



UvA-DARE (Digital Academic Repository)

Having it both ways: Larry Wall, Perl and the technology and culture of the early web

Stevenson, M.

DOI

[10.1080/24701475.2018.1495810](https://doi.org/10.1080/24701475.2018.1495810)

Publication date

2018

Document Version

Final published version

Published in

Internet Histories: Digital Technology, Culture and Society

License

CC BY-NC-ND

[Link to publication](#)

Citation for published version (APA):

Stevenson, M. (2018). Having it both ways: Larry Wall, Perl and the technology and culture of the early web. *Internet Histories: Digital Technology, Culture and Society*, 2(3-4), 264-280. <https://doi.org/10.1080/24701475.2018.1495810>

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (<https://dare.uva.nl>)

Having it both ways: Larry Wall, Perl and the technology and culture of the early web

Michael Stevenson

University of Amsterdam

ARTICLE HISTORY

Received 20 March 2018


Accepted 26 June 2018

Introduction: Perl and the exceptionalism of the 90s web

What image defines the 1990s web? Perhaps it is an “under construction” gif, a “starry night” background or some other fragment of what net artist and scholar Olia Lialina dubbed “a vernacular web” (2005). If not a vernacular, perhaps a sign of an increasingly commercial and professional web – the first banner ad, announcing that this particular information superhighway would be dotted with billboards and shopping malls, or a jutting line graph showing the precipitous rise of the Nasdaq composite index.

Of course, the answer is both, or all of the above. The 90s web was defined by its contradictions: amateur and professional, playful and serious, free and incorporated. Early descriptions of the World Wide Web’s significance oscillated between, on the one hand, an accessible and open alternative to walled gardens like America Online, and on the other hand an electronic frontier ripe for commercialization (Markoff, 1993; Wolf, 1994). Long before social media or web 2.0 became buzzwords, startups and new media gurus claimed the web was both a place of community and a place of commerce (Silver, 2008). Importantly this was not a matter of two webs existing side-by-side: the 90s web was all of these things at once. Perhaps it was this capacity for having it both ways, more than any single technical feature, that made the web feel new.

The height of have-it-both-ways-ism was the wave of hype and financial speculation surrounding all things “open source” from 1998 to 2000. The term was adopted and promoted by a group of hackers and entrepreneurs who wanted to legitimize free software projects and their practices of sharing code, both as something technically on par or even superior to commercial software, and as a commercially viable model of software production (Kelty, 2008; Raymond, 1999). This required ditching the term “free software,” which they felt carried too much baggage, and the new concept clearly embraced both volunteerism and professionalism, commons and ownership, and pleasure and profit. It spread like wildfire. Before long, companies branding

CONTACT Michael Stevenson  m.p.stevenson@uva.nl

© 2018 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group

This is an Open Access article distributed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives License (<http://creativecommons.org/licenses/by-nc-nd/4.0/>), which permits non-commercial re-use, distribution, and reproduction in any medium, provided the original work is properly cited, and is not altered, transformed, or built upon in any way.

themselves as open-source were on Wall Street, setting records with their Initial Public Offerings. Elsewhere, others began asking how this concept could be applied to other domains. Open-source journalism? Of course. A volunteer encyclopedia you could rely on? Sure. Open-source cola? Why not.

How was this situation possible? Or more generally, how did this form of positioning come to be so taken-for-granted, such that startup CEOs could pen slogans like “Don’t be evil” without inspiring derision from their peers or parody sitcoms? For one of the most visible advocates of open-source software, hacker and author Eric Raymond, the case of “open source” was a matter of waking up. After “[t]wenty years of repeatedly watching brilliant ideas, promising starts, and superior technologies crushed by slick marketing,” free software hackers had realized their ad-hoc efforts that ran on little more than technical ingenuity and passion could compete with corporate behemoths like Microsoft and IBM – not to mention make a few bucks – if only they learned to do better public relations (Raymond, 1999, p. 210). According to critics, it was the opposite: it was delusional to think open source software could somehow remain an unspoiled gift economy while also being sponsored by capital, and instead this was ultimately a familiar story of cooptation (Barbrook, 1998; Morozov, 2013).

What both Raymond’s heroic narrative and critical accounts neglect are various material, practical and discursive interventions that prefigured the rise of a business-friendly face of free software in the late 1990s. To shed light on this case in particular, and more generally on having it both ways when it came to imagining the value or character of the web in the 1990s, this article explores the history of the Perl programming language’s rise from a free software utility created by an idiosyncratic system administrator in his free time, into an indispensable technology for countless web companies, professional web developers and homepage-building amateurs.

Why Perl? Although Perl was first released in 1987 and thus preceded the World Wide Web by a few years, the material and conceptual links between its history and that of the 90s web are myriad.¹ Perl, most notably, was an essential ingredient in the invention and use of the Common Gateway Interface (CGI), a web server feature that for many years was the de facto means for making web pages interactive and dynamic (Stevenson, 2016). Meanwhile, the web is what made Perl a surprise competitor to languages backed by large corporations. As the web grew, so did Perl, from a few thousand users in 1990 to over a million a decade later. While many of these users were casual programmers adding a guestbook or contact form to their website, other users like early web giants Yahoo! and Amazon demonstrated Perl’s legitimacy as professional-grade technology. Perl’s history, like the early web’s, is one of contradictions and synthesis. In addition to Perl acting as a platform for both amateur and professional web development, its creator Larry Wall actively sought to invent practices and institutions of collaboration between the free software world and commercial interests, something he did largely in concert with his eventual employer, book publisher Tim O’Reilly.

