A Christian Analysis of Gabriel's "Mob Software"

Thomas VanDrunen
Wheaton College
Thomas.VanDrunen@wheaton.edu

> *"At the risk of sounding blasphemous, Jesus doesn't have much do to with linked data structures."*
>
> *— Student course evaluation comment, in response to "Did the teacher successfully integrate the subject matter with your Christian faith?"*

It is difficult to convince students that integration of the Christian faith in computer science is possible, apart from very general ethical questions. I can sympathize, because I was once unconvinced. My undergraduate education at Calvin College was founded on such ideas as "there is not a square inch in the whole domain of our human existence over which Christ, who is Sovereign over all, does not cry: 'Mine!' " (Kuyper, 1998, pg 488). I could see no clear understanding, however, of how the Christian faith shaped computer science or its cognate activities, such as computer programming.

In October 2000, I attended the conference OOPSLA (Object-Oriented Programming, Languages, Systems, and Applications) where I heard Richard Gabriel give a keynote address, "Mob Software: The Erotic Life of Code," simultaneously published as an essay (Gabriel, 2000). In it, Gabriel proposed a vision for software creation and use where no source code is proprietary and nothing is master-planned. According to the proposal, a mob of programmers should be working on software; this will bring computing to the masses, promote programming as art, and produce a greater diversity of good software. This vision, he claims, goes farther than the "open-source software" movement already in place.

The talk contained a fascinating sweep through such diverse fields as poetry, biology, architecture, and economics, from which Gabriel produced illustrations and support for his critique of current modes of software production and his vision for change. The audience showed its enthusiasm with a standing ovation. I too found many things attractive in his hopes for the field of computer science in general and software development in particular. What was most interesting to me, however, was how Gabriel's understanding of software development had grown from his presuppositions about the world—and that those presuppositions were unbiblical.

Gabriel was integrating *his* faith with computer science. That faith was in the essential chaos in the universe and the accomplishments of collective human effort. The route from his presuppositions to conclusions about software development is long and, in some places, subtle. Nevertheless it is clear that his view of his field is shaped by his starting point. Perhaps an investigation of that route would illuminate how my presuppositions do—or should—shape my own understanding of computer science.

Were the conclusions he drew dangerous for a Christian to accept? Or, by common grace insights, is his proposal or something like it an effort in which a Christian can participate? This paper traces the source of Gabriel's ideas by examining the authorities he cites and how he uses them and evaluates their validity on their own terms. It then considers what basic elements we can identify in a Christian's participation in software development, including what in Gabriel's essay and proposal are useful to a Christian and not in conflict to biblical thinking. Similarly, it identifies what a Christian must deny in Gabriel's proposal. It concludes with suggestions for Christian participation in software development in the world.

General outline of Gabriel's essay. Gabriel's essay criticizes widespread modes of software development. He describes how the software industry, in its greed for profits, has made information on how their software works—the "source code," described later (Section 3.2, page 15)—proprietary. This has prevented the development of a body of software literature, source code to which many people can learn from, modify, and contribute. This, he claims, has led to isolation of software developers (since one interacts only with developers at the same company) and duplicated efforts. It has also taken the development of software out of the hands of those who will use the software. This stifles the training of more programmers, and it has restricted the variety of uses to which computing can be put. Gabriel's problem statement is, "[T]he ways we've created to build software matches less and less well the way that people work effectively" (Gabriel, 2000, pg 1).

Gabriel believes the solution is for software to be developed by what he calls a "mob" of programmers—large numbers of people, drawn from those who will be using the software, all making small contributions, modifying and sharing software from an evolving pool of code, a body of software literature. His thesis is, "The way out of this predicament is this simple: Set up a fairly clear architectural direction [for a new piece of software], produce a decent first cut at some of the functionality, let loose the source code, and then turn it over to a mob" (Gabriel, 2000, pg 3).

I will explain and, in some cases, critique Gabriel's claims by examining four themes in his essay. The first has to do with **order emerging from chaos**. Gabriel quotes from authors who describe how simple rules of local interaction can give rise to group behavior that would be difficult to predict from the rules themselves. "Mob software" results from this emergent behavior of the mob of programmers. Second, Gabriel talks throughout the essay of a "**gift economy**," which he contrasts with a commodity economy. The economic model of gift exchange is necessary for mob software because mob software requires participants to share their code freely. Gabriel also is advocating viewing software more as art than as engineering, and in Gabriel's presentation the gift economy is the proper mode of exchange for art. The third theme also connects software to art, as Gabriel discusses the ***duende***, a personification of sudden inspiration. Gabriel uses this to contrast his vision with the highly-structured approach to programming taught in schools. Finally, Gabriel's essay contains many references to an **earlier time in the history of computing** which looked more like the mob-software approach, and it contains evidence that we are already seeing the work of the mob on the rise.

General context for Gabriel's essay OOPSLA, the conference at which Gabriel gave this talk, is notable in that it has a strong constituency from both academia and industry. The sessions for technical papers are respected academically, but many attendees to the conference do not go to any

of the technical paper sessions, going instead to the practitioner forums, workshops and tutorials, and informal talks. The result is that the conference serves as a meeting ground between academics and professionals. Gabriel plays prominently at OOPSLA; in the years since the talk he has served both as program chair (2005) and conference chair (2007).

One influence for quite a bit of Gabriel's thinking is architect Christopher Alexander. Alexander and his collaborators described what he calls the "timeless way" of building in a series of books (Alexander, 1979; Alexander et al., 1977). The main idea is that the best way to produce good, usable, and "alive" buildings is to have the users of the building involved in the planning and construction process and that the buildings should arise from piecemeal growth. Alexander believes that this is how construction historically has been done, with the modern approach of professionals planning with blueprints being a recent disruption of this timeless way. The integrity of the building and even of neighborhoods and larger communities is preserved because participants in the building process will share a common "language" of patterns, where a pattern is general solution to a common problem that is implemented in many ways. Patterns that appear in Alexander's *A Pattern Language* range from T-Junctions (the roadway should be laid out to avoid four-way intersections) to Alcoves (a large, common room should have small spaces at its edges from semi-privacy) (Alexander et al., 1977).

Whenever I meet architects, I ask them what they think of Alexander's ideas. Invariably, they have not even heard of him. The software development world, however, has imported his ideas enthusiastically. Most notably, the book *Design Patterns* by Gamma et al catalogs a pattern language for common solutions in software design (Gamma et al., 1995). These patterns are analogous to Alexander's architectural patterns in that they are general solutions that are reused in many contexts in the building of software. This pattern-language approach to building software is well-known to OOPSLA attendees; two of the coauthors of *Design Patterns* have each served both as conference chair and program chair, and Christopher Alexander himself gave a keynote talk at OOPSLA in 1996.

1 Theme: Chaos in nature and emergent activity

I want to begin by inspecting sources Gabriel quotes about biology in particular and the natural universe in general; specifically he cites the work of Lewis Thomas, Stewart Kauffman, and Joshua Epstein and Robert Axtell. I believe this most clearly shows his understanding of the world, and we will see how his critique of current modes of software production and his proposal of mob software fit into this.

