

The Open Source Singularity: A Postmodernist View

John Lenarcic

School of Business Information Technology
RMIT University
Melbourne, Victoria, Australia
john.lenarcic@rmit.edu.au

Eric C. Mousset

School of Electrical and Information Engineering
The University of Sydney
New South Wales, Australia
mousset@ee.usyd.edu.au

Abstract

A philosophical meditation on the open source movement as a mechanism for the design and distribution of software is presented. As a development paradigm, the open source method is depicted as being a radical novelty in the gamut of engineering phenomena and an exemplary metaphor for postmodernist design.

Keywords: Software design, software distribution, open source, postmodernism.

1 A sweeping overview of meandering proportions

A singularity in the astrophysical sense refers to the nucleus of a gravitationally collapsed star or “black hole”. It signifies a region of space-time where the laws of physics break down and is shielded from reality by an event horizon. The term “singularity” as used in the title of this paper is a metaphor for the role of postmodernism in contemporary software development: A school of anarchic thought that shatters the traditional wisdoms of the practice and its aspirations to a science yet one that is alien territory to many if not most practitioners of software engineering. Engineering by its very definition is applied science; therefore it would seem that programmers by default must be in league with practitioners of the scientific method. Yet, the term “computer science” is to some extent an oxymoron as the discipline, at least from the vantage of software design, is more a cross between a craft and an art. In this case, it is not solely dictated by unambiguous physical principles but is subject to the myriad dimensions of construal formulated through a tension between the dynamic human constraints of cognition and emotion.

As a controversial branch of interpretive philosophy, postmodernism itself is open to a myriad of readings. A core foundational suite of premises and motivations may be found though that applies to most epistemological factions. The present study may choose alternatively, and warn the reader thereof from such common ground, or from connotations more specific to a given theoretical base or a given author. The arguments outlined within should be considered within the spirit of the branch of thinking that they are elucidating upon. Postmodernism embraces subjective experience and shuns objectivity. Reflexivity and relativity reign supreme in this line of thought with an emphasis on self-conscious analysis applied to a particular procedure rather than a finished work. Opinion is revered and even considered to be a super set of processes such as the scientific method. Within this sphere of reasoning, even the latter can appear to rest on pliable foundations, as empirical data, controlled experimentation, statistical analyses and mathematical logic are manifestations of human conventions with all their concomitant foibles.

In such a framework, questions such as the nature of human experience, the structure of knowledge, the mechanisms governing its legitimacy, control or circulation, and the vehicles for creative self-determination at both a cognitive and emotional level are central to both postmodernist thought and this present reflection. The shared representation of these factors is subject to linguistic constraints as will be revealed.

A primary impetus for the ideas expounded in this discourse is found in the observation by Lyotard (1984) that the obsolescence of the metanarrative apparatus of “legitimation” results in the emergence of “institutions in patches” and “local determinism”. In other words, the justification of widely held bodies of authoritative knowledge in any domain can often become an outmoded custom and what is a monolith of common sense one day can fracture to become a set of disparate though vaguely related beliefs the next. Descriptions about descriptions are malleable and subject to change over time. In such a recursive tableau, Lyotard emphasizes the importance of the role played by what he calls “language games” at both individual and collective levels, similar to the arguments adopted by Wittgenstein (1994). For example, there are

some words that are notoriously difficult to comprehensively define yet people are still aware of the generic conceptual attributes denoted by that term (e.g. the word “game” itself). Some words can be so abstruse that the quest for their elusive meaning is designed more to obfuscate than elucidate, a pertinent illustration being some of the words in this very sentence. Verbal statements of a self-referential nature can also be paradoxical in meaning, a case in point being: “Please do not read this sentence.” The value in all of this could be that wisdom can sometimes be the residue of confusion. Terms such as “postmodernism” and “singularity” are semantic prisms that serve as factories of argument in themselves. Of course, this understanding is often culturally biased and highly variable for that very reason. In this respect, the evolving design methodologies and models that delineate the gamut of software engineering can be regarded as a constantly shifting archipelago of cultural beliefs, subject to the ebb and flow of ideology intermingled with popular opinion masquerading as absolute truth.