The historical narrative is structured in two parts. First, I argue that Perl must be understood in the context of Wall’s work on previous free software projects. Drawing on Boltanski and Thévenot’s (2006) concept of justificatory regimes, I argue that this body of work was notable for how it committed to a “civic world” of valuing software in terms of a uniform collective interest. However, this framework also helps to

uncover differences between Wall's free software work and that of Richard Stallman, founder of the Free Software Foundation. Second, I show how Perl's success in the 1990s was tied to its culture of synthesis, or having it both ways.² Both in terms of the language's design, as well key actions and decisions made by Wall and others regarding the organization and culture of the Perl community, Perl's success was tied to how it emphasized flexibility and evolution while encouraging integrity and portability, how it neutralized potential conflict, and how it balanced between Wall's creative control and the collective efforts of an increasingly large and devoted community of volunteer developers. Most significantly, through their collaboration around Perl, Wall, O'Reilly and others instituted forms of cooperation between free software communities and the mainstream computing industry.

In short, my claim is that Perl's significance for the 90s web goes beyond the well-documented use of the language in web development. The language's history illustrates the kinds of material, social, economic and discursive arrangements that enabled an odd form of 'autonomous' production within the emerging field of new media. For this history I have drawn on a range of archival sources - various Usenet newsgroups dating from the early 1980s, documentation for past versions of the Perl programming language and related software, Perl reference books, published journalistic interviews with Perl creator Larry Wall and several other key figures in Perl's history - supplemented by semistructured interviews with Larry Wall, Jon Orwant, and several practitioners who either identified as members of the Perl community or identified Perl as a key technology in their work as web developers in the 1990s.

Free software and the civic order of worth: *rn*, *patch* and *metaconfig*

Beginning in 1984, Larry Wall developed several influential free software programs and Unix utilities, culminating in his work on Perl. Before releasing Perl in 1987, Wall wrote *rn*, a Usenet newsreader, *patch*, a program for managing versions of software and easing the distribution of edits to source code, as well as *metaconfigure*, a utility that software developers could use to automate the process of writing configure scripts for their programs. These programs (in particular *patch*) are acclaimed for how they facilitated and anticipated the decentralized mode of software production that would eventually be dubbed the open-source model of development (Raymond, 2003, p. 38). As I argue here, they may also be understood in terms of how they justified this form of production - that is, how they served as a set of arguments about how software should work, or what should be valued in software. Here, I discuss Wall's body of work both as software objects that enabled and constrained certain actions and behaviors on computers, and as cultural expressions accompanied by implicit and explicit attempts to educate users and other developers in terms of best practices. Wall not only provided tools for navigating Usenet newsgroups or for making software production and maintenance less arduous, but in doing so also promoted particular values, ideas and practices present within the cultures of Usenet and Unix. In other words, this software formed a material intervention expressing Wall's position on the structure and content of a "recursive public" of geeks (Kelty, 2008). Not least, Wall's work

suggested a commitment to a civic order of worth, or regime of justification that privileges the interests of collectives over individuals (Boltanski & Thévenot, 2006).

The basic arc of Wall's free software development in this period goes like this: he developed the feature-rich `rn` newsreader and released this large and complex program (relative to other free software) to Usenet in 1984, where it quickly gained a large user base. Several `rn` users sent Wall bugs, feature requests, and suggested improvements to the code, and Wall responded by posting source code patches to the program. The work of maintaining `rn` was cumbersome, however, and led Wall to write `patch` and `metaconfig`, two tools geared toward ensuring standardization and portability for Unix software. While working as a consultant on a project for the US National Security Agency, Wall was tasked with repurposing `netnews` (Usenet's server software) into a distributed, synchronized report system with cross-references between articles. For this Wall could draw on his experience writing `rn`, however he found it easier to write his own scripting language than to use available tools. He would later call the language Perl, and release it to Usenet in December 1987. Perl then took on a life of its own.

At the time `rn` was released, the Usenet network consisted of an estimated 900 sites, representing universities, research centers, companies and other organizations, and roughly 225 articles per day posted to various subject-specific newsgroups (Hauben & Hauben, 1997, p. 44). Usenet was very much a volunteer operation, a "poor man's Arpanet," with system administrators maintaining the network in their spare time (*ibid.*). As it connected primarily computer scientists and engineers, it was common for users to share small Unix programs and utilities. These were always distributed as source code, since Unix was not a single operating system but many: the "Unix wars" of the early 1980s pitted Bell Labs' commercial System V operating system against the Berkeley Standard Distribution (BSD), while a range of different system architectures and a few startups building their own proprietary versions of Unix combined to produce a highly fragmented landscape. Sharing software as source code meant an administrator could edit a program so that it worked on her particular system.

Both the existing practice of sharing software and the fragmented Unix landscape helped shape Wall's contributions. As Wall noted in a speech about Perl and Linux many years later, sharing code in this way allowed for a departure from various conventions and expectations – not least that code had to be perfect (Wall, 1999). At the same time, both the risks and rewards of sharing software on Usenet were tied closely to an individual's reputation. On the one hand, shared code was an opportunity for personal, often creative expression. One could display the kind of clever workarounds and repurposing of technology highly valued in a profession marked by complexity and opacity (Coleman, 2012). On the other, one had little control over how the code would be used or in what context. Wall's own reputation had been "besmirched" a few years earlier, when a file management system he helped write while in graduate school was poorly reimplemented with his name still attached (Wall interview).