1.1 Context

Lewis Thomas on cells. Lewis Thomas was a physician who wrote essays relating biological topics to many areas. His book *The Lives of a Cell* (Thomas, 1974) is a collection of essays centered around the observation that the pattern of cellular life (many cells—which in some ways could be viewed as organisms in their own right—together function as an organism) appears in other areas of nature. An example he frequently uses is that of social insects such as ants: an

ant colony acts like an organism, with the individual ants as cells. The essay "On Societies as Organisms," from which Gabriel quotes, says: "A solitary ant, afield, cannot be considered to have much of anything on his mind... It is only when you watch the dense mass of thousands of ants, crowded together around the Hill, blackening the ground, that you begin to see the whole beast, and now you observe it thinking, planning, calculating" (Thomas, 1974, pg 12). In this same essay, Thomas considers the same phenomenon to occur in human societies, such as the scientific community. "We like to think of exploring in science as a lonely, meditative business, and so it is in the first stages, but always, sooner or later, before the enterprise reaches completion, as we explore, we call to each other, communicate, publish, send letters to the editor, present papers, cry out on finding" (Thomas, 1974, pg 15).

Stuart Kauffman on autocatalytic sets. Stuart Kauffman—also a medical doctor by training but more recently a researcher in genetics and complexity theory—takes this idea further. In his book *At Home in the Universe* (Kauffman, 1995), he argues that self-organization emerging out of chaos is part of the fundamental laws of the universe. He proposes understanding many phenomena in the universe in terms of "autocatalytic sets." Kauffman finds examples of autocatalysis in many fields, but the most important (and the field from which the term *autocatalysis* was taken) is chemistry:

> *Many chemical reactions proceed only with great difficulty. Given a long expanse of time, a few molecules of A might combine with molecules of B to make C. But in the presence of a catalyst, another molecule we'll call D, the reaction catches fire and proceeds very much faster. ... Catalysts such as enzymes speed up chemical reactions that might otherwise occur, but only extremely slowly. What I call a collectively autocatalytic system is one in which the molecules speed up the very reactions by which they themselves are formed. (Kauffman, 1995, pg 49)*

The importance of this idea lies in its potential, in Kauffman's view, of explaining mysteries of the origins of life. The chemical system in a cell is an autocatalytic set. "At its heart, a living organism is a system of chemicals that has the capacity to catalyze its own reproduction" (Kauffman, 1995, pg 49). Kauffman cites scientists who calculate that the chances that simple living things (for example, the simplest bacterium of which we know) being assembled from prebiotic material is improbably low, that the age of the universe is too small to account for it. "[T]o duplicate a bacterium... it would be necessary to assemble about 2000 functioning enzymes. The odds against this would be ... 1 in 10^{40000} ... [which is] unthinkably improbable" (Kauffman, 1995, pg 44).

The solution, Kauffman contends, is that these calculations overlook the fundamental likelihood of autocatalysis. These scientists "have failed to appreciate the power of self-organization. ... [T]here are compelling reasons to believe that whenever a collection of chemicals contains enough different kinds of molecules, a metabolism will crystallize from the broth" (Kauffman, 1995, pg 45). Kauffman sees this not only in the origin of life but in the history of evolution. He feels that natural selection alone cannot account for what evolution appears to have accomplished. "[I]f selection, working on random variations, is the sole source of order, then we stand twofold stunned: stunned because the order is so very magnificent; stunned because the order must be so very unexpected, so rare, so precious" (Kauffman, 1995, pg 98). Again, he proposes that self-

organization provides the explanation. "Only those systems that are able to organize themselves spontaneously may be able to evolve further" (Kauffman, 1995, pg 185).

For Kauffman, fundamental laws of self-organization would do more than explain certain parts of paleontology. Kauffman feels that science has removed our traditional sense of value and purpose. "Paradise has been lost, not to sin, but to science. Once... we of the West believed ourselves the chosen of God, made in his image, keeping his word in a creation wrought by his love for us. ... [Now] we are but accidents, we're told. Purpose and value are ours alone to make. ... We bustle, but are no longer at home in the ancient sense" (Kauffman, 1995, pg 4). In its place, Kauffman "hold[s] the hope that what some are calling the new sciences of complexity may help us find anew our place in the universe, that through this new science, we may recover our sense of worth, our sense of the sacred..." (Kauffman, 1995, pg 4–5). Again, "I found myself hoping that large networks of genes would spontaneously exhibit the order necessary for ontogeny. That there was a sacredness, a law—something natural and inevitable about us" (Kauffman, 1995, pg 99).

Epstein and Axtell and artificial agents. Gabriel also makes reference to a set of experiments in what we may call artificial sociology. Complexity researchers Joshua Epstein and Robert Axtell created a virtual world called Sugarscape inhabited by "agents" and containing a resource ("sugar") distributed unevenly. The agents have a visual acuity, a metabolism, and a carrying capacity; based on a set of simple rules, the agents move about the world looking for, storing, and consuming sugar, without which they starve. The researchers observe identifiable patterns of activity among the agents, such as group migrations and competition for resources. Variations on the simulation are made by making each agent a member of one of two tribes, and giving them a preference for proximity to members of the same tribe or inciting the tribes to war on each other; by allowing agents to reproduce (and, with that, adding the sexual transmission of attributes and the bequeathing of stored sugar to the next generation); or introducing a second resource ("spice") and allowing the agents to engage in trade between the two resources. The thrust of the work is a demonstration of complex global behavior emerging from simple rules executed locally. (Epstein and Axtell, 1996)

1.2 Content

In the opening paragraphs, Gabriel says, "I've despaired that the ways we've created to build software matches less and less well the ways that people work effectively" (Gabriel, 2000, pg 1). Moreover, "Every piece of software built requires tremendous attention to detail and endless fiddling to get right" (Gabriel, 2000, pg 3). His solution (and thesis) is that the right way to build large pieces of software is to "set up a fairly clear architectural direction, produce a decent first cut at some of the functionality, let loose the source code, and then turn it over to a mob" (Gabriel, 2000, pg 3). Some of the technical details of what that means will be discussed later, such as in Section 3.2. Our first concern is what Gabriel means by "the mob" and how he postulates that this is the right way.

Gabriel claims, "One of the remarkable discoveries of recent times is that complex behavior by a group of individuals requires only that each individual follow simple rules and the collective behavior of the group [is] nowhere apparent in those rules" (Gabriel, 2000, pg 5). Lewis Thomas's

observations about termites and the agents of Sugarscape demonstrate this. Gabriel concedes that the real-world significance of experiments like Sugarscape is debatable. Its value, he says, is to direct our attention to the boundary between order and chaos. "Between order and chaos, interesting and unexpected combinations come about and last long enough to have repercussions" (Gabriel, 2000, pg 7).

Mob software is based on this principle. Many independent programmers would be working on a piece of software—modifying it, expanding it, and adapting it to their own needs. The software produced would be the aggregate of their efforts. "What I'm talking about is the kind of swarming activity we see exhibited by social insects, a kind of semi-chaotic self-organizing behavior ... Mob artifacts include massive software, built by the multitudes. ... Anyone can add to it" (Gabriel, 2000, pg 18).