Low et al (1996) outline a postmodernist perspective on software engineering, claiming that it can no longer be viewed as a deterministic endeavour with a concurrent unitary purpose, methods and notions of truth. This view is particularly applicable to the reusable universe of code promoted by object-oriented design with its reliance on the intertwined principles of inheritance, information hiding and polymorphism. However, the rise of the open source approach in software engineering is another possible self-evident demonstration of postmodernism in practice, as will be discussed below.

2 Open-source as a tool of thought

In its analytical dissection of this software movement, the present manuscript does not aim at revisiting or challenging accepted definitions be they from influential individuals (e.g., Raymond, 2001; Stallman, 2002) or organizations (e.g., Open Source Initiative (2004) and the Free Software Foundation (2004)). The fine distinction between the definitions offered by the latter advisory bodies with regard to the commercial destiny of software artefacts is valid, but falls beyond the scope of the present reflection. The meaning of the term open-source software, as utilised in this text, has no pretension to prevail outside these established boundaries. It has been chosen deliberately broad in order to cover and refer to all instances of *freely modifiable and redistributable* software. In order to ascertain that a software instance does fall in such category, a common routine consists of providing the recipient with the original, human-readable, human-editable form of the computer program, namely the *source code*. This is in contrast to the *compiled*, or *executable* computer code, which, in most cases, is opaque to humans but necessary to the operation of the software by the machine. Whether a given software instance is made freely modifiable and redistributable by opening access to its source code or by different means, it will still be referred to as open-source within the scope of this text.

In addition, there are other characteristics attached to the open-source distribution and design paradigm, which do not appear explicitly in the reference above but need to be enclosed within the scope of the study. These include the clear negation of liability with regard to potential hazards resulting from, or associated to, the usage of the software; open-source software, as referred to herein, is provided “as is” – see the Apache (2004) document for an example of the wording of one of the most widespread licenses. These also include the very means by which the terms of the license by the original authors are propagated.

The authorisation details of a typical open source license explicitly permit a window of opportunity for amendments to the original artefact and redistribution of the altered artefact by a third party, as explained above. On the other hand, and in fact, it is common practice – and we consider this practice as a necessary component of our portrait of the open-source distribution and design paradigm – that the terms of the license make one explicit exception to the clause of modifiability: that applies to the substance of the original artefact identifying its authors. Text-based mentions of the author names, and/or the organisation they associate with, must be reproduced integrally and appear at specific locations – see the Apache (2004) document for an example. For the sake of brevity, we may use “open-source software”, or just “open-source”, to designate the design and distribution paradigm as outlined above as well as its outcomes (i.e. individual software artefacts). The combination of the aforementioned components of the definition of open-source has important implications.

3 A paradigm shift and its ripple effects

Let us recapitulate the four open-source propositions of interest in this study, designated by the terms (P1), (P2), (P3) and (P4), and listed as follows: (P1) Obligation of propagation of unaltered original authors’ credentials; (P2) Unrestricted alteration rights; (P3) Unrestricted distribution rights, provided that the source code remains accessible; (P4) Denial of responsibility. These statements apply to the recipient and are bound to some restrictions in order to maintain overall consistency (e.g. (P1) restricts the scope of (P2) and (P3)).

The clause of free modifiability, another description of (P2), can be taken as a first example. Under the open-source paradigm, the clause of free redistribution allows the distribution process to unfold through a dynamically growing network of nodes; following a tree-like pattern in the abstract or mathematical sense. This instils a first increase of complexity in comparison with the conventional, “modern” paradigm of distribution, which, because of the limitations due to the enforcement of proprietary-based licenses, resembles more of a flat, or at least static, structure.