Beginning with the first release of `rn`, Wall was clearly motivated by personal pleasure as well as the idea of contributing to and improving Usenet's volunteer infrastructure and practices of sharing code. Wall annotated his source code with jokes, and

took pride in how his program was “human-engineered,” with keyboard mappings and interface design choices motivated by user-friendliness rather than software conventions (Wall, 1984a). Most of all, though, he hoped users would appreciate the configure script that used tests and a question and answer format to help administrators edit the program to work on their particular flavor of Unix. The script was exceptionally useful given the diverse implementations and architectures, while it also gleefully narrated its actions with inside jokes like “Congratulations! You’re not running Eunice” (a reference to a particularly noncompatible Unix-like system). Regardless of whether users like `rn`, Wall wrote, he hoped that `configure` might set a new standard for “friendly distribution” (Wall, 1984a).

A day after releasing `rn`, Wall posted the first two patches to the program, both of them for bugs found by users (Wall, 1984b). As he continued to fix bugs and add features via patches, Wall experienced both the pleasures and pains of maintaining free software among a diverse user base. On the one hand, his active maintenance meant users were more inclined to give feedback, ideas and suggestions he found useful. When he released `rn` 4.3 in June 1985, he thanked users for “encouragement and ideas” and wrote that it was a “truly a net-wide project” (Wall, 1985). On the other hand, the problem of fragmentation was exacerbated by `rn`’s numerous patches, as newer patches were often dependent on previous ones that users may or may not have added. Fragmentation and the pace of `rn`’s development thus formed a material resistance to both the intrinsic and altruistic motivations for maintaining such a project.

Wall soon began writing `patch`, a program that automated much of this maintenance work through several features for context-aware updates. Users would run `patch` to update `rn` (and before long many other programs that used `patch`), and `patch` would perform several actions to check the expected source code against the actual code. When these did not match, the program would repurpose a Unix command called “context diff” to scan up and down the source code to see if it matched on other lines. If the code was similar enough, `patch` would be able to make the necessary changes without overwriting any customized code. By contrast, every patch file included an argument where it would refuse to run if the previous patch was not installed – to use the program, one had to install patches in sequence. The program thus circumvented problems created by the lack of standardization and uniformity, while also disciplining users to ensure their code stayed in sync.

Where `patch` supplied users with a useful program for keeping `rn` up to date, it added to Wall’s list of maintenance tasks. He wrote `metaconfig` to automate the work of writing configure scripts for the various versions of the programs he shared (including `warp`, a space-war game Wall had written previously), first for his own use with `rn` and `patch`, and eventually releasing the program in January 1988. “This package knows just about everything I know about portability to Unix systems,” Wall wrote, before describing how the program wrote configure scripts that would test for each necessary software package and its dependencies separately (Wall, 1988b). This bottom-up approach to determining system context was – along with `patch`’s capacity for dealing with custom code – a means to manufacture portability without standardization. It was a means of keeping a diverse collective in sync so that a common good (the code base for `rn`, or later Perl) could be more easily maintained and grown.

What arguments were made by `rn`, `patch` and `metaconfig`? In the framework of “orders of worth” from Boltanski and Thévenot (2006), the programs can be seen to promote a technological civility. The worth of these projects – in Wall’s words, in the praise he received – was certainly in part about their capacities for increasing the values at the core of the industrial order of worth. But value was found just as much, if not more, in how these added to the Usenet commons, growing out of collective action – for example, in Wall’s (1985) pride that `rn` had become “truly a net-wide project” – as well as codifying, automating and to some degree even enforcing such collective action through `patch` and `metaconfig`. In 1987, Wall began work on Perl. Although this work – unlike `rn` and the other programs – was done officially on the employers’ dime, from the beginning Wall knew the program could be useful for others and planned to give it away.

From the GPL to dual licensing

Wall’s work gave new impetus to an ethic of sharing code that had hit roadblocks in a Unix landscape fragmented by competition between proprietary distributions. Not coincidentally, this was the same context in which Richard Stallman wrote the GNU manifesto (1985), outlining his plans to create a free Unix-compatible operating system and calling on programmers to contribute to it. Stallman had seen colleagues leave MIT to work for proprietary computing companies, and took their newfound unwillingness to share code as an affront. Although the manifesto presents many of its arguments for free software in appeals to the field’s dominant logic of industrial efficiency (e.g. by noting how the free operating system will prevent “wasteful duplication of system programming effort”), Stallman clearly grounds his arguments in a moral economy of civic value, arguing that “software sellers want to divide the users” and that “[i]f anything deserves reward, it is social contribution” (Stallman, 1985).

Wall signaled his sympathy for Stallman’s project by releasing Perl 3.0 under the GNU Public License (GPL) in 1989. Prior to this, Perl came with a short notice saying “You may copy the perl kit in whole or in part as long as you don’t try to make money off it, or pretend that you wrote it” (Wall, 1988a). Although the GPL was arguably more permissive (in that it does not preclude anyone from selling copies of the licensed software), Wall’s decision was striking, as Stallman’s project, and in particular the socialist leanings implied by it, were viewed skeptically by many of those who discussed Stallman’s manifesto on Usenet. Given that the first version of the GPL was published just a few months before Perl 3.0 was released, Perl was likely one of the first non-GNU projects to bear the license.

Despite this move, there were also signs that Wall’s commitment to GNU’s politics, and more generally to a civic order of worth, was ultimately limited. As he explained in a longer discussion of motivations for producing “free software,” Wall (1988c) worked primarily to please himself, and he only released these products so that his “life’s work” wouldn’t “die when this computer is scrapped”. Wall suspected his work had little market value, and even if it did he couldn’t see himself act like a “mercenary.” This led him to the following conclusions:

I guess I'd say that the reason some software comes free is that the mechanism for selling it is missing, either from the work environment, or from the heart of the programmer [...]

