The Art and Craft of Hacking

Gisle Hannemyr
gisle@ifi.uio.no
University of Oslo

Abstract

"Hackers" are identified as software developers sharing a specific work practice. The process of hacking and the characteristics of the resulting artifacts are discussed. Some research questions following from these findings are posed.

Introduction

> The word 'hack' doesn't really have sixty-nine different meanings. In fact, 'hack' has only one meaning, an extremely subtle and profound one which defies articulation. (Steele *et al* 1983)

The meaning of the words "hacking" and "hacker" as applied to computer work is not very clear. Webster defines "to hack" as "to cut with repeated irregular or unskilful blows", and "a hacker" as "one who forfeits individual freedom of action or professional integrity in exchange for wages or other assured reward". These definitions, however, bear no resemblance to the common usage of the words "hacking" and "hacker" in the context of computer work.

Part of the confusion surrounding the word "hacker" may stem from the fact that it has been applied to at least three distinct communities: Computer workers subscribing to a common set of values and a shared culture; activists viewing the computer as an instrument for political empowerment; and digital vandals who break into computer systems for fun and profit (Hannemyr 1999).

In this essay, I shall ignore the political and criminal aspects that popular media unfortunately has managed to make synonymous with "hacking". Instead, I

© Scandinavian Journal of Information Systems, 1998, 10(1&2):255–262

shall focus on "hacking" as a work practice, and on "hackers" as software development practitioners.

My motivation for writing this essay is primarily to give this practice the recognition I believe it deserves. Hackers have created a large body of non-trivial computer software. Still, hacking is little mentioned in literature describing miscellaneous software development practices. A secondary consideration is my belief that hacking or hacker-like approaches in some circumstances can bring specific qualities to software development processes and to the resulting artifacts. To understand these qualities and their application is a challenge to current software development research.

Samplings

> Linux is subversive. Who would have thought even five years ago that a world-class operating system could coalesce as if by magic out of part-time hacking by several thousand developers scattered all over the planet connected only by the tenuous strands of the Internet. (Raymond 1998)

Software created through the work practice known as "hacking" is implemented by and within self-organising communities through a highly iterative process where development of new and adaptation of existing software components are equal and integral parts of the development process. Development phases such as "implementation", "debugging", "usability testing", "release" and "maintenance" are collapsed into an ongoing, all-encompassing and sometimes perpetual "hacking" phase. Words such as "developer", "programmer" and "user" are used interchangeably to signify oneself and other process participants (applying the term "hacker" to oneself is considered hubris). The derogatory term "luser" is sometimes used to identify individuals who "only" want to use the software, but chooses not to participate in the community effort surrounding it.

The Linux operating system kernel is the most publicised example of software produced through the practice of hacking (IEEE 1999). Other significant developments include the TeX and LaTeX typesetting systems, large portions of the Unix operating system, and the body of communication software that evolved to become the Internet. Of particular interest is a community effort known as the GNU project, which has organised the efforts of literally thousands of programmers to implement several hundred software systems, ranging from simple games (e.g. nethack) to massive software development tools such as the GNU G++ kit, consisting of context sensitive editors, standard libraries, several compilers, debuggers and profilers.

Comparing and contrasting these software artifacts to counterparts created outside the hacker community indicate (Hannemyr 1999):

1. Software created by hackers has in common such usage properties as tailorability, adaptability and openness. Commercial software, on the other hand, favours such qualities as extensibility, completeness and immutability.
2. Software created by hackers often requires a steep learning curve, and has little visual appeal. Commercial software is deliberately designed to appear more "user friendly", and

focus more on production values (i.e. use of colour, typography, animations and graphical design).

3. Software created by hackers is often technically superior (sampling such aspects as crash rate and availability of task-oriented features). Most users (sampling actual choices made by non-hacker users when presented with alternatives) prefer commercial software.

Explaining these findings warrants further research. My preliminary hypothesis is that the steep learning curve and the lack of production values that characterises most software implemented by hackers, together with the denigrating attitude towards "lusers", partly explain why these software artifacts have limited appeal to users outside the hacker community. Also, the present installed base of end user software is commercial packages, whose extensibility, completeness and immutability (deliberately?) make interoperation with foreign software artifacts exceedingly difficult.

The Hacker Ethic

> [Tim Berners-Lee] didn't patent the [World Wide Web]. He didn't copyright. He made that openly available. And that's what has fuelled a great deal of the network development, and all the innovative ideas. [...] There is a continuing ethic in the community to give back to the network what it has given you. (Cerf 1997)

The idea of computer software as a communal resource has been one of the identifying traits of the hacker community since its inception in the sixties. In these early days, most software was "given away" as an appendage to the hardware it ran on. There existed virtually no market for software and since computer configurations was not well standardised and operating systems and application software had to be distributed as source format and adapted to each installation on the site. Both the availability of source code and the lack of price tags and formal licensing affixed to the software created an environment where is was legitimate for programmers to work on improving and adapting operating systems and applications by "hacking" the source code; and to freely share among themselves the improved programs, fragments and algorithms that resulted from such activities.