But where the paradigm shift really occurs is by the addition of the clause of free modifiability, an alternate label for (P3). A node may act as a simple replication, or reproduction engine. But it may also break the symmetry of the distribution and replication process by applying a deliberate alteration to the software artefact it has

obtained from its parent node. Consequently, the clause of free modifiability gives rise to a remarkable initial observation: That is, the loss of determinism and predictability of the distribution process. In an alternate definition of the term, Vinge(1993) denoted a singularity as being an instance in time when technological evolution will lead to a massive cultural change that will ultimately alter the perception of reality. In this context, the aforementioned nodes where the software artefact is altered may be viewed as *points of singularity* in the replication process. In other words, there is a profound metamorphosis in the practical reality of the mechanism for all stakeholders. The *raison d'être* for open-source as an ideological design practice is in part due to a conceptual exchange between the constructs of difference and repetition in the creative process ultimately manifesting itself in a dialectic tension similar to that recounted by Deleuze (1994).

Let us examine, as a second example, the implications of the “propagation” clause, a variant tag for (P1). Its effect results in the propagation of original authors’ credentials at all levels of the tree, namely the root of course, nodes and leaves. This forms the underpinning of a second remarkable property: The preservation of a unique, ubiquitous identity throughout the whole tree, i.e. the distribution process. Despite alterations in the reproduction process, all resultant artefacts still bear and share the same identity. A valid representative image taken from biology comes to mind, namely the process of continued operation, replication and functional differentiation of cells of a given living organism, yet bearing the same, intact, unique genetic signature, the same DNA message that was first carved in the original egg. In such a context, the metaphor with biology made in common references to the “life cycle” of software instances is clearly more appropriate in the case of open-source.

In addition, one should not be mistaken by the meaning and effect of (P4), as it bears significant impact with regard to computer ethics. It does not mean that the open-source model intends to shield its adherents from taking a responsible stance in the lifecycle of a given software artefact. Quite the contrary in fact: Because the source code is made available to all recipients, according to (P3), and part/all of those recipients is assumed to be literate and competent, as indicated by (P2), the responsibility becomes unreservedly shared and distributed throughout the whole distribution process and its stakeholders. This marks a radical difference with the modern proprietary model, where the responsibility is restricted in scope and audience, and diminished even further by accepted practices such as the so-called “certification”.

The examples offered above show that the additional degrees of freedom introduced by the combination of the open-source axioms reshape the process of software distribution and induce a significant leap in complexity, in contrast to the typical proprietary model in modern deployment. Such a paradigm shift elevates to an utmost level of significance and relevance any projects revisiting notions such as production, reproduction and differentiation; all of which have been, and still are,

central to postmodernism. More generally, by providing a uniquely innovative perturbation to entrenched principles in a highly self-conscious fashion, the open-source movement meets a criterion that is the universal essence of postmodernism.

4 A collision of design ideals and linguistic constructs

Rather than being viewed as a static artefact that is also a proprietary product, the open source perspective of a computer program can be interpreted as treating software akin to natural languages. None of the latter is owned by anyone in the commercial sense and all are subject to evolutionary change via public collaboration over extended life cycles. Software is written using a programming language and there is a reflexive relationship between both entities. Programming languages by their very structure can be viewed as being linguistic constructs, in actuality highly restricted subsets of the English language.

Who is the author of a mutual dialect? This is a moot point, as language is a dynamic collection of arbitrary signs sharing a common meaning to those fluent in it and is subject to continual transmutation. Text can be viewed as a variant of an idiom and is governed by its inherent rules of syntax and semantics. When one authors a text one is surely inspired by other previously read texts. When one reads a text one’s interpretation of it in effect adds to the text. When one translates a text into another language there occurs a metamorphosis that broadens the original. Thus, text is not an inert entity but a vibrant extension of the language continuum, one that has the potential to span cultures. Derrida (1976: p.158) distilled the essence of this in his enigmatic maxim “*Il n’ y a pas de hors-texte*” (“There is nothing outside of the text.”).