What programmers like me need is a benefactor, like the old composers and artists used to have. Anybody want to support me while I make beautiful things? (ibid)

Here one sees how Wall's position differed not only from typical market or industrial logics, but also seeds of difference from the arguments made by Stallman. Where Stallman's manifesto consistently returns to the theme of solidarity among programmers, and this uniformity of critique is inscribed in the GPL, here value was found (or should be found) not only in a collective interest, but also, and perhaps even more so, in the creative work itself and in the autonomy of authorship. Driving this point home he appended the following in the release notes of Perl 3.0:

Just a personal note: I want you to know that I create nice things like this because it pleases the Author of my story. If this bothers you, then your notion of Authorship needs some revision. But you can use perl anyway.: -)

- The author (Wall, 1989)

In addition to signaling his spirituality, here Wall indicates a limit on the extent to which Perl would conform to outside interests. The rejection of compromise, the emphasis on authorship, and even the suggestion of a greater spiritual authority guiding his work all point to what Boltanski and Thévenot describe as the "inspirational world," a moral economy in which value is accorded to those who devote themselves to a purpose external to both themselves and outside interests (e.g. in devotion to the work of art), those who are perceived to be truly original in their thinking, and thus those who are seen to inspire passion and make possible a break from entrenched routines or practice.

In 1991, Wall carried out what he later called a "cultural hack" (quoted in Tamiya, 2001), inventing the dual license by releasing Perl under both the GPL and The Artistic License. The latter was authored by Wall, and distinguished itself from the GPL in two important ways. First, it laid out a number of provisions to ensure the relative integrity of Perl, allowing modifications and reuse but maintaining "some semblance of artistic control" over "the standard distribution" (Wall, 1991, emphasis in original). Second, it made explicit that Perl could be used to create commercial products that could be released as proprietary software.

The Artistic License thus emphasized authorship and control at the same time that it was more permissive than the GPL. If the GPL enforces a form of solidarity in which any modifications or extensions are returned to the commons, the Artistic License seeks to maintain Perl's integrity and Wall's reputation while simultaneously allowing for commercial use. The fact that users could choose which license they wanted to use Perl under was itself innovative. Rather than remove the GPL and risk alienating the free software community, Wall's solution meant having it both ways: Perl belonged to the collective and to the author; it was an expression of GNU solidarity while also a tool for commercial software production.

Perl's culture of synthesis

During the 1990s, Perl grew from a Unix hacker's tool into the "glue" holding the web together, and from a user base of a few thousand to over a million. Likewise, it transitioned from a free programming language shared among system administrators into a competitor with languages such as Sun-backed Java and a source of profit for internet startups, software vendors and programmers who used their knowledge of Perl to land web development jobs. What tied these changes together was a particular pattern of decision-making and positioning – rooted in the language's design, the organizational structure of the Perl community as well as in Wall's evolving theories of Perl and its success - that facilitated and emphasized synthesis of opposing or contradictory practices and belief systems.

Technological synthesis

Perl's affordances positioned the language for both professional and amateur web production, allowing it to compete both for prestige in the field of restricted production (among experts and hackers) and for market share in the field of mass production (c.f. Bourdieu, 1993). Perl's intentional flexibility – the way its interpreter worked hard to understand code and 'forgave' such practices as not declaring variables, or using numeric operators on strings, as well as the interpreter's capacity for automatically managing memory – had two notable effects. On the one hand, it allowed for an uncommon level of expressiveness whereby expert programmers could employ terse but powerful statements, inviting the kind of creative, playful interaction that could be both productive (in terms of maximizing efficiency) and was celebrated in such extracurricular practices as Perl poetry and Perl Golf. On the other, it meant that scripts were sturdy, if not always efficient or pretty: Perl allowed amateur programmers to write (or copy) CGI scripts that worked, even if their command of the language was minimal (Garfinkel, 1997).

Perl's affordances for rapid prototyping and text-processing were equally important for its use in CGI programming, especially given the dynamics of the emerging industry in the mid- to late-1990s. Web companies, in contrast to existing software development companies, had strong incentives to adopt Perl beyond its suitability for CGI. Optimizing for performance would require using a compiled language like C, however Perl could reduce production time and add functionality to a site quickly, something that was essential given the fast pace of change for web startups in the late 1990s (Stark, 2011). While Wall noted that mainstream computing companies rejected the language because they perceived it as "schlocky" (quoted in Kim, 1998), startups like Amazon and Yahoo relied heavily on Perl. Yahoo, for example, used Perl for numerous maintenance tasks and product features – these included a program to automatically find and remove broken links from its directory, as well as a name recognition program that matched news items to company names for Yahoo's stock market product (O'Reilly, 1998).

Perl's growth following the release of Perl 4 had two important consequences for the work of developing and maintaining the language. On the one hand there was an increasing number of programmers willing and able to contribute to Perl's core

development, much of which involved porting the language to different architectures. On the other, as Perl became more widely used, it began to fragment. Wall had seen how tcl, another scripting language, had spawned several “custom” versions that added features for interfacing with particular databases or other specific uses (Wall interview). The same process began to affect Perl, for example with Kevin Stock’s oraperl, a customized version of the perl interpreter designed to interface with Oracle databases. Running an oraperl script through the normal perl interpreter would simply cause the program to die. And so Wall, after his experiences with rn and patch, faced another situation where his program was evolving outside of his control and generating compatibility issues and potential confusion. The solution was both technological and social. First, for Perl 5, Wall wrote an extension mechanism that provided a standard format for incorporating such custom code. Second, this was accompanied by efforts to coordinate and centralize the development of Perl modules (Bunce, 1994), eventually leading to the Comprehensive Perl Archive Network (CPAN). Once again, Wall sought to achieve synthesis between opposing desires – keeping Perl core development integrated on the one hand, and encouraging growth and evolution on the other.