On this basis he finds fault in the current modes of producing software in that they are rooted in a desire for order and assume master-planning. "Command-and-control systems are based on the need for control, predictability, and order, things Stuart Kauffman says have no real place in biologically based systems" (Gabriel, 2000, pg 9). Gabriel describes the avionics software for the Space Shuttle, produced by 260 people working over 20 years, as the best the old way of software development has to offer. The software is successful in that it has had a very low occurrence of bugs, but because of design issues it is "an unrealistic model for future systems" and has a cost so high it is justified only in that "we cannot afford to have deaths in the space program" (Gabriel, 2000, pg 21). In contrast, "The mob-software theory is that this project needed 26000 programmers, not 260. The job could have taken less than a year, probably with better quality, and a lot cheaper" (Gabriel, 2000, pg 21).

1.3 Critique

In Gabriel we have an example of worldview affecting programming. He believes there are fundamental laws of self-organization, order arising from chaos, and interesting things happening at the boundary of order and chaos—or at least he finds that the most compelling available theory. His description of "the ways that people work effectively" and the way we ought to be writing software reflects what he believes about the universe. This calls for us to consider the validity of this view of the world and the validity of its connection to software development.

Kauffman and meaning from chaos. Any talk on the theory of origins will arouse passion among Christians. However, regardless of one's opinion on the age of the earth, how much evolution occurred in the world becoming what it is today, or generally how we reconcile real or apparent tension between scientific discoveries and the biblical narrative, there is a need to critique Stuart Kauffman's view of nature. He is explicit about his understanding that science has removed God from giving meaning and purpose to our existence.

It is interesting that Kauffman seems to feel we have lost something important. He expresses sympathy with some prescientific beliefs, that some of them were, in his view, the best explanation of the world from the information available at the time. He freely uses biblical images, speaking about a "real" Garden of Eden (the place in eastern Africa where *homo sapiens* is

supposed to have originated (Kauffman, 1995, pg 3)) and a "real" Noah's ark (a thought experiment about what would happen if the proteins of all living things were allowed to mix in a highly concentrated solution (Kauffman, 1995, pg 113 et seq)). He even employs a concept of God in describing the chaotic aspects of nature too complex for human minds to penetrate. "Only God has the wisdom to understand the final law, the throws of the quantum dice. Only God can foretell the future. We, myopic after 3.45 billion years of design, cannot" (Kauffman, 1995, pg 29). Taken out of context, it sounds like something many Christians (at least those with an "old earth" view) could accept. But Kauffman's work is about a rediscovery of the sacred, and it amounts to a proposal of the laws of self-organization as a new deity. At the very least it is what he proposes to redefine our place and purpose, our being at home in the universe.

Being no natural scientist, I cannot deal fairly with the biology or chemistry, either to critique it or to find what to redeem from it. At least it can be said that Kauffman's work takes the form of a proposal, implying it is unconfirmed. In the subtitle of the book this is called the *search* for laws of self-organization.

More vital is whether this view of the universe—valid or not—appropriately transfers to the production of software. Gabriel's assumption is that a biologically-based system is the appropriate model. At least this is what he has in mind with mob software.

Evaluating emergence. One thing we find in common with Lewis Thomas's ants, Kauffman's autocatalytic sets of proteins, and the agents inhabiting Sugarscape is that they all lack intelligence. Proteins react based on the chemical properties. The movements of Sugarscape agents are calculated based on a set of rules. It might sound hasty to claim that ants have no intelligence, but at least in Thomas's view, "A solitary ant . . . with only a few neurons strung together by fibers,. . . can't be imagined to have a mind at all, much less a thought" (Thomas, 1974, pg 12). With more advanced individuals, we need to take into consideration the influence of the individual. Consider Gabriel's claim, which I believe to be incorrect, that the "simple but beautiful and graceful flocking patterns of birds and schools of fish" (Gabriel, 2000, pg 5) can be generated by three rules:

- Each individual shall steer toward the average position of its neighbors.
- Each individual shall adjust its speed to match its neighbors.
- Each individual shall endeavor to not bump into anything.

I have implemented these rules in my own simulation software (this simulation is available for execution or download at an accompanying website (VanDrunen, 2009)). Like Sugarscape, this software implements a collection of agents inhabiting a grid. Each individual has a speed and direction, and from these the simulation calculates the next grid position into which the individual moves for every cycle of the simulation's running time. Gabriel's rules are a bit vague, but I believe I have interpreted them fairly: Before each move, each individual scans the neighborhood around it and gradually adjusts its speed to the average of the agents near it; likewise it gradually adjusts its direction to what it calculates as the center of mass of the agents near it. Although the rules stipulate that individuals should "endeavor" not to hit each other, the means of avoiding collisions is unspecified. I can think of two interpretations of the rule. If the position the agent calculates

for its next move is already occupied, it simply can skip a turn and not move at all, picking up again where the process left off in the next simulation cycle. A more realistic interpretation, in my opinion, is that the agent should decrease its speed until it calculates a next position that is unoccupied.

In neither of these does a flocking pattern emerge. In the first interpretation, there is a general tendency for the agents to clump together into a few groups of agents, but the motion of the groups in general and the motion of agents within the groups remain chaotic. In the interpretation where the agents slow down to avoid collisions, since agents imitate the speeds of their neighbors, the decrease of speed spreads through the entire system like entropy. Eventually all agents stop moving altogether.

There are several parts of flock and school motion that these rules fail to capture. To begin with, the movements of birds and fish have some sort of purpose—finding food, escaping from a predator, or migrating, for example. (Gabriel generally refers to research that has been done on animations of flocks and schools. Although he does not cite any particular source, there are extended bibliographies of this literature available (Reynolds, 2001). One seminal paper in particular discusses how the rules defining the "steering behaviors" of the individuals are combined to achieve *goals*, such avoiding obstacles or moving toward a location (Reynolds, 1999).) Second, individual birds and fish are capable of independent thought—or if that sounds too anthropomorphic, they are capable of independent response to stimulus. This is related to the problem of the rules lacking a purpose of movement; if one individual sees food or a predator, its sudden change of speed and direction will alert the rest of the group. In other words, the rules given by Gabriel describe only the *conforming* aspect of group behavior. In reality, there is a tension between independent and conforming tendencies, and the flock patterns emerge from the interaction between the two.