Deconstruction as developed by Derrida is a hallmark of postmodernism. According to this doctrine any text – such as a novel, an essay, a script, a libretto, a poem, a scholarly paper or possibly even software – can be interpreted in almost any manner possible, either completely noncompliant to its original intended meaning, slightly varied or somewhere in-between. In other words, a text cannot be compared to any definitive standard beyond itself. All that is understood, it would seem, is derived from an interlinked and crosshatched tapestry of texts that in effect constitute the sum total of human knowledge. For example, the author Herman Melville wrote the classic novel “Moby Dick” in 1851 but this was actually inspired by stories of past whaling expeditions. In 1938, “Moby Dick” appeared in a musical incarnation in the form of a cantata composed by Bernard Herrmann for male chorus, soloists and orchestra. The film director John Huston crafted a motion picture adaptation of the novel in 1956, a comic book version created by artist Bill Sienkiewicz surfaced in 1990, a television mini-series directed by Franc Roddam was produced in 1998 and so forth.

A similar argument could be proposed with regard to the nature of open source: Should software be considered a textual spectrum with unregulated frontiers? A comment

on the substance of source code – i.e. a text complying with some programming language’s lexical/syntactical convention – will help answering this question. By definition of the compilation or translation procedure, the transformation of a given source code into its executable counterpart is intrinsically deterministic. However, it would be misleading to restrict the essence of source code to its sole operation-purposed end, i.e. functional, machine-translatable semantics.

There is another substance captured in the source code: the know-how of its human author. The current state-of-the-art is such that not all steps of the software engineering process can be fully automated. Some still require the intervention of a human agent and have a dependence on a mix of experiential and theoretical knowledge as recounted by Lyotard (1984). Examples of such crucial steps can be found in “requirements analysis” and “software architecture” activities, which aim at ascertaining that the hypothetical behaviour of the software in gestation meets real-world operational scenarios and contingencies. Despite the availability of powerful tools for software modelling and architecture, the success of such operations dramatically depends on the knowledge of its human authors. Common strategies for knowledge development – i.e. learning – apply. By opening access to source code, the open-source paradigm expands the potential variety of experiences that a given learner may choose to be exposed to; in this case, learning by comparing peers’ practical problem-solving heuristics as embodied within their source code artefacts. This is where open-source meets a key postmodernist condition in that maximal opportunities emerge for participating agents involved in the process (Bauman, 1992). In addition it is deserving of the postmodernist label by preserving a functional habitat where consideration rests on the actions of the human interpreter, as referred to by Fish (1980), as well as simply facilitating the circulation of knowledge in the sense described by Lyotard (1984).

Design in general has been suggested to be a form of bricolage related to postmodernism (Louridas 1999; Lyotard 1984). Levi-Strauss (1966) originated this term in denoting the performance of a creative task based on improvisation and the opportunistic use of limited resources that are subject to constraints. Let us appraise what open source is *not* meant to achieve: Commercial profit enters such category, as nothing in open-source predicaments, nor in their combination, tends to set profit as a driver for any open-source-based effort. In fact, the latter remark may be generalised to almost every instance of any long-term goal.

The careful formulation of open-source axioms conveys the will of their creators to keep the model as free as possible from any form of cynical manipulability. In fact, open-source’s most prominent intents are, arguably: a) preserving the overall model, and b) placing the software author’s motives at the centre of the picture. This means that the potential variety and richness of the open-source model and its outcomes is of same order as those of human individuals. A couple of significant properties are worth highlighting at this point. First, the implicit rules of

the open-source game – once again, a language game similar to that noted by Lyotard (1984) and Wittgenstein (1994) – evoke a permissive sense of restrained chaos suggestive of postmodernist artistic endeavours of the 20th century that subverted the perception of structure. Second, thanks to these unwritten rules, software design in this style promotes self-actualisation in its practitioners through the interlacing of bricolage and aesthetics: In the former, creation is driven by inquisitive experimentation, while in the latter innovation is due to the pursuit of elegance and the satiation of associated hedonic impulses (e.g. as alluded to by Evans & Reddy (2003) in their analysis of motivating factors in open source practice).