As the software industry matured, the software sector became increasingly commercial. Formal licensing agreements, binary only distributions and non-disclosure agreements became the norm rather than the exception. Richard M. Stallman gives this description of how many hackers felt about these developments:

> Many programmers are unhappy about the commercialization of system software. It may enable them to make more money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now typically used essentially forbid programmers to treat others as friends. (Stallman 1985)

The commercialisation of the software industry prompted Stallman to quit his job at the MIT AI lab and to write The GNU Manifesto. Part autobiography and part call to arms, the manifesto outlines

the rationale behind his resolve to create free software:

> I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. [...] So that I can continue to use computers without dishonor, 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 (ibid.).

In the manifesto, Stallman coins the concept "free software". As he later has taken great care in pointing out, the word "free" does not necessarily mean "gratis". Stallman's use of the phrase is instead intended to convey the following four aspects of freedom:

1. That it is free of any restrictions that limits its use and application;
2. that it is freely distributable;
3. that it is freely portable between different operating platforms; and,
4. that the source code is available, so users are free to modify and tailor the software

To Stallman, and to most hackers, "proprietary software" is an oxymoron. Software that for technical or legal reasons cannot be modified or adapted, is a dead end.

Hacking in the Real World

> Modern societies have engineers, illiterate societies has bricoleurs, or tinkerers. [...] we, as engineers, are trained to optimize, while as bricoleurs, we are trained to satisfice (Dahlbom and Mathiasen 1993, p. 174).

When asked: "In your work, do you view yourself as a 'tinkerer' or an 'engineer'?" one hacker answered: "Any real developer has to be both. This is what you have to learn from transmission outside the scriptures, from working with other people: When you have to be bottom-up and when you have to be top-down." This brief statement summarises what I consider the three most pronounced aspects of hacking: The emphasis on skills acquired through practice ("outside the scriptures"), the importance of the community ("working with other people"), and the equal emphasis put on engineering ("top-down") and bricolage ("bottom-up").

A graphic account of the hacker as a programmer is to be found in a recent essay named The Cathedral and the Bazaar by Eric S. Raymond (1998). Part diary and part essay, it tracks the development of a particular software system (fetchmail) implemented by Raymond himself and a number of collaborators co-operating across the Internet.

Reading the fetchmail development saga, I was first struck by the similarities between the work practices described by Raymond, and the software development models posed as alternatives to the waterfall model by a number of researchers from the mid-eighties (e.g. STEPS (Floyd 1989) and ETHICS (Mumford 1995)). All the basic ideas (rapid prototyping, iterative development, and strong

user participation) advocated in these models is evident in Raymond's practice, viz.:

> I released early and often (almost never less than every ten days, during periods of intense development, once a day) (Raymond 1998).
>
> One interesting measure of fetchmail's success is the sheer size of the project beta list [...] At time of writing it has 249 members and is adding two or three a week (ibid.).
>
> Users are wonderful things to have, and not just because they demonstrate that you are serving a need, that you've done something right. Properly cultivated, they can become co-developers. [...] Given a bit of encouragement, your users will diagnose problems, suggest fixes, and help improve the code far more quickly than you could unaided (ibid.).

But, as evident by the three quotes from Raymond's paper given above, there are also some important differences:

Firstly, what distinguishes the practice described by Raymond from models such as STEPS and ETHICS is the absence of formalism. Raymond is flying by the seat of his pants, not following any prescribed method.

Secondly, levering on modern tools for automatic system updates and the Internet as an infrastructure for user/programmer contact, Raymond speeds up his development cycles to a frenzy, co-opts his users as debuggers/co-developers, and indiscriminately adds everyone who wants to participate to the project beta list. This is different from the carefully metered out development cycles, and the clear division of roles between users and programmers in STEPS and ETHICS.

Thirdly, users' desire for participating in the endeavour is more or less taken for granted in STEPS and ETHICS. Raymond acknowledges that securing participation from all parties may pose a problem, and argues that the project instigator needs to have some of his/her focus on other people's motivation.

Fourthly, STEPS, ETHICS and similar models are presented as universal approaches that can be used regardless of circumstances. Raymond lists three necessary pre-conditions for his work practice to be applicable: 1) The projected system must fill an unfilled personal need for the instigator; 2) the project needs to secure user participation and maintain continued user support; and, 3) the instigator must have good interpersonal and communication skills.

Future directions

With the exception of Raymond's fetchmail essay, and an enthusiastic, but fragmentary, attempt by Browne (1998) to prescribe hacking as a method for decentralised software development, I am not aware of any serious effort to discuss hacking as a programming practice. So far, there is too little data available to draw conclusions about its merits, qualities and applicability.