Mob activity. On one hand this examination of flocking rules is nit-picking on Gabriel's example, but I think it reflects the difficulty of his applying the principle to mob software. His examples of "mob activity" among people—the making of the Oxford English Dictionary, cathedral-building, and open source software discussed later—all had oversight, master-planning of some sort. They did not emerge merely from agents acting locally under simple rules which would self-organize. The history of the Internet provides another example. In the 1960's and into the early 1970's, many computing research centers were developing computer networks with varied technology and design principles and using different protocols for exchanging information between machines. Some networking sites were similar enough that they could be connected to each other through a single network, the ARPANET. In 1973, various interested parties met to design a new network protocol which would transform the ARPANET into what we now know as the Internet, which would allow very different networks to be connected. Research on the various individual network technologies was something like a swarm, but it required a central effort for any sort of coherence. Janet Abbate says in her history of the Internet, "Establishing a single universal protocol was not the only possible approach to building an internet. One obvious alternative would have been to continue using different host protocols in different networks and create some mechanism for translating between them. . . . [S]uch a design would not scale up gracefully: if the number of networks being connected were to grow large, the translation requirements would become unworkable" (Abbate, 1999, pg 128). These examples differ from a swarm which self-organizes

without central coordination. "The locusts have no king, yet all of them march in rank" (Prov 30:27, ESV). It is hard to imagine that efforts like the OED or the Internet infrastructure would have converged without oversight. (To be fair, Gabriel does not believe mob software projects need to converge (Gabriel, 2000, pg 26).)

Even when swarm or mob activity does converge, there is no guarantee that it will converge to something desirable. The connotations of the word *mob* imply something unruly and destructive. Gabriel limits his use of the word by quoting from the OED, "A multitude or aggregation of persons regarded as not individually important" (Gabriel, 2000, pg 2). Interestingly, this definition has been removed from the OED's Draft Revision 2009 (Oxford English Dictionary, 2009). Moreover, it ignores how the mob can serve as a cover for vice and foolishness. As Kierkegaard said, "[T]here is no place, not even one most disgustingly dedicated to lust and vice, where a human being is more easily corrupted—than in the crowd. . . . [T]he crowd is untruth. It either produces impenitence and irresponsibility or it weakens the individual's sense of responsibility by placing it in a fractional category" (Kierkegaard, 1999a).

Gabriel's boldest claim for the power of mob software is that a mob of volunteers could have written the space shuttle software, in implied analogy with the OED project. "The job could have taken less than a year, probably with better quality, and a lot cheaper" (Gabriel, 2000, pg 21). Perhaps a private spacecraft developer one day will be able to test this theory. For now we can note that the space shuttle software was one of the most complex pieces of software built at the time, produced using an innovative (for its time) development process, in a tight release cycle (17 releases over 31 months), with stringent yet evolving requirements, and many layers of oversight and accountability (Madden and Rone, 1984). The OED volunteers primarily gathered information; the dictionary itself was produced by a professional committee (Oxford University Press, 2009). Their work was hardly comparable to the sort of disciplined testing and verification done by the shuttle software developers.

In summary, the emergence of patterns from agents acting locally under simple rules is a fascinating subject. However, Gabriel's projections from the behavior of social insects to potential achievements of mob software are doubtful.

2 Theme: Gift economy vs commodity economy

The second theme in Gabriel's essay that I identify is concerned with the economic model of software. A contrast is drawn between a gift economy and a commodity economy. The current mode in which software is produced and distributed is a commodity economy. The work of the mob, in Gabriel's view, fosters and requires a gift economy, and it is under this model that software, too, would flourish.

2.1 Context

Gabriel's quotations in "Mob Software" show that on this topic he is primarily informed by Lewis Hyde in his book *The Gift* (Hyde, 1983). It is difficult to say precisely what Lewis Hyde is by training or practice. An Internet search reveals that his academic credentials are in sociology

and comparative literature and that he has held professorships in "Art and Politics" and in Creative Writing (Wikipedia, 2009). In his own words, he has "tried to make [his] way as a poet, a translator, and a sort of 'scholar without institution' " (Hyde, 1983, pg xiii). In the course of the book, he discusses anthropology, folklore, theology, and economics.

The gift economy. In a gift economy, the circulation of goods happens as a gift exchange. This exchanging of gifts creates a bond between the giver and the receiver. The expectation is that the gift will be passed on and thus will circulate. The bonds of gift exchange create and sustain communities and are the basis of interactions between communities. Prosperity is found in the distribution of property, not in its accumulation. In a commodity economy, goods are transferred for profit between autonomous people, forming no bond. "The commodity [is] an alienable object exchanged between two transactors in a state of mutual independence. . . [the gift is] an inalienable thing or person exchanged between two reciprocally dependent transactors" (Morris, 1986, pg 2).

Hyde illustrates this with a rich set of examples, such as stories about tribal societies. "The Indians of the Northwest American coast also give gifts in order to 'make a name' for themselves, to earn prestige.. . . The man who has emptied himself with giving has the highest name. When *we* say that someone has made a name for himself, we think of Onassis or J. P. Morgan or H. L. Hunt, men who got rich. . . [Among the Kwakiutl tribe,] a man makes a name for himself by letting wealth slip through his fingers" (Hyde, 1983, pg 78–79). He finds the concept in the anarchist ritual of destroying records of properties holdings (such as done by the Anabaptists in Münster (Hyde, 1983, pg 87)) and to a certain extent in the scientific community's exchange of ideas (Hyde, 1983, pg 79).

Items or resources distributed or circulated as gifts never become exhausted. This, Hyde says, is "a paradox of gift exchange: when the gift is used, it is not used up. Quite the opposite in fact: the gift that is not used will be lost, while the one that is passed along remains abundant. . . . Gifts are a class of property whose value lies only in their use and which literally cease to exist as gifts if they are not constantly consumed. When gifts are sold, they change their nature as much as water changes when it freezes" (Hyde, 1983, pg 21).

The hoarding of gifts takes away this attribute and puts the receiver at odds with the community. "One man's gift [, the Native American mind-set is,] must not be another man's capital" (Hyde, 1983, pg 4). Quoting Wendy James, "any wealth transferred from one subclan to another, whether animals, grain, or money, is in the nature of a gift, and should be consumed, and not invested for growth. If such transferred wealth is added to the subclan's capital [cattle in this case] and kept for growth and investment, the subclan is regarded as being in an immoral relation of debt to the donors of the original gift" (Hyde, 1983, pg 4). The gift economy works under the assumptions of abundance, the commodity economy under those of scarcity (Hyde, 1983, pg 19).

Art and the gift economy. Hyde's main concern, however, is with art and how the commodification of art (that is, turning it into a commodity) destroys art because art flourishes as a gift. "The more we allow such commodity art to define and control our gifts, the less gifted we will become, as individuals and as a society. The true commerce of art is a gift exchange, and where that commerce can proceed on its own terms we shall be heirs to the fruits of gift exchange: in this case, to a creative spirit whose fertility is not exhausted in use . . . " (Hyde, 1983, pg 158).

In the quote given above, the word "gift" seems to be used equivocally. It can mean object or artifact that is passed on, or it can mean artistic talent. Hyde is self-conscious about the apparent equivocation. "There are several distinct senses of 'gift' that lie behind these ideas, but common to each of them is the notation that a gift is a thing we do not get by our own efforts. ... Thus we rightly speak of 'talent' as a 'gift,' for although a talent can be perfected through an effort of the will, no effort in the world can cause its initial appearance" (Hyde, 1983, pg xi–xii). Moreover, the divine (or at least supernatural) is often assumed the originator of or is otherwise involved in the cycle of gift-giving. Hyde finds examples of this both in tribal religions (Hyde, 1983, pg 26 et seq) and the Bible (Hyde, 1983, pg 19–20).