Cultural synthesis

This emphasis on integrating Perl also led to collaboration between Perl 5’s volunteer core developers and a commercial vendor supported by Microsoft. In 1995, Microsoft agreed to fund a port of Perl 5 to Windows, however the code soon forked into two separate projects: the commercial vendor, ActiveWare (later ActiveState), optimized Perl to work with Microsoft’s NT server product, while volunteers coordinated on the Per5-porters mailing list to improve compatibility between the Unix and Windows versions of Perl. In 1997, with encouragement and support from O’Reilly & Associates as well as Larry Wall, the two project teams began to coordinate regularly on “Oneperl,” producing a shared source tree (and thus compatibility) by the summer of 1998 with the release of Perl 5.005. The collaboration was unique at the time, and ActiveState represented a model of how a company could justify its actions within different orders of worth, switching between civic and market worlds depending on whether its audience was the free software developers it relied on or the company’s corporate clients and end-users.

The collaboration between the Perl core developers and ActiveWare was facilitated by another business relationship: in 1996, O’Reilly & Associates hired Larry Wall as senior software developer to work on Perl-related products. Wall called his employment “a form of patronage” (quoted in Tamiya, 2001), fulfilling the desire he expressed years earlier on Usenet (Wall, 1988c). In another interview, Wall explained that his position allowed him to exercise control over Perl’s commercialization, in particular ensuring integration. In reference to the different versions of Win32 Perl, he noted:

[W]e actually see this sort of fragmentation in the Linux market. And I’ve taken a slightly different tack than Linus Torvald [the creator of Linux] has. He’s gone to work for a chip manufacturer because he wants to stay totally out of the fray. My feeling on that matter is: I know that there has to be some growth in the commercial area as well as in the

freeware area. So I'd like to be close enough to that action that I can sort of bless some aspects and not-bless other aspects of it.

That's the main motivation for me coming to O'Reilly a year ago. So that I could be part of this and sort of boot up the commercial aspects of this in a controlled fashion (quoted in McMillan, 1997).

Here, Wall's actions and how he justifies them suggests synthesis between three ways of valuing Perl and the various connections made between free software and commercial entities: civic values, to the extent that all sides would put extra work into ensuring compatibility; market values, to the extent that these efforts could be framed as ensuring a stronger competitive position for Perl and Perl-related products; and, in particular for Wall, the value of maintaining control over Perl's identity.

Organizational synthesis

Beginning in 1996, several initiatives were launched to promote and legitimize Perl, although these diverged notably in character. On the one hand, there were centralized operations that sought to promote Perl to the mainstream computing world and potential corporate users, while on the other were grassroots efforts that promoted informal community and volunteer participation.

The first group included The Perl Institute, a non-profit founded by Wall, Schwartz and Christiansen in 1996 that listed among its original goals to "keep Perl free by [...] interfacing between the freeware and commercial communities" (Schwartz, 1996). The Perl Institute had an ambitious vision of Perl conferences and corporate training programs to fund Perl development (Maxwell, 1996), however little would come of this as the institute was dissolved in 1999 following financial troubles. The first annual Perl Conference, organized by O'Reilly & Associates in August 1997, was more successful. The conference had just over a thousand attendees, and was organized along three tracks: core development, Perl for Windows, and Perl for the web. In addition to promoting Perl as what O'Reilly called "the Intel of the information revolution" (McMillan, 1997), the conference aimed to showcase how free software as a commercial opportunity. The latter point was driven home in particular in the two keynotes: Wall opened the conference with a talk about Perl culture that also served as a call for cooperation between the free software and business communities, while Eric Raymond presented "The Cathedral and the Bazaar" for the first time in the US, and thus the argument that the informal, collaborative style of free software production seen in Perl, Apache and Linux yields better software than top-down structures.

The second group consisted of informal, grassroots efforts such as Perl Mongers, which began in 1997 as a New York City users group. The format spread to several hundred other cities both in the US and abroad within the next 2 years (Michalski, 1999), and many of these groups remain active in the late 2010s. Perl Mongers meetings range from informal after-work events to small events with invited speakers. Another grassroots effort was the annual Yet Another Perl Conference (YAPC), organized by Carnegie Mellon graduate student Kevin Lenzo in 1998. From the start, YAPC was clearly distinguished from O'Reilly's Perl Conference. From the insider's name (a reference to Unix software called Yet Another Compiler-Compiler) and cheap

registration (\$60 USD and free for Carnegie Mellon students) to the informal nature of its talks and its location on the East Coast, the conference seemed in some sense an exercise in subcultural distinction. Where the Perl Conference served largely to legitimize Perl to an outside world of mainstream computing companies and potential commercial interests, YAPC aimed to facilitate the kind of deep affection felt by those who already identified with the language, as well as providing a platform for peer recognition.

Despite some obvious differences between the two kinds of initiative, both sides saw the relationship as complementary and cooperative rather than competing. On the one hand, the Perl Mongers team was careful to coordinate with The Perl Institute and define its mission in terms separate from those of the Institute (Michalski, 1999), and when The Perl Institute dissolved, its board voted to gift the few resources it had to the Perl Mongers organization. On the other hand, the Perl Conference provided the opportunity for the first face-to-face gathering of around 50 core Perl developers at Wall's home, mirroring the kind of informal and collegial setting among insiders that YAPC would seek to create, while O'Reilly and ActiveState were among the conference sponsors for YAPC. Organizationally, then, Perl resisted any single logic, a situation made possible by an informal hierarchy and Wall's benevolent dictatorship. For Wall, this informal structure was motivated by more than pragmatism. Rather, it was essential to sustaining the creative energy needed for the project (both his own and that of the community). This is seen for example in how he regularly dismissed the idea of submitting Perl to a standards organization, as such "headless standards committees tend to reduce everything to mush" (quoted in Kim, 1998).

