7

Not Just for Fun

Geoff Cox and Alex McLean

There is something inherently human about the ability to perform creative actions with verbal language expressed in the form of jokes. To Paolo Virno, jokes are not simply Freudian clues to the unconscious, but diagrams of innovative action that represent how humans can diverge from social norms.[1] This chapter argues for something similar in the way fun is had with software which exemplifies the innovative action of code in addressing software norms. This indeed is common in critical approaches to making software, where fun often indicates the use of irony or satiric humour by programmers to jolt the user out of their usual interactions with normative tools and productive work. But, as the distinction between work and play becomes ever more blurred, fun is an expected part of all kinds of serious production; indeed fun like most of lived experience has become subsumed into the 'social factory'.

That fun can be had with software is demonstrated in the numerous titles that make explicit reference to it: for instance *Fun with Computers* by T. Ramasami, or the many examples of technical manuals that try to convince the user that things are far easier and more fun than the otherwise serious reality of coding. A quick search reveals numerous examples of this tendency: *Fun and Games with the Computer* by Edwin R. Sage; *Fun with Computers* by A. Roy Chowdhury; *Fun with Computers and BASIC* by Donald D. Spencer; *Fun with Computer Electronics* by Luann Colombo and Peter Georgeson; (and even more assertively) *Computers are Fun* by Tony Gray and Carl Billson; and *The Fun of Programming* by Jeremy Gibbons and Oege de Moor.[2] Fun is a useful way of encouraging productive work with computers and viewing some of the underlying difficulties as enjoyable challenges. This is perhaps particularly noticeable with game development, exemplified by Raph Koster's *A Theory of Fun for Game Design*, where the inherent fun of computer games is understood through cognitive science as 'the feedback the brain gives us when we are observing

patterns for learning purposes', taking players beyond mere entertainment value and suggesting 'alternatives to fun'.[3] Deviating from the norm in this way is widely regarded as an essential part of the process of capitalist innovation as well as its critique; fun in this sense seems to have become instrumentalized.

A further example is Linus Torvalds's book title *Just for Fun: The Story of an Accidental Revolutionary*, written with David Diamond, which is part autobiography and at the same time charts the serious development of the GNU/Linux operating system.[4] The development of the Linux kernel makes a good case study as it emerges from a culture of sharing across code repositories and software distribution networks, like sharing a good joke. Yet this is to be expected as the history of sharing is as old as the sharing of recipes, as Richard Stallman puts it, 'as old as computers, just as the sharing of recipes is as old as cooking'.[5] With free software development, each individual's work is valued in the context of the multiple efforts of all contributors, indeed it emphasizes that innovation and value creation need to account for a deeper understanding of collaborative processes and the social relations that arise from these cultures of sharing.

This point is also what Adrian Mackenzie develops when he claims that the Linux kernel represents a particularly unstable relation to commodified software and hardware as performative action. He thinks that the way in which Linux is produced and continually changed cannot be separated from its performative structure as code, and describes it as a performative 'speech act' that produces an uncertain relation between the code object (the Linux kernel) and the code subject (the programmers who use it), and thus challenges its property relations and corporate relations of production.[6] It diverges from the software norm and innovates in ways that follow the logic of free software development. Fun in this case coexists with the serious business of making source code freely available and open to further modifications. But rather than declaring development by accident in the case of Torvalds, this chapter makes reference to Virno's conception of jokes as an expression of innovative actions drawn from the kernel of political possibilities across human and program languages. Thus the chapter is a speculation on the source code of (software) jokes, and explores this at the intersection of human and machine interpretation.

On jokes

The point for Virno is not the content of jokes, but that newly invented forms diverge from established rules and perceived norms as a result of the innate creativity embedded in language itself. The chapter extends this to program language, to coding practices and, in particular, the ways that code jokes most often do not operate within the confines of formal language, but either outside it in comments, variable names and layout (as with code poetry or codeworks) or in the development of playful esoteric languages (such as brainfuck and befunge, which will be introduced later). It is not that jokes are embedded in the machine's unconscious, of course, but that jokes can be based on innovative deviations from the conventions of coding practices and their formal expression and interpretation.

Before discussing Virno's ideas in more depth, it is worth considering some other commentaries on jokes and the phenomenon of humour from the wider sources at hand. There are far too many to mention but Sigmund Freud's reflections on jokes and the 'wittiness' of dreams is clearly an important reference. In *The Joke and Its Relation to the Unconscious*, first written in 1905, Freud draws upon his theory of the unconscious to explore the idea that jokes express hidden desires and fantasies.[7] Joke-work, like dream-work, is thereby understood as a means of negotiating the ways that consciousness censors and disguises intellectual processes. Joke techniques of condensation, displacement, representation by absurdity or by the opposite, indirect representation, and so on are all present in dreams. But whereas the dream is asocial and serves to spare displeasure, jokes are distinctly social and operate to gain pleasure by releasing inhibitions and discharging excess energy through laughter which involves the whole body. Both share the complex workings of the psyche that does not express itself by accident even when appearing to mash up ideas or develop an operating system. This social aspect of jokes is crucial.