Hyde contrasts the spirits of gift and commodity economies with the Greek words *eros* and *logos*. In his mind, *eros* stand for imagination, *logos* for logic; *eros* for synthesis, *logos* for analysis or dialectic; *eros* for bonding (including the bonding of people in a relationship), *logos* for differentiating into parts. In this dichotomy, with *eros* we have gift exchange and with *logos* market exchange. "If we say, as Marx suggested, that 'logic is the money of the mind,' then we might add as a corollary that the imagination is its gift" (Hyde, 1983, pg 155). (The subtitle of *The Gift* is "Imagination and the Erotic Life of Property." Although a provocative title, the content shows that Hyde is not using *eros* in a specifically sexual sense.)

In addition to Hyde, Gabriel's understanding of the gift economy is informed by a paper by classicist and ancient historian Ian Morris, "Gift and Commodity in Archaic Greece" (Morris, 1986). He gives evidence that the gift economy and commodity economy coexisted for centuries in ancient Greece as it transitioned from a clan society to one of state and empire. Hyde, in fact, concedes at the end of his book that a similar coexistence is realistic. "My position has changed somewhat. I still believe that the primary commerce of art is a gift exchange... But I no longer feel the poles of this dichotomy to be so strongly opposed. ... [G]ift exchange and the market need not be wholly separate spheres" (Hyde, 1983, pg 273). (In 2008, Hyde was at least comfortable enough with the commercial world to write an article updating *The Gift* appearing in a magazine section of the Wall Street Journal which contained mostly high-end advertising (Hyde, 2008).)

2.2 Content

In Gabriel's description of mob software, one of its primary differences from other modes is its economic aspect. Mob software uses and requires a gift exchange, whereas our current models of software development are rooted in the commodification of software. In Section 4, I will analyze in more detail Gabriel's claims about how the current commodification came about and what its effects are. In brief, it stopped software from flourishing, meaning both that there grew no respect for it (particularly as an art form) and that the variety of uses of computing was stifled.

This is related to the commodity economy's arising from scarcity. "The history of computing is rife with resource limitations," primarily insufficient computer speed and memory, and the undermanning of software development projects (Gabriel, 2000, pg 13–14). Software knowledge as a commodity is something proprietary, something not shared by a community, and this situation forces people to work in isolation. "Because of this we've managed to build only very small pieces of software, and even those are horribly error-ridden. ... [B]ecause development has been done primarily in a landscape riddled by many isolated islands of similar or identical activities, we have,

over the last 40–50 years, written the same code over and over. . . . Are Word, Excel, and Powerpoint really the best possible programs in those categories?" (Gabriel, 2000, pg 14). Certainly proprietary code is shared property among those working in a corporate development team, but it is not common to the larger community of software developers and users. More sharing would, in Gabriel's view, allow more experimentation and diversification, and thus true alternatives to dominant products like Microsoft's Office suite.

In Gabriel's view, scarcity is not only a condition which, when it happens to occur, encourages commodification. Scarcity like this is also something that those profiting in a commodity economy have an incentive to maintain. "[T]he commodity economy. . . depends on scarcity. Its most famous law is that of 'diminishing returns,' whose working requires a fixed supply. Scarcity of material or scarcity of competitors make high profit margins" (Gabriel, 2000, pg 23). "When computers were commoditized, the resource limitations inherent in software development became an opportunity for exploitation, and any relief to those limitations meant less wealth to go around. Draw your own conclusions" (Gabriel, 2000, pg 14).

One of the most important resources in software development is skilled programmers. Gabriel mentions a question posed by the open source community (a movement I will describe in Section 3.2 and which Gabriel believes shows some aspects of mob software but does not go far enough): "What if what once was scarce is now abundant?" (Gabriel, 2000, pg 17) The meaning is, what if there were more people with significant skill in developing software? How would that change how software is developed and distributed and to what uses software is put? In Gabriel's view, mob software and this sort of abundance would mutually reinforce each other. In describing a future characterized by mob software, Gabriel says, "Mentoring circles and other forms of workshop are the mainstay of software development education. There are hundreds of millions of programmers" (Gabriel, 2000, pg 28) and "Resource scarceness created by artificial boundaries no longer exist, and in an era of abundance, excess thrives" (Gabriel, 2000, pg 29).

2.3 Critique

Gabriel's unspoken premise is that it is reasonable to think of software as art, since it is to the "commerce of the creative spirit" that Lewis Hyde applies the gift economy. To be fair, Gabriel's audience may know of this premise, since Gabriel has talked about it elsewhere, for example his proposal for a Master of Fine Arts in Software degree (Gabriel, 2003). I do not find that way of thinking about software unreasonable, but it certainly is incomplete. The places where the analogy breaks down call into question the strength of Gabriel's argument that the commodification of software is behind (and necessarily so) Gabriel's opening claim that "the full expanse of what computing could do—to enhance human life, to foster our creativity and mental and physical comfort, to liberate us from isolation from knowledge, art, literature, and human contact—is left out of our vision" (Gabriel, 2000, pg 1).

A concrete version of Hyde's understanding of commodification ruining art is found in Hyde's opening example: a drugstore romance novel series intentionally formulated to fit the interests of customers found by market research (Hyde, 1983, pg xi). Hyde rightly expects his readers to agree that books produced this way will never be high literature, and he sees this as degrading to the novel as an art form. It is not clear, however, that the commodification of software is analogous.

One would rather expect that market research would be useful in designing software intended for a general audience.

Some of Gabriel's examples of a gift economy are suspect. After quoting from Ian Morris, Gabriel says, "In a typical gift exchange situation, a patron would provide food and shelter to a poet in exchange for poems about the patron" (Gabriel, 2000, pg 22). The context implies that ancient Greece is the setting for this example. However, Morris never uses artistic patronage as an example of gift exchange. His paper has in view gift-giving between chiefs for making and preserving alliances. Elsewhere Gabriel says, "A healthy Western family operates on a gift economy" (Gabriel, 2000, pg 23). Certainly one cannot imagine a family impersonally exchanging commodities, but that does not mean a gift economy is an appropriate model either. For example, in the gift economies described by Hyde, it is unethical to turn a gift into capital. In my experience, parents help their adult children through interest-free loans and similar measures for the very purpose of allowing them to build capital (the proverbial "nest egg") or to buy a house or pay off debts.

I believe Gabriel's application of the gift economy to software development is of some use in describing the workings of the open source and free software community—described later—but does not prove that its absence in the software industry has inhibited the production of good software.

3 The *duende*

One of the most mysterious aspects of Gabriel's essay is his discussion about the need for programmers to have duende or to battle with the duende. He borrows this duende concept from theories about certain kinds of Spanish folk music and art. The Spanish poet Federico García Lorca spoke and wrote about the *duende* as a source of artistic inspiration. Gabriel borrows this concept to exhort software developers to more ambitious programming.