Perl and the inspirational order of worth

The many ways in which Perl synthesized opposing practices or values were not lost on Wall, who actively culled together an image of Perl as an unorthodox upstart within the history of computing. In conference talks and interviews, Wall often return to the same theme: Perl's idiosyncrasy lies in its flexibility and its embrace of contradictions, and this in turn fuels Perl's success. Beyond theorizing Perl, Wall regularly used his acclaimed keynotes to provide a kind of "moral education" aimed at both the free software community and the increasing industry presence at Perl and Open Source conferences. For instance, in Wall's keynote at the O'Reilly conference in 1997, he argued that if there is "an important idea in Perl Culture, it's this: that too much control is just as deadly as too little control." Wall went on to explain that the language's success was due to an intellectual flexibility – a way of seeing convergence in ideas that appear at first to be incompatible – and a corresponding practice of allowing this set of ideas to evolve (Wall, 1997). The latter he compared to the work of the "best artists [...] who see a work of art beginning to take shape, and are able to exercise just the right amount of control to let the art have its own way" (ibid).

Wall then drew this lesson out to an explicit appeal to free software supporters and industry executives to apply the same kind of intellectual flexibility: "[A]sk yourself whether your belief system is closed or open [...] Can you imagine Perl evolving to fill new ecological niches, while still remaining the same comfortable, old Perl?" Rather

than hold onto old convictions and ideas, both sides could “help us do something new. A new model of cooperation is emerging” (ibid).

If Perl succeeded because of the contradictory beliefs designed into it, and because it balanced control with a lack of control, how could this form of positioning be translated to the larger question it now faced about commercialization? It’s important to note that the “new model of cooperation” that Wall had in mind as a solution was not what Raymond called the bazaar model of free software development. Rather, it was the model of cooperation on display at the conference: Perl’s association with O’Reilly, and ActiveWare’s intermediary role between the Perl community and Microsoft. O’Reilly could promote and legitimize Perl to a larger audience in a way that Wall and free software advocates could not, while it also profited from selling Perl products. ActiveWare demonstrated commitment to the Perl commons while it also sought to open up a commercial market for proprietary additions to free software. It was a matter of finding a middle ground between two different ways of valuing software and justifying the vast efforts involved in its production, namely the civic and market orders of worth.

How was this cease-fire possible? Between the lines in Wall’s keynote, the answer is clear: through a mutual commitment to the integrity and autonomy of Perl itself (and by extension its author). In this light, Wall’s metaphor of Perl as art is crucial, as it serves to insert a third, inspirational order of worth, or means of calculating Perl’s value. As art, Perl is valued for its originality and understood as an agent in its own right, something that evolves largely according to an interior logic. In this metaphor, the software is not tainted by commercial interest or market success (or, on the other side, the conservatism of a standards committee) so long as Perl’s autonomy and self-determination are not compromised. In this natural development, Perl can evolve to “fill new ecological niches,” commercial or otherwise, while remaining “the same old comfortable Perl” (ibid).

Wall’s appeal to the inspirational order of worth served to smooth over contradictions in Perl culture at the same time that it helped make the case that Perl and other open-source software represented a break from computing tradition. Although the theme runs throughout Wall’s books, documentation and interviews in the 1990s, it is most explicit in Wall’s keynote from the LinuxWorld Expo in 1999, titled “Perl, the first postmodern computer language.” Drawing again on analogies to cultural production – music, art, architecture, journalism – Wall sets up an opposition between modern and postmodern computing. Perl and Linux were the first examples of the latter, as they left behind the modernist cults of spareness, originality, seriousness and objectivity. Perl and Linux were unabashedly pastiche while leaving behind the “Serious Business” of closed-source software. Finally, this argument is folded back into Wall’s definition of an ideal open-source subject. At the peak of financial speculation around Linux, and when various Linux tribes competed along the fault line between market- and civic-oriented actors, and when the same rift still affected Perl, Wall argued that there were two kinds of joiners. Where modern joiners naturally gravitated to the prototypical center of a tribe, the “open source movement is energized by the other sort of joiner,” one who looked a lot like Perl:

This sort of person joins many tribes. These are the people who inhabit the intersections of the Venn diagrams. They believe in ANDs rather than ORs. [...] I call these people “glue people,” because they not only join themselves to a tribe, they join tribes together (Wall, 1999).

From his early work on `rn` and `patch` to the language design principles expressed in Perl, and from his earliest Usenet postings and the dual license to a variety of books, interviews and conference talks, Wall’s contributions to free and open source software go beyond technical facilitation. Just as importantly, this work provided a moral education that was unique in its emphasis on “joining” not just competing systems but competing belief systems. What made the market- and civic-oriented actors compatible was a set of technical, legal, social and discursive practices that were justified in terms of their devotion to the larger project of Perl, and thus drew heavily on the conventions Boltanski and Thévenot associate with the inspirational mode of justification.

Postscript

Five ceramic coffee mugs flew in sequence through the air, one of them breaking into pieces as it made impact with the wall of the hotel’s meeting room. The message these mugs sent, directed at a small group of Perl developers gathered in a side-session during the 2000 Perl conference, was that something had to change. The discussion had turned to the Perl community, how it was stagnating and fragmenting. The cause was unclear. Was it because technical limitations inherent in Perl’s design were revealing themselves as its usage expanded? Had Perl’s informal hierarchy and loose mix of centralized and grassroots organization reached its limits? Was it the fact that Perl’s major niche, the web industry, increasingly adopted Python, PHP and Javascript? Or, in retrospect, was it one of so many signs that the dot.com bubble was starting to burst? Maybe it was all of these things.