Writing more recently, the philosopher Simon Critchley has examined the importance of humour, emphasizing how a change of situation exerts a powerful critical function, of particular relevance to creative practices. In an interview for *Cabinet* magazine in 2005, drawing upon his book *On Humour* (2002), he emphasizes humour as a social practice that reveals wider social insights.[8] Repression is clearly part of this, with the example given from Plato's *Republic* where it is explained that the guardians of the *polis* were not meant to laugh because laughter was considered too uncivilized and bestial. It is because

laughter is considered deviant that jokes are seen to exhibit the potential to be a powerful critical tool.

Yet Critchley also explains how witticism (as a sophisticated style of humour) is also considered to be a sign of advanced civilization as opposed to common jokes, where witticism is tied to the development of liberal democracy, and characterized by the figure of the dandy or wit. The witticisms of Slavoj Žižek seem to exemplify this, including his many jokes related to the subject of really xisting Socialism.[9] One example he gives is a joke that reveals the futility of dissident protest and the lack of recognition of wider conditions. His claim is that no one really took the subject of totalitarianism seriously but rather as a joke; and that progressive intellectuals who thought it was serious merely indicated the futility of their protests. As we can see from the above, the function of humour for Žižek lies in its ability to invert common sense, following the Marxist-Hegelian logic of dialectical reversal. Moreover, conceptualizes such things in terms of the 'return of the real' (reworking Freud's 'return of the repressed'), where impulses previously repressed erupt into social life at unexpected moments and confound previously held certainties. This explains how we laugh in bad taste, and as a way of coping with the disappointment of our lived realities. Can we also ask whether anyone really thinks Windows or Mac OSX are serious operating systems in similar terms? Or indeed, that their free software alternatives are rendered as jokes too (to counter seriousness), and reveal their futility as they in turn become commercialized (as in the case of Ubuntu perhaps)?

The point we are making is that it is possible to draw some analogies between jokes and code, as witty utterances hidden behind the surface of normative software production and totalitarian server-client architectures that really are a joke. This issue is also explored in Inke Arns's essay, 'Read_Me, Run_Me, Execute_Me', with its subtitle 'Software and its Discontents',[10] implying that the performative dimension of code lies repressed, as with the description of the Linux kernel cited earlier,[11] and further evoking Freud's *Civilisation and its Discontents,* that capitalism was founded on the repression of the libido. In post-Marxist thinking (where the constitution of subjectivity is considered more fully), something similar can be detected in the way that repression is considered to be socially determined and can only be released by freeing productive desire, perhaps expressed as nervous laughter. Drawing these threads together in the 1970s, Herbert Marcuse's *Eros and Civilisation* and, to a greater extent, Gilles Deleuze and Félix Guattari's *Anti-Oedipus: Capitalism and Schizophrenia* argue

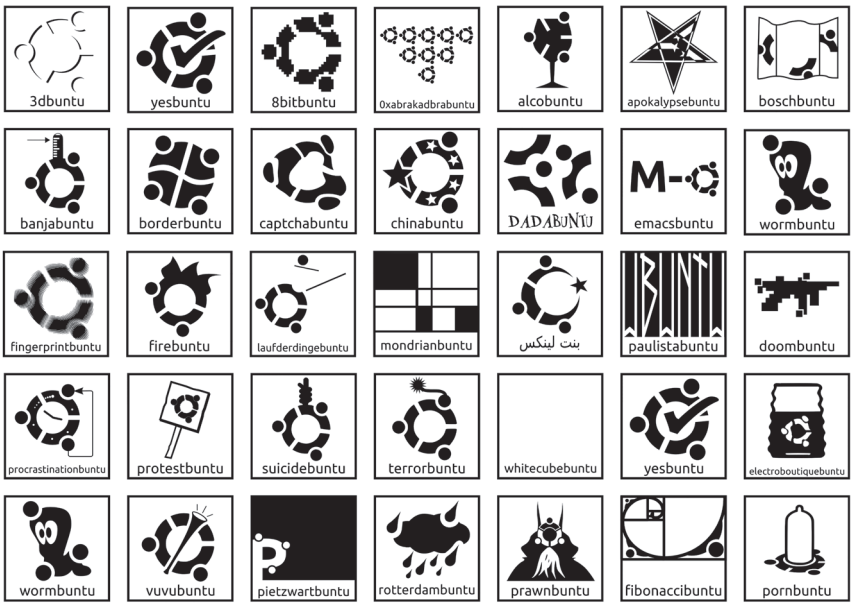


Figure 7.1 Sample of logos from Gordan Savičić and Danja Vasiliev's *120days of *buntu* (2011)

Source http://120buntu.com/

for the liberatory potential of desiring-production.[12] So in this way the release of free software can be understood as less an imaginary force based on lack (as in Freud), and more a real, productive force, as a neat example of a 'desiring-machine..[13] A good example of this is perhaps Gordan Savičić and Danja Vasiliev's project *120days of *buntu* which proposes 120 modified humorous and useless Ubuntu Operating Systems.[14] By making playful reference to de Sade's *The 120 Days of Sodom,* the suggestion is that alternative operating systems might be able to liberate desire, in excess of the masochistic desires of free/libre software development in which Ubuntu is seen to operate too close to normativity.[15]