3.1 Context

The concept of duende is difficult to pin down. It is a personification of a kind of spirit of artistic spontaneity. Etymologically, it comes from *duen de casa*, "master of the house." Shortened to *duende*, the term refers to various kinds of genies or elves in Spanish and Latin American folklore. Originally it seems to have been a sort of playful sprite that would mysteriously break or hide household objects (Maurer, 1998, pg ix). It seems that in the confluence of musical and other artistic styles found in the region of Andalusia (Roma and Moorish influences play strongly there), duende took on the meaning of a spirit that would creep into an artist during a performance.

Christopher Maurer describes García Lorca's duende as "irrationality, earthiness, a heightened awareness of death, and a dash of the diabolical. The duende is a demonic earth spirit who helps the artist see the limitations of intelligence" (Maurer, 1998, pg ix). To understand what is meant with terms like "diabolical" and "demonic," García Lorca seems to want to distinguish the duende from the devil as understood in the Christian tradition. "I do not want anyone to confuse the duende with the theological demon of doubt at whom Luther... hurled a pot of ink..., nor with

the destructive and rather stupid Catholic devil…" (García Lorca, 1998, pg 49–50). I take what he says to mean that the duende is something desirable, not just something that haunts; it teases rather than oppresses, and it is not something we need to mock in order to calm our fears.

In describing the duende, García Lorca contrasts it with other conceptions of inspiration that have been personified by supernatural beings, in particular angels and muses. Angels, in García Lorca's view, guide, defend, and forewarn (or perhaps evangelize); he associates these three functions with Raphael, Michael, and (the angel) Gabriel respectively. What characterizes angels is that they are "*ordering*, and it is useless to resist their lights, for they beat their steel wings in an atmosphere of predestination" (García Lorca, 1998, pg 50). The muse is an intellectual spirit, "but intelligence is often the enemy of poetry, because it limits too much…" (García Lorca, 1998, pg 51). Another difference is that "[T]he muse and angel come from outside us…but one must awaken the duende in the remotest mansions of the blood" (García Lorca, 1998, pg 50). (Interestingly enough, García Lorca finds the duende in some aspects of the Christian tradition: he describes Teresa of Ávila as being full of the duende (García Lorca, 1998, pg 58).)

The duende is contrasted with technical skill. García Lorca quotes one artist critiquing another with "You have a voice, you know the styles, but you will never triumph, because you have no duende" (García Lorca, 1998, pg 48). Again, García Lorca says,"[I]t is not a question of ability but of true, living style, of blood, of the most ancient culture, of spontaneous creation" (García Lorca, 1998, pg 49). In fact, the duende opposes the orderly, disciplined approach one has been taught. "[He] rejects all the sweet geometry we have learned, …he smashes styles…" (García Lorca, 1998, pg 51).

The artist has a battle with the duende, and the battle involves risks. The artist must invite the duende to battle by taking the ultimate risk. "The duende does not come at all unless he sees that death is possible. The duende must know beforehand that he can serenade death's house…" (García Lorca, 1998, pg 58). It is not immediately clear what García Lorca means by "the possibility of death." His own illustrations sound cartoonish: Without the duende, the poet "forgets that ants could eat him or that a great arsenic lobster could fall suddenly on his head" (García Lorca, 1998, pg 51). Gabriel gives a corroborating quote from Christopher Alexander by way of Stephen Grabow, "All the Japanese arts recognize the fact that, finally, you have to meet the fear of death in order to do anything—landscape painting, flower arrangements, and so on" (Grabow, 1983, pg 86). In context, Alexander's comparison is between being a swordsmaster and a flower arranger, and it sounds anticlimactic to the point of humor. However, Alexander proceeds to put things more plainly,

> *[I]f you take the fear of humiliation…and you try to trace it, you realize that you have a whole series of linkages in your mind which ultimately go back to the fear of death. For example, if you are mocked you may lose your job, and if you lose your job perhaps you will end up in the gutter… (Grabow, 1983, pg 86)*

Perhaps this sort of fear is what García Lorca has in mind and what the duende is conceived as coming to challenge. That is at least how Gabriel interprets it (Gabriel, 2000, pg 8).

3.2 Content

The economy of gift exchange requires a source for the gift. In Lewis Hyde's understanding of the commerce of the creative spirit, the source is artistic talent, which Gabriel connects to the duende of García Lorca. This seems consistent with what Hyde says. He even refers to the duende at one point where he describes how those marginalized from a commodity economy are better suited for gift exchange:

> *A commodity is truly "used up" when it is sold because nothing about the exchange assures its return. . . . Gifts that remain gifts can support an affluence of satisfaction, even without numerical abundance. The mythology of the rich in the overproducing nations that the poor are in on some secret about satisfaction—black "soul," gypsy duende. . . —[has] a basis, for people who live in voluntary poverty or are not capital-intensive do have more ready access to erotic forms of exchange that are neither exhausting nor exhaustible and whose use assures their plenty. (Hyde, 1983, pg 23).*

Gabriel introduces the duende to put a face on this source. "When you do battle with the duende . . . you might find a gift, a gift of talent or insight. . . " (Gabriel, 2000, pg 9). Near the beginning of his essay, he hints at the duende as a necessary part of coming out of the software development islands where Gabriel sees the commodity economy has put us: "The way out requires just one thing from us—a strange, frightful thing—something slant. It is this: Find a way to fight our fear of death" (Gabriel, 2000, pg 1). Participating in mob software takes on the risk of failure and of scorn and rebuke from our colleagues.

The duende fits nicely into Gabriel's understanding of order arising from chaos as a fundamental law of the universe. "Duende, poetry, and perhaps life itself and life in our works of artifice are denizens of the boundary between order and chaos" (Gabriel, 2000, pg 9).

From this Gabriel critiques current modes of producing software, which spring from a cowardly need for order. "Early computing practices evolved under the assumption that the only uses for computers were military, scientific, and engineering computation—along with a small need for building tools to support such activities" (Gabriel, 2000, pg 11). It was engineering and science types, as opposed to, for example, artists, who defined how software production was done and understood.

To understand what Gabriel is getting at in this context we must know something about the languages used for composing software. The most basic, "assembly" or "machine" languages, are directly interpreted by the computer hardware. Each kind of computer (say, each microchip model from each chip manufacturer) has its own such language, but they all consist in simple instructions for moving small chunks of data (say, one number or one alphanumeric character) between memory locations or performing one arithmetic or logical operation. The instructions are laid out in a linear fashion and executed one after the other except where broken up by instructions to jump to a different position in the instruction sequence. "Higher level" programming languages were invented (in part) in light of the difficulty of programming in assembly language. Fortran was one of the earliest languages. It was designed for use by scientific and engineering concerns, and while it still conceived a program as a sequence of instructions, it allowed programmers to

describe computational processes more succinctly; a small number of lines of Fortran could then be translated into a large number of assembly language instructions. The programming language C also conceives programs in essentially the same way as assembly language, but it came a bit later than Fortran and was designed for the purpose of writing operating systems and other basic tools for using the computer. Thus Fortran stands for Gabriel's "military, scientific, and engineering computation" and C for the "small need for building tools to support such activities."