The solution was also not clear, and before mugs started flying over their heads the participants were half-heartedly proposing ideas for a Perl constitution, in order to formalize and institutionalize changes to how Perl was run. Mug-thrower Jon Orwant devised his piece of theater, however, to argue that they were looking in the wrong place. If people were not excited about a constitution, they should ditch that idea and find a better one. He used Napster, the controversial mp3 sharing program with an innovative peer-2-peer infrastructure, as an example of a big idea that people were excited about. The light bulb went on in Wall’s head. Yes, the problem was about Perl 5’s limitations and changing technological needs, and it was also about the Perl community. However, the solution to these problems was not to make incremental improvements or to institutionalize; the solution was to inspire the community, and to break from the past. Wall decided they would rewrite Perl from scratch, even making the radical move of breaking backward compatibility, something that had been a hallmark of Perl development. But wouldn’t Perl 6 just speed up Perl’s demise as Perl 5 development stalled even further? No, Wall would say, that assumed they had to choose between one or the other.

Perl 6, though, hit several roadblocks. In the face of the economic downturn O'Reilly laid off a large portion of his staff in 2001, including Wall. Wall also became severely ill, losing at least a year of Perl development. Perl was hurt by brain drain, as key members of the community took non-Perl jobs. Perl 6's development trudged forward, but in its nearly 20-year history implementations have yet to live up to the lofty goals outlined by Wall and the community at the outset. As the years passed, news of Perl's death began to appear with regularity. Much like we are prone to mourning the spirit of the early web, accounts of Perl's death carry a tinge of nostalgia for a time when programming felt fun and experimental while also serious and important (fittingly, the earliest predictions of Perl's fate can be found in an article entitled "The joy of Perl" [Leonard, 1998]). Perhaps instead of mourning the passing of Perl and the early web, though, we should mine those pasts for lessons that will help us make such things new again.

For historians of the technology and culture of the early web, Perl's story carries at least two important lessons. First, it serves as a reminder that understanding technological change requires going beyond structural explanations. Perl's success was certainly a case of being in the right place at the right time, in that it was so clearly suited to the specific technological and economic context of the 90s web. However, Perl was not invented in order to meet these needs, and its innovative technical character was instead grounded in Wall's specific interventions within the world of (free) software production. Perl emerged from a particular computing culture – the set of existing technologies, practices and values associated with Usenet and Unix – as well as Wall's critical reflections on this culture. Second, Perl's history helps to contextualize the either/or logic through which critics view open source projects, and perhaps by extension the communitarian spirit of the early web, where the presence of capital is seen to contaminate and deflate any political potential (e.g. Morozov, 2013). As O'Neil (2011, p. 5) argues in relation to Wikipedia, the politics of peer production lies in communities' "conscious rejection of alienation," and thus in how these ad-hoc organizations aim not just to create democratic structures able to resist outside pressures, but feelings of unity, belonging and purpose. In this light, Wall's efforts both as a language designer and as the leader of the Perl community can be succinctly described as what O'Neil calls "critiques of separation":

The critical operations of people in commons-based peer production projects are critiques of separation: in a world which denigrates solidarity and promotes division into ever-smaller market segments, participants in these projects seek a feeling of unity between their identities as consumers and producers, between their status as experts and amateurs, between their roles as leaders and followers, between their activities of work and play, and between themselves and their fellow participants in the project – a project which they see, more often than not, as a cause to defend (ibid: p. 5–6).

What Perl's history adds to this observation is that such collective projects are justified, organized and mobilized not only in accordance with civic values of solidarity and egalitarianism, but also – and in Perl's case, more so – by drawing on an inspirational order of worth and its value system centered around creativity, authorship and passion.

Lessons drawn from Perl's history carry practical value at a time when the web's past is often called upon to imagine alternatives to its present. Revitalizing the open web or building sustained alternatives to what are broadly perceived as the failings of corporate platforms is certainly possible, and both disaffection with the present state of the web as well as conscious efforts to address this are clearly visible. However, I would argue that the success of these efforts will depend not just on how well they match up with more-or-less shared notions of fairness in political economic terms, but also whether they offer an inspired vision of transformation.

Notes

1. Hereafter I refer to the World Wide Web as 'the web,' and this article adheres to conventional distinctions between the web and the internet as well as other networked computing systems such as Usenet. Although Perl is used widely in a range of applications built on top of the internet and other computer networks, as I outline in this article its history is particularly entangled with that of the web.
2. "Having it both ways" is consciously similar to Perl's central motto, "there's more than one way to do it." In addition to embracing a recursive relationship between the two phrases, I've chosen the former because I wanted to emphasize the simultaneous presence of two value systems in a single person, social group or object.

Acknowledgements

Thank you to Robert Prey and the anonymous reviewers for their helpful comments.

Funding

Nederlandse Organisatie voor Wetenschappelijk Onderzoek

References

- Barbrook, R. (1998). The Hi-Tech gift economy. *First Monday*, 3(12). Retrieved from http://www.firstmonday.org/issues/issue3_12/barbrook/
- Boltanski, L., & Thévenot, L. (2006). *On justification: economies of worth*. (C. Porter, Trans.) (New Ed edition). Princeton: Princeton University Press.
- Bourdieu, P. (1993). *The field of cultural production*. In R. Johnson (Ed.) (1st ed.). New York: Columbia University Press.
- Bunce, T. (1994). The Perl 5 Modules List (long). comp.lang.perl. Retrieved from <https://groups.google.com/d/msg/comp.lang.perl/nD0NTU4hxyz/uuUsF2j7Ud4J>
- Coleman, E. G. (2012). *Coding freedom: The ethics and aesthetics of hacking*. Princeton: Princeton University Press.
- Davis, E. (n.d.). Re: Larry Wall. Retrieved March 8, 2018, from <https://web.archive.org/web/19991009192946/http://www.feedmag.com/re/re172.2.html>
- Garfinkel, S. (1997). Perl: the web is its oyster. Retrieved March 14, 2018, from <https://www.wired.com/1997/01/perl-the-web-is-its-oyster/>
- Hauben, M., & Hauben, R. (1997). *Netizens: On the history and impact of usenet and the internet* (1st ed.). Wiley-IEEE Computer Society Pr.
- Kelty, C. (2008). *Two Bits: The cultural significance of free software*. Durham: Duke University Press Books.