Similarly, Franco 'Bifo' Berardi detects the fundamental political struggle between machines for liberating desire and mechanisms of control over the imaginary.[16] Thus, to Berardi the various liberatory strategies such as refusal of work, the invention of temporary autonomous zones, free software initiatives and so on offer 'dynamic recombination'.[17] Therefore, and further developing the analogy between repression and the performativity of software, free software development (where the code is made openly available/shared) might offer

therapeutic assistance in putting the programmer in touch with his/her, and indeed culture's, sublimated desires repressed under proprietary software development.[18] If normative software could be thought of as software without a body, might desiring-production be explored through free software development with a body? As Christopher M. Kelty describes it, 'geeks' share a social imaginary about the production of actually existing alternatives, and as such the free software movement is an example of a 'recursive public', capable of creating and modifying the domain or platform through which they act.[19] In other words, they share a culture through which their repressed desires find release in the public domain and body politic.

The ideology of the Free Software Foundation clarifies this point with a humorous play on the ambiguity of freedom in 'you should think of 'free' as in "free speech", not as in "free beer"'.[20] Commercial (free market) interests seek to assert their control over free software by replacing the word 'free' with the phrase 'open source', placing emphasis on visibility of code rather than freedom to share.[21] Moreover, the broader ideological issues are evident in the parallel narratives of the development of free and open source software development. On the one hand, there is free software referring back to the 1980s when software freedom was meant in resistance to proprietary software, and on the other, open source that emanated from arguments about its economic benefits and in parallel to free market thinking. Furthermore, like the joke from Žižek, the futility of progressive alternatives like Ubuntu run the risk of missing the mark unless the broader political issue of the ability to modify the domain or platform through which these desiring-practices are enacted is also considered.

That freedom of speech relates to free software (at least according to the Free Software Foundation), and not free beer, may be one of the analogies that leads commentators (such as Mackenzie) to discuss the performative dimension of software and its relation to speech act theory, making reference to John Langshaw Austin's *How To Do Things With Words*.[22] Although the analogy between program code and speech acts has become rather commonplace since it was made by Terry Winograd and Fernando Flores in 1987,[23] and is perhaps left a little general here, the point is to emphasize the ways in which programmers express themselves through the witty manipulation of layers of representation, including symbols, then words, language and notation. But clearly code is a special kind of language, and one that can automatically enable and disable certain kinds of actions.[24] It does this through its special ability to convert action to language and this is where the joke lies.

Also drawing on Austin, Virno refers to linguistic innovation as: 'how to do new things with words', where words constitute an action in and of themselves – like program code in as much as when it is interpreted it says something and does something at the same time. Virno's interest is in the ability of the human animal to execute 'innovative actions' capable of modifying 'consolidated norms'.[25] A key reference underpinning this is Noam Chomsky's description of the apparatus of power as repressing the innate creativity of verbal language, and to Aristotle's description of contingency at the heart of our use of language (in *Ethics*),[26] as well as the more general point that intellectual and linguistic labour are no longer separated from general conditions of informational capitalism. In Virno's opinion, rhetorical persuasion and the concept of the public sphere in which speech is paramount, demonstrate the ability of language to establish social relations between a 'mass of speakers', that is necessarily shared and collective.[27] He emphasizes that the 'language system is a social fact' wherein the human animal is 'ready made for language, but not actually in possession of it' until it starts interacting in the social realm.[28] This is part of early years development (the 'mirror stage', in Lacanian terms) but remains evident in every utterance made thereafter. For Virno, this confirms the biopolitical dimension of the human animal in the world, and the social importance of language and the sharing of codes.