I find it interesting to notice that, however, some hacker-like practices are being adopted in environments where one would least expect it: Microsoft, for instance, is successfully turning customers/users into debuggers as "beta" versions of new products are distributed in massive quantities (literally tens of thousands of copies) on the Internet. Microsoft employees also participate in some

of online communities that have formed around the company's products. Netscape and Sunsoft have gone even further down this route. By making the source code of Netscape's Navigator Internet browser and Sunsoft's Java language open and freely available, actively recruiting users as co-developers, and letting staff spend company time participating in user/developer communities emerging on the Internet. Netscape and Sunsoft are experimenting with hacking as means to develop key software packages.

My own conjecture, based upon grounded research (interviews with hackers, comparing and contrasting hacker and non-hacker artifacts, and more than twenty years as a community participant) is that hacking transcends the orthodox centralised and phased view of software development and replaces it with a distributed and evolutionary approach. Also, abandoning a representative (democratic) user/developer dichotomy and putting in its place self-selected (meritocratic) user cum developer community may have profound effects on how we view participatory design.

The potency evident through the hacker community's ability to sustain huge co-operative efforts such as GNU and Linux do in itself give grounds for studying hacking as a software development practice. The apparent adoption of hacker-like practices by significant entities in the software industry compounds this argument. I hope that this essay stimulates others to participate in this research. To get this ball rolling, the following questions are posed:

1. Component re-use:

   Widespread and interorganisational re-use of software component is frequently and repeatedly proposed as a means to cut costs and reduce lead times in software development. In a world of proprietary source code there is no trivial way of doing this. Various clever schemes such as "object linking and embedding", "software ICs" and "object broker architecture" have been proposed as means to accomplish greater re-use. How do these complex and sophisticated methods compare in practice to hackers' cannibalistic and transparent approach that emerge from the existence of open source code?

2. Time to market:

   Two classic goals of software development is high reliability and short development lead times. At the same time, we find that today the scope of much software (and with it the complexity of software development projects) escalate, making these goals harder to accomplish. Closed, managed and centralised software development seems to operate under what is known in the industry as Brooks' Law, which postulates: "Adding manpower to a late software project makes it later" (Brooks 1975, p. 25). Do the limitations inherent in Brook's Law also apply to the huge, distributed and massively parallel software development efforts undertaken by hackers?

3. Participatory design:

   The hacker community advocates transcending the user/programmer dichotomy that is inherent in our present views on participatory design, and replaces it with a merito-

cratic user cum programmer community. How does this model operate in practice? Is this a fair way to organise the software development process, or does it open up for abuses (e.g. does it allow programmers to manipulate hapless users by means of "model power" (Bråten 1981))?

4. Design for heterogeneity:

   A recent thesis (Thoresen 1999, p. 50) asserts that system development theory lacks systematic approaches to how heterogeneity can be accommodated. To what extent is the study of hacker work practices, such as open source development, bricolage, and continuous adaptation of software artifacts through own use, able to add to the theory in this particular area?

References

Bråten, Stein, (1981). Quality of Interaction and Participation. On Model Power in Industrial Democracy and Computer Networks. In: G. E. Lasker (ed.): Applied Systems and Cybernetics, vol. I, p. 191-200.

Brooks, Jr., Frederick P., (1975). *The Mythical Man-month. Essays on Software Engineering*. Addison-Wesley.

Browne, Christopher B., (1998). Linux and Decentralized Development. *First Monday*, vol. 3:3 1998.

Cerf, Vint, (1997). Father of the Internet, interview conducted by technical editor Leo Laporte; Broadcast by MSNBC: *The Site*, transcript, June 3, 1997.

Floyd, Christiane, (1989). STEPS to Software Development with Users. *Proceedings of ESEC 1989*, University of Warwick, Coventry, England, 11-15 September 1989.

Hannemyr, Gisle, (1999). Technology and Pleasure: Considering Hacking Constructive. *First Monday*, vol. 4:2, February 1999.

IEEE Software, (1999). *Focus: Linux*, vol. 16:1, January/February 1999.

Mumford, Enid, (1995). *Effective Systems Design and Requirements Analysis. The ETHICS Approach.* Macmillan Press, 1995.

Raymond, Eric S, (1998). The Cathedral and the Bazaar; *First Monday*, vol. 3:3, March, 1998.

Stallman, Richard M., (1985). The GNU Manifesto. *The GNU Emacs Manual.*

Steele jr., Guy L, Donald R. Woods, Raphael Finkel, Mark R. Crispin, Richard M. Stallman, Geoffrey S. Goodfellow, (1983). *The Hacker's Dictionary. A Guide to the World of Computer Wizards*. Harper & Row.

Thoresen, Kari, (1999). *Computer Use.* Dr Philos Thesis, March 1999, University of Oslo, Department of Informatics.