These assumptions and the orderliness they forced upon computing stifled more daring exploration of computing. "The very architecture of almost every computer today is designed to optimize the performance of Fortran programs and its operating-system-level sister, C. Further, . . . nonstandard computational models were soundly rejected, snubbed, and even ridiculed" (Gabriel, 2000, pg 11). Along with these languages we have the rise of "software methodologies," the disciplined, orderly way of designing, implementing, documenting, and testing software that one might have learned in school. It came about as an attempt to mimic successful engineering practices (like those found in the space program). "Software development methodologies evolved under this regime along with a mythical belief in master planning. . . . Master planning feeds off the desire for order, a desire born of our fear of failure, our fear of death" (Gabriel, 2000, pg 11).

Software is more like poetry, in Gabriel's view. We cannot emulate NASA and expect to get good software. Instead we must face our fear of failure and battle with the duende. This leads into Gabriel's cynicism towards programming education. If excellence in programming is based more on the presence of the duende than in technical skill, then the effectiveness of software education is limited. "Software skill is a gift from an unknown and unknowable source—perhaps the skill is an artistic talent, perhaps there are parts of the process of writing software that can be taught" (Gabriel, 2000, pg 30). I will take up Gabriel's view of computer science education in more detail in Section 4.

3.3 Critique

Gabriel's use of Kauffman's understanding of the world seems inherently naturalistic, even atheistic. Yet in Kauffman we see a desire to reinvent the sacred. The duende appears to be a fitting god for the world of chaos and order. Gabriel's essay is really about how the programmer can worship it. "When you do battle with the duende. . . you might find a gift, a gift of talent or insight that will make you think, 'Did I say that?'—'Did I do that?' Gifts like this are worth cultivating, even in the software world" (Gabriel, 2000, pg 9).

Considered on its own terms, Gabriel's challenge for the software world to loosen its clinging to things that enforce order and to battle the duende sounds unrealistic. García Lorca's description of the duende contains examples of people whose battle with the duende prompts amazing performances despite a lack of technical skill. "Years ago, an eighty-year-old woman won first prize at a dance contest. . . She was competing against beautiful women and young girls with waists supple as water, but all she did was raise her arms, throw back her head, and stamp her foot on the floor. In that gathering of muses and angels. . . who could have won but her moribund duende" (García Lorca, 1998, pg 54). To apply a duende-like idea to the composition of software, however, presupposes a high level of technical skill to be already present in the programmer. A computer program is not like a poem or a dance in this way; if the programmer is not able to pro-

duce something parsable in the programming language or cannot fit the instructions together in a logical way, the program simply will not work.

Perhaps Gabriel considers the prerequisite of technical skill to be part of the problem, and that it does not need to be this way. It might be that our programming languages and related tools—so order-filled, not inviting to the duende, and designed with engineers in mind—keep the masses away. "The current situation might feel fine to some of you, but suppose all computing were based on the needs of tightrope walkers? Hard to imagine? What we've created was hard for them to imagine" (Gabriel, 2000, pg 16–17). Gabriel asserts that the current situation is hard for the masses to imagine, but does not help us in visualizing the alternative.

Gabriel claims that fear of failure and death prevented the flourishing of approaches to software that departed from the assembly language/Fortran/C model. He cites the programming languages Lisp, Simula, Smalltalk, and Prolog as different ways to think about computation and that were rebuked. The claim that these languages and the models they stand for were rejected or that this rejection was grounded in fear of failure is one interpretation of history, but not the only one. Gabriel's own experience may color his perception. He founded a software company that produced programs for Lisp development and which went bankrupt after 10 years. (That company did experience some successes, though, and some of its products remain, including the software I am using to write this paper.) He also has spoken about why C and related languages are more widely used than those in the Lisp family (Gabriel, 1989). In another interpretation of history, however, Lisp and its derivative languages were very influential in the development of computer science concepts and maintain a loyal following and wide respect. More can be said about Simula and Smalltalk. They represent the earliest attempts at the "object-oriented" approach to programming, and while that approach took some time to catch on and those two languages in particular did not become dominant, one must say that object-oriented programming has won in the long run. Object-oriented languages which are the conceptual descendants of Simula and Smalltalk are almost universally taught in introductory programming courses. Moreover, if Gabriel means to suggest that these programming languages or models could have made programming more accessible to the masses lacking technical skill, it is quite a dubious claim, either that they could have done so if given the opportunity or even that they lacked the opportunity. For example, the Logo programming language, a Lisp derivative, was used to introduce computing to elementary school children in the 1970's.

As final note on Gabriel's use of the idea of duende, it might be that duende in fact conflicts with Gabriel's understanding of the mob. García Lorca's development of the duende concept was part of his exploration of the nature of the *cante jondo* music genre, and came about when in his mind, "cante jondo no longer seemed a 'collective' creation, and the singer no longer seemed a passive 'medium' for the 'voice of the people.' ... [The elements of cante jono] depended upon the 'personality' of an individual performer, and upon his or her search for the spirit known as duende" (Maurer, 1998, pg viii). Mob software is produced by an aggregate effort of programmers who are "not individually important" (Gabriel, 2000, pg 2).

4 Theme: The good old days are gone, but they're coming back

Gabriel's opening line is "I've got good news: That way of hacking you like is going to come back in style" (Gabriel, 2000, pg 1). His regret at the demise of an earlier age of software development—an age presumably friendlier to the mob, more consistent with a gift economy, and less afraid of the duende—pervades the essay, but so does his optimism that a similar movement is underway. The history of computing is naturally the primary reference point for this claim, but parallels are found in other human endeavors.

4.1 Context

Gabriel quotes from Steven Levy's *Hackers*, a popular and selective history of "the computer revolution" (Levy, 1994). As a chronicle, it revolves around two loci of early computing activity: the Massachusetts Institute of Technology in the 1950's and 60's, and the San Francisco Bay area in the 60's and 70's. What makes the story interesting is the characters found in the these communities of "hackers" who were relentless in their seeking access to computers and who, having access, neglected other areas of life to engage in the joy of hacking.

Hacking. A few things should be said about the terms *hacker* and *hacking*. In the popular media and other non-technical usage, the verb "to hack" means to gain illicit access to computers, such as to read sensitive information, to take control of a system, or to use one computer as staging ground to hack into another computer, so as to make one's steps harder to trace. As examples of this usage, in April 2009 the news media carried stories about the infiltration of the American electricity grid and military computer systems by Chinese spies. In coverage of these stories, many mainstream outlets, including the Wall Street Journal, the New York Daily News, the Boston Globe, the BBC, and The Guardian, contained headlines labeling the spies as "hackers" or their activity as "hacking."

Levy bemoans this in the Afterword of his book:

> *The term "hacker" has always been bedeviled by discussion. When I was writing [the first edition of] this book, the term was still fairly obscure. . . . Unfortunately for many true hackers, however, the popularization of the term was a disaster. . . . [T]he word quickly became synonymous with "digital trespasser." (Levy, 1994, pg 432)*

This is not at all what is meant by the term by many in the computing community. Instead, hacking refers to a style of free-spirited programming done by computer wizards, contrasted with disciplined programming methodologies taught in university classes and generally practiced in industry. The definition of "hacker" in *The Hacker's Dictionary* includes

> *1. A person who enjoys exploring the details of programmable systems and how to stretch their capabilities . . .*
> *2. One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming. . . .*
> *3. A person who is good at programming quickly. . . . (Hacker, 1996)*

In the programming courses I teach, the term even in this sense can have positive and negative connotations. I may rebuke a student that his code is a "hack," meaning it uses an ad hoc solution, something that will not generalize to solve wider problems and will not be easily understood and recognized by other programmers. On the other hand, a clever programming technique that accomplishes something in a creative way under constrained resources might be praised as a "hack."

