Shakespeare is by now so stable and monumental as to be thoroughly resistant to radical translation—much less disappearance, but other kinds of works do not have that stabolity and authority), but it is not the texts–and certainly not the books, which "contain" the ideas, concepts, or meanings that clearly result as a process of reading and writing, translating and innovating.

This is all the more evident in the story of regular expressions:

There is no inventive origin point for "regular expressions." What's more it is not clear what allows the series of transformations that take place—from philosophy to neurology to

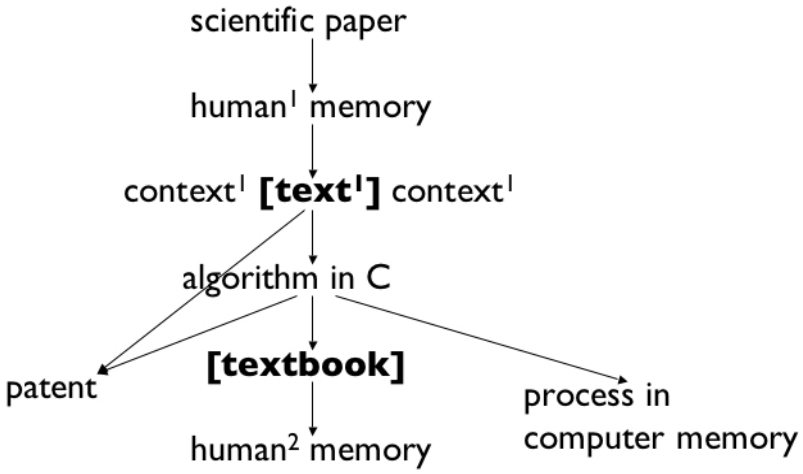


Figure 11:

algebra to a patent application to a software algorithm and so on. What is clear about this example (and what differentiates it from many other possible examples, as well as makes the geneaological tracing possible) is that the "concept" (tool?) of regular expressions has become extremely stable, much like the oeuvre of Shakespeare. Indeed there are probably considerably more copies of `grep` out there in the world's computers than there are of the collected works of Shakespeare—by virtue of the ease of replication of this tool and its standard inclusion in operating systems. Should we identify these copies (which are, by the way, considerably less homogenous than Shakespeare's oeuvre, and consist of a wide variety of derivatives and similar programs) as the container of the concept "regular expressions"? Or should we privelege the algebraic presentation of Kleene's from 1956? Indeed, nothing less than a question of platonic idealism confronts this story: are all these translations and re-uses simply copies of an originary simulacrum of a metaphysical idea? Many computer scientists would adopt this conclusion—with the (textual) mathematical exposition of regular expressions occupying that of the most faithful copy. But many others would object: an algorithm is only an algorithm if it can achieve existence as a process—whether in a computer or in a brain, and so it is the execution of regular expressions that might veer closest to the Platonic ideal. And then again, the translations of McCulloch and Pitts, and of Kleene, and of Thompson are not motivated by any such vulgar Platonism—they are avowedly creative pursuits, meant to build, observe, explore and test models, algebras, algoritms, patent applications and text editors. Alongside

this creativity is a social and cultural context for the authorization and legitimation of certain kinds of texts, objects, productions over others: different interpretive communities that can provisionally agree on certain forms of expression, a publishing industry, a patent system, and an operating system called UNIX: all of these things are necessary to authorize and stabilize the concept of regular expressions.

To return to the first part of the chapter, it is clear that, alongside the conventional scientific and technical systems of legitimation and authorization, the UNIX operating system must be included. Why Unix and not Windows or the IBM/360? Because of its techniques of reproduction and stabilization–techniques that are primarily legal and social, not technical UNIX (and EMACS and *grep* alongside it) have evolved as relatively open objects, through which generations of computer engineering students, hackers and geeks, have come to understand, to *memorize*, and to reproduce the concept of "an operating system," a "text editor," or "regular expressions algorithms."[40] The intersection of legal and social practices of authorizing, legitimizing, learning, copying, writing and re-creating visible in the example of the EMACS controversy, is a paradigm case of the authorization not only of certain tools, ideas, algorithms or software— but of a process of authorization as well, one that is today clearly visible in the emergence of Free Software and Open Source Software.

References

[Brooks, 1975] Brooks, F. (1975). *The Mythical Man-Month: essays on software engineering.* Addison-Wesley, Reading, MA.

[Carnap, 1937] Carnap, R. (1937). *The Logical Syntax of Language.* K. Paul, Trench, Trubner & co., ltd., London.

[Carruthers, 1998] Carruthers, M. (1998). *The Craft of Thought : Meditation, Rhetoric, and the Making of Images, 400-1200.* Cambridge University Press, Cambridge, U.K. ; New York, NY.

[40] I detail this history in [Kelty, nd]

[Doyle, 1997] Doyle, R. (1997). *On beyond living : rhetorical transformations of the life sciences*. Stanford University Press, Stanford, Calif.

[Eisenstein, 1979] Eisenstein, E. L. (1979). *The printing press as an agent of change : communications and cultural transformations in early modern Europe, 2 vols.* Cambridge University Press, Cambridge [Eng.] ; New York.

[Friedl, 1997] Friedl, J. (1997). *Mastering Regular Expressions: Powerful techniques for Perl and other tools*. O'Reilly Press, Sebastopol, CA.

[Johns, 1998] Johns, A. (1998). *The Nature of the Book: print and knowledge in the making*. Chicago University Press, Chicago.

[Kay, 2000] Kay, L. E. (2000). *Who wrote the book of life? : a history of the genetic code*. Stanford University Press, Stanford, CA.

[Keller, 2000] Keller, E. F. (2000). *The century of the gene*. Harvard University Press.

[Kelty, nd] Kelty, C. M. (n.d.). *Two Bits: The Cultural Significance of Free Software*. Manuscript.

[Kleene, 1956] Kleene, S. C. (1956). Representations of events in nerve nets and finite automata. In *Automata Studies*, number 34 in Annals of Mathematical Studies. Princeton University Press, Princeton.

[Knuth, 1997] Knuth, D. (1997). *The Art of Computer Programming, 3rd ed.* Addison-Wesley, Reading, MA.

[Levy, 1984] Levy, S. (1984). *Hackers: Heroes of the Computer Revolution*. Penguin.

[McCulloch and Pitts, 1943] McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–134.

[McLuhan, 1962] McLuhan, M. (1962). *The Gutenberg galaxy; the making of typographic man*. University of Toronto Press, Toronto.

[Thompson, 1968] Thompson, K. (1968). Regular expression search algorithm. *Communications of the Association for Computing Machinery*.

[Whitehead and Russell, 1910] Whitehead, A. N. and Russell, B. (1910). *Principia mathematica, 3vols.* Cambridge University Press, Cambridge, UK.