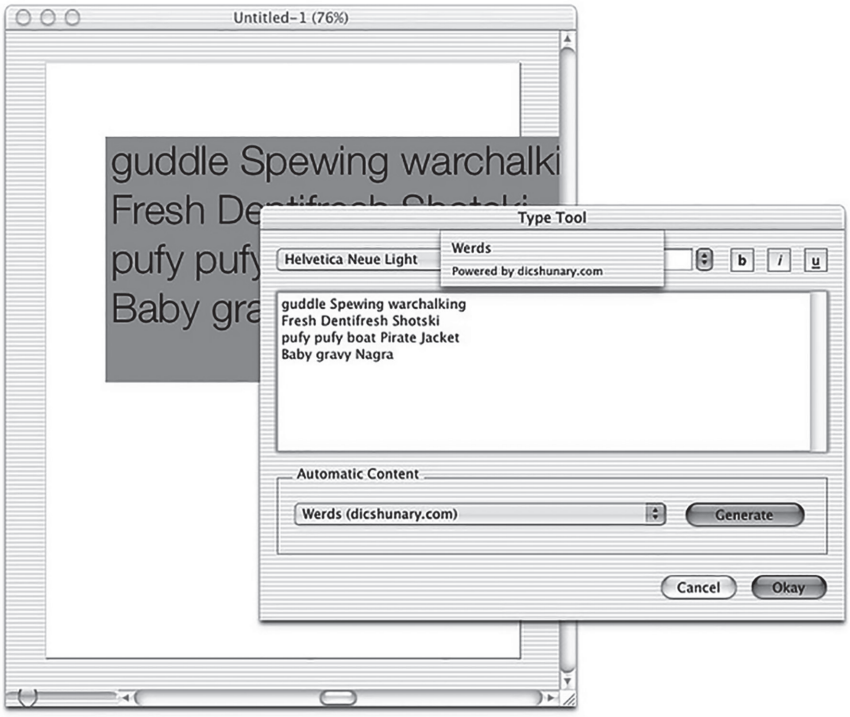
According to Virno, underpinning political possibilities is the simple fact that the human animal is capable of modifying its forms of life.[29] This is what makes for innovation in the general sense that newly invented forms might diverge from established rules and perceived norms – based on the Chomskian innate creativity referred to above, and perhaps also to the idea that jokes can be understood as hard-wired (in as much as creativity relates to playing with language). And this is where jokes also figure as an example of how humans diverge from social norms, how 'linguistic animals give evidence of an unexpected derivation from their normal praxis', as Virno puts it.[30] For him, witty utterances are similar to the performative utterances that Austin described, where words constitute an action in and of themselves.[31] But as already emphasized, the point for Virno is not the content of jokes, which might poke fun at social norms, hierarchies or the ruling order, for such jokes tend to obscure what is important: the apparatus or the '*logicolinguistic resources* that jokes utilize'.[32] His argument is that innovative action uses these resources like a toolbox or library. In this way, it produces ambivalences: oscillating between the 'determined *rule* and the *regularity* of species-specific forms of conduct'.[33] Such

contradictory factors characterize the social character of the human species and its innovative force, despite its repression by power structures. In other words, jokes demonstrate innovative techniques and the possibilities for transforming the linguistic operating system. This happens in two main ways according to Virno: first by demonstrating how divergences in following rules often result in changing the rule itself (put differently the application of the norm also contains surprises, and situations where the rule is broken and justified in terms of exceptional circumstances[34]); and secondly, through the incorrect use of semantic ambiguity, an 'error' (or glitch).[35] The rules are not only there to be broken, but applied differently, adapted and modified, and ultimately transformed. So how does software demonstrate similar possibilities at the level of code, and where the modifications undermine normalized behaviour?

Code jokes

It would be relatively easy to mention the use of parody in this connection to software, but perhaps this is too predictable under present conditions and its effectiveness thereby questionable in a similar manner to a political joke that fails to register its wider effects once executed. So although an example like Gordan Savičić's *sing_slavoj_sing* might poke fun at the seriousness of Žižek's philosophizing, by hacking *billy the fish* and replacing the soundchip with a good quote or two, this is not the kind of joke that we wish to emphasize for our argument as it does not present an intervention in terms of the apparatus of language.[36] If humour is somewhat hard-wired, then jokes are an inevitable component of the structures of language where humans demonstrate their innate ability to innovate new forms. But can code jokes be reduced to their functional aspects in this way, as procedures that are open to recombination as language is more generally? Our concern is about jokes at the level of changing rules and the use of semantic ambiguity, in keeping with Virno's emphasis thus far.

Such tendencies can be detected in the example of Signwave's *Auto-Illustrator*, an experimental, semi-autonomous, generative software artwork that includes a wide range of generative and procedural techniques, packaged as a fully functioning parody of Adobe's vector graphics application *Illustrator.*[37] Humour is an important part of the software package as indicated, not least, by the release date of version 1.2 on April Fool's Day, in 2003. The new release included parody plug-ins that serve to question how contemporary software should 'behave',



Figures 7.1 and 7.2 Screenshots of Signwave's *Auto-Illustrator* (2001): 'Dicshunary' spelling tool and 'Psychosis' preferences option

Source http://swai.signwave.co.uk

including one function that would shut the computer down with no notice if the software 'decided' the design was unworthy. The user's expectations were challenged by presenting tools and functions that do not conform to expectations of what software usually does as a tool. Perhaps most controversial was the terms of use which insisted that designs using the software were co-credited to the author and the software company. This caused some outrage by users who assumed the company were overstepping their proprietary rights and imposing measures beyond a joke. Yet, to Signwave, this seemed to be an entirely logical statement which recognized the difficulties of making any claim of single authorship, and moreover exemplified the principle that the making of software can be regarded as an artwork in itself. Thus Signwave extended some of the ambiguities built into software production into a playful form that relates to the way social relations are organized with normative software development. If *Auto-illustrator*'s satire of Adobe's *Illustrator* operates at the level of code in a general sense – in the domain of intellectual property within the legal system, itself a system of rules, codes and speech acts – could this also work in the domain of source code?