5 The aesthetic imperative as an engineering stimulus

Rather than being reviled as an ideology lacking in depth, form and specific purpose, postmodernism could be tentatively embraced by those seeking alternative motivating factors to increase the productivity of software engineers. Philosophy as a discipline evolved as a structured approach to question the reason for essentially everything in comparison to dogma where beliefs are accepted without challenge. Contemporary information systems development is littered with creed-like guidelines for optimal success. To become a business analyst or a developer, especially in the commercial arena, one must conform to the professional tenets of the day. But where is the irrefutable proof of their validity? What are the mechanisms for their legitimation? The radical philosophy of postmodernism is diametrically opposed to canonical acceptance of knowledge. For example, established project management techniques are bastions of modernism often reliant on rigid hierarchical methodologies and an overwhelming emphasis on “focus” as a driving force to ensure completion. Postmodernism, on the other hand, opens the door to a world of play and serendipity in the production of technical artefacts. A creative environment can emerge in which the aesthetic imperative is the dominant source of inspiration rather than monetary concerns. The ensuing apocryphal quote by Aristotle provides a succinct epitaph to this line of reasoning: “*Pleasure in the job puts perfection in the work.*”

6 References

- Apache (2004): Apache Software License. <http://www.apache.org/LICENSE.txt>. Accessed 26 Jan 2004.
- Bauman, Z. (1992): *Intimations of Postmodernity*. London: Routledge.
- Free Software Foundation (2004): <http://www.gnu.org>. Accessed 26 Jan 2004.
- Deleuze, G. (1994): *Difference and Repetition*. (P. Patton trans.) London: The Athlone Press.
- Derrida, J. (1976): *Of Grammatology*. (G. Chakravorty trans.) Baltimore: Johns Hopkins University Press.
- Evans, D. S. & Reddy, B. J. (2003): Government Preferences for Promoting Open-Source Software: A

- Solution in Search of a Problem. 9 Mich. Telecomm. Tech. L. Rev. 313.
<http://www.mttl.org/volnine/evans.pdf>.
Accessed 6 Apr 2004.
- Fish, S. E. (1980): *Is there a text in this class?: The authority of interpretive communities*. Cambridge, MA: Harvard University Press.
- Louridas, P. (1999): Design as bricolage: anthropology meets design thinking. *Design Studies* 20: 517-535.
- Lyotard, J.F. (1984): *The Postmodern Condition: A Report on Knowledge*. (G. Bennington & B. Massumi trans.) Manchester: Manchester University Press.
- Lyotard, J.F. (1992): Answers to the Question: What is the Postmodern?. In *The Postmodern Explained to Children: Correspondence 1982-1985*. (Don Barry et al. trans.) London: Power Institute of Fine Arts.
- Levi-Strauss, C. (1966): *The savage mind*. Chicago, University of Chicago Press.
- Open Source Initiative (2004):
<http://www.opensource.org>. Accessed 26 Jan 2004.
- Raymond, E.S. (2001): *The cathedral and the bazaar: Musings on Linux and Open Source by an accidental revolutionary*. Cambridge, MA: O'Reilly.
- Stallman, R. (2002): *Free software, free software: Selected essays of Richard M. Stallman*. (J. Gay ed.) Boston, MA: GNU Press/Free Software Foundation.
- Vinge, V. (1993): Technological singularity. *Whole Earth Review* Winter 1993, No. 81: 88-95.
- Weinberg, G. M. (1971): *The Psychology of Computer Programming*. New York, NY: Van Nostrand Reinhold.
- Wittgenstein, L. (1994): *Tractatus logico-philosophicus*. (D.F. Pears & B.F. McGuinness trans.) London, Routledge.