- Kim, E. E. (1998). A conversation with Larry Wall. Retrieved March 8, 2018, from <http://www.drdobbs.com/a-conversation-with-larry-wall/184410483>
- Leonard, A. (1998). The joy of Perl. Salon. Retrieved March 14, 2018, from https://www.salon.com/1998/10/13/feature_269/.
- Lialina, O. (2005). A vernacular web. Retrieved March 17, 2018, from <http://art.teleportacia.org/observation/vernacular/>
- Markoff, J. (1993). A free and simple computer link. Retrieved January 19, 2011, from <http://www.nytimes.com/library/tech/reference/120893markoff.html>
- Maxwell, N. (1996). The Perl institute. *The Perl Journal*, 1(3). Retrieved from http://www.foo.be/docs/tpj/issues/vol1_3/tpj0103-0009.html
- McHugh, J. (1998). For the love of hacking. Retrieved March 6, 2018, from <forbes/1998/0810/6203094a>
- McMillan, R. (1997). Perl: the challenges ahead. Retrieved March 18, 2018, from <https://web.archive.org/web/19990203011331/http://www.sunworld.com/swol-08-1997/swol-08-wall.html>
- Michalski, B. (1999). What the heck is a Perl Monger?! Retrieved March 16, 2018, from <https://www.perl.com/pub/1999/01/foy.html/>
- Morozov, E. (2013). The Meme Hustler. *The Baffler*, (22). Retrieved from <http://www.thebaffler.com/salvos/the-meme-hustler>
- O'Reilly, T. (1998). The open-source revolution. *Release 1.0*, 3–26.
- Raymond, E. S. (1999). Revenge of the hackers. In C. DiBona, D. Cooper, & M. Stone (Eds.), *Open sources 2.0: the continuing evolution* (Vol. 1, pp. 207–219). Sebastopol, CA: O'Reilly Media, Inc.
- Raymond, E. S. (2003). *The art of UNIX programming*. Addison-Wesley Professional.
- Schwartz, R. (1996). Announcing: The Perl Institute: Helping people help Perl help people. comp.lang.perl.misc (Usenet). Retrieved from <https://groups.google.com/d/msg/comp.lang.perl.misc/ow2bcc8TmVQ/Pl0e0Wlm9HwJ>
- Silver, D. (2008). History, hype, and hope: An afterward. *First Monday*, 13(3). Retrieved from <http://firstmonday.org/ojs/index.php/fm/article/view/2143>
- Stallman, R. (1985). The GNU manifesto. Retrieved from <https://www.gnu.org/gnu/manifesto.en.html>
- Stark, D. (2011). The sense of dissonance: Accounts of worth in economic life. Princeton University Press.
- Tamiya, M. (2001). LWN interview: Larry Wall. Retrieved March 8, 2018, from <http://lwn.net/2001/features/LarryWall/>
- Wall, L. (1984a). HARK! The rn kit has been posted. net.news (Usenet). Retrieved from https://groups.google.com/d/msg/net.news/1qvOELAhg7A/Ewk6hGu9_AcJ
- Wall, L. (1984b). rn—first blood. net.news.b (Usenet). Retrieved from <https://groups.google.com/d/msg/net.news.b/hel8-vVnb8c/zT1CN200Ys4J>
- Wall, L. (1985). The REAL rn (rn 4.3 - part 0 of 9). mod.sources (Usenet). Retrieved from https://groups.google.com/d/msg/mod.sources/ZGohbRfEi2w/a9BZgi_KnjoJ
- Wall, L. (1988a). Perl 1.0 patch #1. comp.sources.d (Usenet). Retrieved from <https://groups.google.com/d/msg/comp.sources.d/8lqjvcfoz6s/PRxoaaMulAJ>
- Wall, L. (1988b). v13i001: Perl, a “replacement” for awk and sed, Part01/10. comp.sources.unix (Usenet). Retrieved from <https://groups.google.com/d/msg/comp.sources.unix/Njx6b6TiZos/X-JaOCXhPrsJ>
- Wall, L. (1988c). “Free” software. comp.sources.d (Usenet). Retrieved from <https://groups.google.com/d/msg/comp.sources.d/HmNVNWuxN1o/srlQwmj6F10J>
- Wall, L. (1989). v20i084: Perl, a language with features of C/sed/awk/shell/etc, Part01/24. comp.sources.unix (Usenet). Retrieved from https://groups.google.com/d/msg/comp.sources.unix/5_Hg-th6l7w/5pdJ7frTphwJ
- Wall, L. (1991). The Artistic License. Retrieved from <https://opensource.org/licenses/Artistic-Perl-1.0>
- Wall, L. (1997). Perl culture. Retrieved March 20, 2018, from <https://web.archive.org/web/20080924193858/http://www.wall.org:80/~larry/keynote/keynote.html>

Wall, L. (1999). Perl, the first postmodern computer language. Presented at the LinuxWorld, San Jose, CA. Retrieved from <http://www.wall.org/~larry/pm.html>

Wolf, G. (1994). The (second phase of the) revolution has begun. Retrieved February 16, 2016, from <http://www.wired.com/1994/10/mosaic/>