To repeat the Virno formulation, jokes are identified as operating in two significant ways: first by demonstrating how divergences in following syntactical rules often generate a change in the rule itself; and secondly, through the ironic use of semantic ambiguity. This perhaps happens in a general way with *Auto-illustrator*, but how do these ideas translate to code more precisely? In this sense, it could be said that we wish to crack the source code of jokes by examining the concept of interpretation in more detail. The interpreter is a codification of the language in which the source code is expressed, or, in other words, a partially evaluated computational process.[38] The source code is not a program on its own, rather it is a replaceable component of a larger computation. The relationship between source code and interpreter is recursive; the rules behind an interpreter are themselves implemented in source code, requiring another interpreter. If we follow these recursive layers of interpretation we find hardware microcode that mediates between the discrete digital world and our continuous physical world.

With few exceptions, mainstream programming languages are Turing-complete. This means it is possible to write a program in any of the languages that interprets any other computer language. Therefore the rules of any language can be changed simply by writing an interpreter for another language within it; and thereby the scope for breaking rules is boundless. Turing-completeness extends beyond mainstream language into some surprising places. Simple

cellular automata, including the well-known *Game of Life* by John Conway and Stephen Wolfram's *Rule 110*,[39] even some configuration files such as that of the 'sendmail' electronic mail transport software, have turned out to be Turing-complete. The fact that Turing-completeness comes so easily to these invented languages, allowing any symbolic constraint to be broken at will, provides a programmer's playground with glimpses of the infinite.

With this emphasis on breaking rules, it is no surprise that source code humour centres on interpretation of code. *Esoteric* programming languages are those which take a humourous approach to language design, challenging the norms of source code interpretation. For example, obfuscated programs written in the brainfuck language consist entirely of punctuation, each of the eight characters '><+-.,[]' representing a single elementary operation. Brainfuck is Turing-complete, and so can be used to write any program in theory, but in practice it is extremely difficult to write any program in it at all. Below is *The Game Of Life* implemented in brainfuck.[40]

```
  +>>++++[<++++>-]<[<++++++>-]+[<[>>>>+<<<<-]>>>>[<<<<+>>>>>>+<<-]<+
   +++[>+++++++++<-]>.[-]<+++[>++++<-]>+[>>.+<<-]>>[-]<<<++[<++++++>-]<.<<[>>>>+
 <<<<-]>>>>[<<<<+>>>>>>+<<-]<<[>>>>.+<<<+++++++++[<[>>+<<-]>>[<<+>>>>>++++++++
 +++<<<-]<[>+<-]>[<+>>>>+<<<-]>>>[>>>>>>>>>>>>>+>+<<     <<<<<<<<<<-]>>>>>>>>>>
>>[-[>>>>+<<<<-]>[>>>>+<<<<-]>>>]>       >>[<<<+>>  >-    ]<<<[>>+>+<<<-]>[->[<<<
<+>>>>>-]<[<<<  <+>       >>>-]<<<< ]<     ++++++  ++       +[>++++<-]>>[<<+>>>-]<
<[>---<-]>.[- ]          <<<<<<<<< <       <<<<<< <          -]++++++++++.[-]<-]>>>
>[-]<[-]+++++            +++[>++++         ++++<      -     ]>--.[-]<,----------[<+
>-]>>>>>>+<<<<< <      <[>+>>>>>+>[       -]<<<       <<   <<-]>++++++++++>>>>>[[-]
<<,<<<<<<<->>>> >     >>[<<<<+>>>>>-]<<<<[>>>>+       >+<<<<<-]>>>>>----------[<<<<
<<<<+<[>>>>+<<<       <-]>>>>[<<<<+>>>>>>+<<<-       ]>[>-<-]>++++++++++[>++++++++
++<-]<<<<<<[>>>       >+<<<<-]>>>>[<<<<+>>>>>       >+<<-]>>>>[<<->>-]<<++++++++++
[>+<-]>[>>>>>>>       >>>>>+>+<<<<       <<<<<       <<<<-]>>> >>      >>>>>>>[->>>
>+<<<<-]>[>>>>        +<<<<-]>> >        ]>> >            [<< <         +>>>-]+<<<[>
>>-<<<-]>[->[<       <<<+>>>>-]          <[ <             < <           <+>>>>-]<<<
<]<<<<<<<<<<, [     -]]>]>[-+++         ++                +    +++      ++[>+++++++
++++>+++++++++ +     +<<-]>[->[>>>       +<<<-      ]>>>[ <     <<+       >>>>>>>+>+<
<<<<-]>>>>[->[> >     >>+<<<<-]>[>       >>>+< <    <<-]> >     >]>       >>[<<<+>>>-
]<<<[>>+>+<<< -      ]>[->[<<<<+>       >>>-] <    [<<< <     +>>        >>-]<<<<]<<
<<<<<<[>>>+<< <      -]>>>[<<<+>>       >>>>> +    >+<< <               <<-]<<[>>+<<
-]>>[<<+>>>>>       >+>+<<<<<-]>>       >>[-[ >    >>>+ <             <<<-]>[>>>>+<
<<<-]>[>>>>+<       <<<-]>>]>>>[ -     ]<[>+< -     ]<[ -            [<<<<+>>>>-]<<<
<]<<<<<<<<]<<       <<<<<<<<<++++ +    +++++  [    >+++ +     ++++++[<[>>+<<<-]>>[<<+
>>>>>++++++++ +     ++<<<      -] <    [>+<- ]     >[<+ >     >>>+<<<-]>>>[<<<+>>>>-]
<<<[>>>+>>>>  >     +<<<<      <<      <<-]> >     >>>>        >>>[>>+<<<-]>>[<<+<+>>
>-]<<<------ -     -----[      >>       >+<<< -     ]>>>        [<<<+> > >>>>>+>+<<<<
<-]>>>>[-[>> >     >+<<<<     -] >      [>>>> +     <<<<-        ]>>> ]   >>>[<<<+>>>-
]<<<[>>+>+<< <     -]>>>      >>           > >     [<<<+               >>>-]<<<[>>>
+<<<<<+>>-                    ]>            >      >>>>>[<              <<+>>>-]<<<[>
>>+<<<<<<<                    <<+           >      >>>>>-]<          <<<<<<[->[<<<<+
>>>>-]<[<<<<+>>>>>-]<<<<]>[<<<<<<     <+>>>       >>>>-]<<<<     <<<<<+++++++++++[>
>>+<<<<-]>>>[<<<+>>>>>>>+>+<<<<<-]>>>>[-[>      >>>+<<<<<-]>[>>>>+<<<<-]>>>]>>>[<<<
+>>>>-]<<<[>>+>+<<<-]>>>>>>>[<<<<+>>>>-]<<<[     >>>+<<<<<+>>>-]>>>>>>>[<<<<+>>>>-]<<<
[>>>+<<<<<<<<<<+>>>>>>>-]<<<<<<<[->[< <  <     <+>>>>>-]<[<<<<+>>>>>-]<<<<]>[<<<<<<<
+>>>>>>>-]<<<<<<<<<<++++++++++[>>> >         >>>+>+<<<<<<<<<-]>>>>>>>[-[>>>>+<<<<-
]>[>>>>+<<<<-]>>>]>>>[<<<+>>>>-]<<< [        >>+>+<<<<-]>>>>>>>[<<<+>>>>-]<<<[>>>+<<
<<<+>>>-]>>>>>>>[<<<+>>>>-]<<<[>>>+<         <<<<<<<<+>>>>>>>-]<<<<<<<[->[<<<<+>>>>-
 ]<[<<<<+>>>>>-]<<<<]>[<<<<<<<+>>>>>      >>>-]<<<<<<<----[>>>>>>>+<<<<<<<+[>>>>>
 >>-<<<<<<<[-]]<<<<<<<[>>>>>>>>>>>>>+>+<<<<<<<<<<<<<-]<[   lft@df.lth.se   ]>>>>>
   >>>>>>>[-[>>>>+<<<<-]>[>>>>+<<<<-]>[>>>>+<<<<-]>>]>>>[-]<[>+<-]<[-[<<<<+>>
      >>-]<<<<]<<<<<<[-]]<<<<<<<[-]<<<<-]<-]>>>>>>>>>>>[-]<<]<<<<<<<<<<]
```

A comment on the left side of the web page where the program is posted, simply states 'get a Life);'.

The tortuous nature of brainfuck escapes the world of computation to be wrought on the human body as bodyfuck, a brainfuck interpreter using computer vision techniques to map from bodily gestures to the brainfuck instructionset (like a desiring-machine). The brainfuck demo shows how much physical exertion is required to produce a short sequence of symbols,[41] and the programmer uses his/her body in a more overt manner than previously required. To take another example, befunge is an esoteric language that breaks the usual downward direction of interpretation through a program to create two-dimensional syntax. This is done using punctuation, as in brainfuck but in a more understandable way; each of the four instructions '^>v<' represent graphical arrows, which change the direction of control flow as you might expect. The question mark character '?' changes the direction in a random direction. The instruction set goes beyond this, and is again Turing-complete, but by following the arrows we can already understand the operation of the following program, which, when read starting from the top left-hand corner, outputs a random number from 0 to 9:

```
vv < <
   2
   ^ v<
v1<?>3v4
   ^ ^
> >?> ?>5^
   v v
v9<?>7v6
   v v<
   8
. > > ^
^<
```

Despite being Turing-complete these interpreters appear useless, and some esoteric languages seem to exist simply as 'in jokes' for geeks, having unimplementable instruction sets. In the case of IRP (Internet Relay Programming) there is no formal instruction set at all, the interpreters are human participants in an internet chat room, a running joke since 2005. Below is the obligatory 'Hello, World!' example:

<GregorR>	Please say 'Hello, World!'
<jix>	Hello, World!

The likelihood of an IRP program being interpreted as you wish is improved if you are polite, however this does not always work:

<GregorR>	Please, write the lyrics to the song 99 Bottles of Beer on the Wall.
<memonic>	go to hell

Recursion is possible, up to a point:

<CakeProphet>	Could someone please ask someone to repeat this request?
<pikhq>	Could someone please repeat the previous request?
<RodgerTheGreat>	Could someone please ask someone to repeat this request?

In recasting English as a machine language and humans as interpreters, we are given the opportunity to examine the relationship between performative phrases and computer code in some detail. In this way, we might begin to understand that jokes can be interpreted in multiple ways which reflect the complexity of human and machine logic that moves beyond simple amusement. Indeed all jokes express a purpose. As in Virno's earlier descriptions, this is where innovative techniques are demonstrated that diverge from established conventions by changing and breaking rules, and playing with ambiguities related to code. This is where other possibilities reside.

But is software development taken too seriously when it is a joke all along? Indeed software development is in danger of losing its sense of humour altogether as it becomes more and more standardized and packaged. For instance, with service-based platforms access to source code is no longer possible, and the differentiation between files, software and network services evaporates altogether. This is an apt description of the Apple iOS paradigm of software development, where users of the Apple iPad and iPhone are allowed only restricted access to programming language interpreters, and software licenses favoured by the Free Software Foundation are forbidden. Access to humour is denied and the example confirms that proprietary logic is a serious business of repression. In Berardi's terms, interpretation has become schizophrenic (like fast speech), and the relations between metaphors and things, representation and life, have become thoroughly confused. This is particularly evident when it comes to language that is more and more influenced by machines, leading to a situation where the learning of language and affectivity

have been separated.[42] On the contrary, what needs to be rediscovered are forms of happiness and laughter tied to collective formations: the sharing of code that liberates desire and mechanisms of control over the imaginary from the serious business and sense of determinism that is normally associated with code.[43]

The reference to Virno's work on jokes draws attention to their function in relation to 'innovative action' in the public sphere. In the case of the title of this chapter, the use of 'not' as prefix to Torvald's title (in 'not just for fun') is in keeping with Virno's comments that the system of language both 'does' negation (by identifying what something is not), and 'is' negation (in as much as it can only signify something): 'The negation, or something that language *does*, is understood, above all, as something that language *is*'.[44] He is speculating here on a non-representational form of politics, and despite recognizing the sovereign forces that restrain such abilities, concludes that humans are capable of adapting themselves and their circumstances in parallel to their linguistic abilities and possibilities for innovative action. We have attempted to consider the execution of program code in similar terms; that the potential for divergent forms comes into existence through social interactions and modifications. Rules are not just there to be broken, but transformed altogether through negation. For instance, the function of humour (to Žižek) lies in its ability to invert common sense. Code jokes remind us of the possibilities of non-deterministic interpretation in this inverse way, as deviations from the conventions of coding as formal expression. What we mean to stress is that having fun with software does not simply encourage people to work with computers (and distract them from the hard work that is unavoidable when programming) but offers a way to rethink political possibilities in public, and to reimagine the seriousness of normative visions of life that code otherwise implies.

Notes

1 Virno, Paolo, *Multitude: Between Innovation and Negation*, trans. Isabella Bertoletti, James Cascaito and Andre Casson (Los Angeles: Semiotext(e) Foreign Agents, 2008).

2 This issue of fun is also something that Christopher M. Kelty raised in his talk 'No Fun. Work, Labor, Action in Free Software', presented at 'The Internet as Playground and Factory Conference', Eugene Lang College, The New School, New York (12–14 November 2009), http://digitallabor.org/. His presentation slides

can be seen online at http://www.scribd.com/doc/22394057/No-Fun-Slides (all accessed 12.12.2013).

3 Cited in Bogost, Ian, 'An Alternative to Fun', in *Unit Operations* (Cambridge, MA: MIT Press, 2006), 118.

4 Torvalds, Linus, *Just for Fun: The Story of an Accidental Revolutionary* (New York: HarperCollins, 2001), http://en.wikipedia.org/wiki/Just_for_Fun (accessed 12.12.2013).

5 Stallman, Richard, *Free Software, Free Society: Selected Essays of Richard M. Stallman*, ed. Joshua Gay (Boston, MA: Free Software Foundation, 2002), 15.

6 Mackenzie, Adrian, 'The Performativity of Code: Software and Cultures of Circulation', in *Theory, Culture & Society*, 22(1): 13.

7 Freud, Sigmund, *The Joke and Its Relation to the Unconscious* (London: Penguin, 2002).

8 Dillon, Brian and Simon Critchley, 'Very Funny: An Interview with Simon Critchley', in *Laughter*, *Cabinet*, 17 (spring 2005), http://www.cabinetmagazine.org/issues/17/dillon.php (accessed 12.12.2013).

9 In this connection, it is well worth watching his performance in a clip from YouTube where he tells an old Russian joke, http://www.youtube.com/watch?v=XEnkDEgALGI (accessed 12.12.2013).

10 Arns, Inke, 'Read_Me, Run_Me, Execute_Me: Software and Its Discontents, or: It's The Performativity of Code, Stupid', in *Read_Me: Software Art & Cultures – Edition 2004*, (eds) Olga Goriunova and Alexei Shulgin, 176–93 (Aarhus, Denmark: Digital Aesthetics Research Centre, Aarhus University, 2004).

11 See note 6.

12 Deleuze, Gilles and Guattari, Félix, *Anti-Oedipus: Capitalism and Schizophrenia*, trans. Robert Hurley, Mark Seem and Helen R. Lane (London: Athlone, 1990).

13 Ibid.

14 Savičić, Gordan and Vasiliev, Danja,*120 days of *buntu* (2011), http://120buntu.com/ (accessed 12.12.2013).

15 See Cox, Geoff, 'Notes on 120 days of *buntu', in *World of the News*, (eds) Geoff Cox and Christian Ulrik Andersen (Berlin/Aarhus: transmediale/DARC, 2012), 9.

16 Berardi, Franco 'Bifo', *Precarious Rhapsody: Semiocapitalism and the Pathologies of the Post-alpha Generation* (London: Minor Compositions, 2009). His performative reading of the source code of the 'I Love You' virus resonates with this description; he read the source code of the virus at the D-I-N-A (Digital Is Not Analog) digital art festival in 2001, http://www.digitalcraft.org/iloveyou/loveletter_reading.htm (accessed 12.12.2013).

17 Ibid., 72.

18 In French, free software is known as 'libre' software. Although it does not have an analogue in English, this unambiguous term evokes a libertarian attitude.
19 Kelty, Christopher M. *Two Bits: The Cultural Significance of Free Software* (Durham, NC: Duke University Press, 2008).
20 See http://www.gnu.org/philosophy/free-sw.html. This was made into more of a joke by the Danish artists Superflex in their project Free Beer, in which they share recipes for free beer, http://www.superflex.net/projects/freebeer/ (both accessed 31–3–88).
21 Releasing and sharing source code therefore represents a number of ambiguities relating to trust, cost, liberty, making free but making money on the stock market instead, a belief in open standards or a cynical business move to capitalize on free labour.
22 Austin, John Langshaw, *How to Do Things with Words* (Cambridge, MA: Harvard University Press, 1962). For an elaboration of this, see Cox, Geoff and Alex McLean, *Speaking Code: Coding as Aesthetic and Political Expression* (Cambridge, MA: MIT Press, 2011).
23 Winograd, Terry and Flores, Fernando, *Understanding Computers and Cognition: A New Foundation for Design* (Reading, MA: Addison-Wesley, 1987).
24 Chun, Wendy, *Programmed Visions: Software and Memory* (Cambridge, MA: MIT Press, 2011), 27.
25 Virno, *Multitude: Between Innovation and Negation,* 20.
26 Ibid., 13.
27 Ibid., 46.
28 Ibid., 47.
29 Ibid., 69.
30 Ibid., 72.
31 Ibid., 85.
32 Ibid., 165.
33 Ibid., 166.
34 The so-called 'state of exception', invoking Carl Schmitt, to describe the way the state legitimates the breaking of its own legal rules in exceptional circumstances. According to Giorgio Agamben, this has become the working paradigm of modern government.
35 Ibid., 73, 74.
36 *sing_slavoj_sing* was developed for DEVICE-ART Zagreb, by fleshgordo (Gordan Savičić) in 2006. See http://www.youtube.com/watch?v=D9FXyr-LLeI (accessed 12.12.2013).
37 See http://swai.signwave.co.uk/ (accessed 12.12.2013).
See Signwave, *Auto-Illustrator Users Guide* (first written 2001), published to

coincide with the boxed set as part of the *Generator* exhibition, Spacex 2002–03, and touring in the UK.

38 Futamura, Yoshihiko, 'Partial Evaluation of Computation Pprocess – An Approach to a Compiler-compiler', in *Higher-Order and Symbolic Computation*, 12(4) (1999): 381–91, http://portal.acm.org/citation.cfm?id=609205 (accessed 12.12.2013).

39 John Conway's *Game of Life* is a cellular automaton devised in 1970, consisting of a collection of cells which, based on a few mathematical rules, can live, die or multiply. See http://www.bitstorm.org/gameoflife/ (accessed 12.12.2013). Similarly Stephen Wolfram's *Rule 110* is an elementary cellular automaton with interesting behaviour on the boundary between stability and chaos that is considered to be Turing-complete, meaning that in principle, any calculation or computer program can be simulated using this automaton. See http://www.wolframalpha.com/input/?i=rule+110 (accessed 12.12.2013).

40 Akesson, Linus, *The Game of Life* implemented in brainfuck, http://www.linusakesson.net/programming/brainfuck/index.php (accessed 12.12.2013).

41 See http://www.nikhanselmann.com/projects/bodyfuck/ (accessed 12.12.2013).

42 Berardi, *Precarious Rhapsody*, 9.

43 As our smiles are clearly not simply fixed like George Maciunas' *Flux Smile Machine* (1971), a device that forces the shape of the user's mouth into a smile. This work was the reference point for the exhibition *Smile Machines*, curated by Anne-Marie Duguet, for the Transmediale Festival for Digital Culture, in 2006.

44 Virno, *Multitude: Between Innovation and Negation,* 50.