The Hacker Ethic. The communities Levy describes share a set of values that he calls the Hacker Ethic, summarized as

- Access to computers—and anything which might teach you something about the way the world works—should be unlimited and total.
- All information should be free.
- Mistrust authority—promote decentralization.
- Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race, or position.
- You can create art and beauty on a computer.
- Computers can change your life for the better (Levy, 1994, pg 40–45).

In Levy's story, The Hacker Ethic was used to justify a good deal of privilege and license. The original hackers—mostly MIT undergraduates and dropouts who were employed by the Artificial Intelligence Lab—felt free to look at any lab user's programs, borrow anyone's tools, and enter rooms to which they did not have permission. On the other hand, it represented a certain sense of responsibility. Anyone who produced a clever hack owed it to the world to leave the tape containing the code in the lab desk drawer so others could study it.

This put the hackers at odds with the software industry as it developed a life independent of the computer hardware industry. In 1975, a young Bill Gates complained in an open letter about "hobbyists" who had obtained unpaid-for copies of his implementation of BASIC (a programming language) for the Altair (an early home computer) (Levy, 1994, pg 229). This thinking ran counter to the computer revolution in the minds of hackers, who reasoned that software was information and information should be free. The problem was not so much with the selling of software but with restrictions that were put on the purchaser, who, in their view, should be able to study, modify, share, and even resell it.

A modern expression of similar attitudes towards software is made by what is alternately called the "open-source" or "free" software movement. Gabriel does not cite any description of this movement, but it is well-known to his audience. As an example, the Free Software Foundation defines four freedoms it believes should apply toward software.

- The freedom to run the program, for any purpose (freedom 0).

- The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this. (Free Software Foundation, 2007)

"Source code" refers to the original text of a program written by human programmers. In order for the program to be executed by the computer, it must either be compiled (that is, translated) into the machine language of hardware instructions or be interpreted by another program. Commercial software is usually distributed only in its compiled form. Since the machine language is not easily human-readable and few programmers are thoroughly trained in it, this effectively makes it impossible for the consumer to study or modify the software.

The terms "open-source software" and "free software" are often used interchangeably, although the Free Software Foundation adamantly distinguishes between merely "open-source software" (meaning the source code is made available) and "free software" defined by the freedoms listed above. Many people misunderstand the term "free software," however. While much of it is available for download at no cost, there are also companies that sell it and charge for customer support. The software is "free" in the sense that the user may do anything with it (including modifying and redistributing it), not in the sense that it is necessarily given without price. We will avoid the terminology debate by referring to such software as open-source/free software (OS/FS). The most visible OS/FS product is the Linux operating system kernel, but there also exists OS/FS for almost every software category.

The old way of building. The ideas of Christopher Alexander are relevant here as well. We shall see that Gabriel argues that the work of the mob is nothing new. Alexander believes this about his philosophy of architecture. The very title of his book indicates that he believes his way of building is "timeless." He says

> *This one way of building has always existed. It is behind the building of traditional villages in Africa, India, and Japan. It was behind the building of the great religious buildings: the mosques of Islam, the monasteries of the middle ages, and the temples of Japan. It was behind the building of the simple benches, and the cloisters and arcades of English country towns; of the mountain huts of Norway and Austria; the roof tiles on the walls of castles and palaces, the bridges of the Italian middle ages; the cathedral of Pisa. (Alexander, 1979, pg 10–11)*

Like Gabriel (as we shall see), Alexander sees the breakdown of the timeless way of building as a recent trend induced by industrialization and the removal of the users of a building from the process of building:

> *[I]n the early phases of industrial society which we have experienced recently, the pattern languages die. Instead of being widely shared, the pattern languages which*

determine how a town gets made becomes specialized and private. Roads are built by highway engineers; buildings by architects; parks by planners . . .

The people of the town themselves know hardly any of the languages which these specialists use. . . The languages start out by being specialized, and hidden from the people; and then within the specialties, the languages become more private still, and hidden from one another, and fragments. (Alexander, 1979, pg 231–232)

4.2 Content

Gabriel claims that the mob software approach or something like it was captured by several early communities in the computing world. Two paragraph-length quotes from *Hackers* describe how members of the Homebrew Computer Club were continually sharing and improving on each other's discoveries of what they could do with their home machines, a "textbook example of" synergy (Gabriel, 2000, pg 4); and how at MIT the ITS operating system was produced incrementally, implemented by the very people who were and would be using it.

Drawing apparently from his own experience, Gabriel describes a golden age of sharing source code in the research world in the 1960s and '70s, particularly in the orbit of the programming language Lisp and the field of artificial intelligence. This appears to be a foundation for the body of software literature he elsewhere states is badly needed. The rise of the software industry, however, killed the nascent mob software approach. "In the 1980s, it became apparent there was money to be made by writing and selling software, and that a premium might be had for being clever about it.. . . At that point [with few exceptions], software source code essentially disappeared" (Gabriel, 2000, pg 19)

As mentioned above, the mob aspect of computing was smothered by the rise of an industry that was short-sighted in greed for profit, which Gabriel characterizes as "high-octane capitalism" (Gabriel, 2000, pg 1) and "fast-lane capitalism" (Gabriel, 2000, pg 17). The Homebrew hobbyists interacted through a gift economy, just like the OED volunteers. This gift economy of the mob was replaced with a commodity economy with several negative results, starting with the loss of the body of literature. "The effect of ownership imperatives [of source code] has caused there to be no body of software as literature. It is as if all writers had their own private companies and only people in the Melville company could read *Moby Dick* and only those in Hemingway's could read *The Sun Also Rises*" (Gabriel, 2000, pg 17)

Along these lines, no conception of programming as an art could develop. "If all remnants of literature disappeared, you'd expect that eventually all respect for it. . . would disappear. And so we've seen with software" (Gabriel, 2000, pg 17). In this context, the teaching of programming has failed. "We find little or no code education. . . . [I]f the student becomes a proficient programmer, it is due to luck and hard work, in that order" (Gabriel, 2000, pg17).

Some of the other aspects opposed to mob software—either suppressing it or resulting from its absence—seem, in Gabriel's view, to have existed earlier in the computing community (though the implication is that the rise of software as commodity has prevented the correction of these things). "Early computing practices evolved under the assumption that the only uses for computers were military, scientific, and engineering computation" (Gabriel, 2000, pg 11). Rather than

encouraging a mob making incremental improvements to evolving software, the early practices had in mind a proverbial sole physicist whose approach to programming was "figure it out, code it up, compile it, run it, throw it away" (Gabriel, 2000, pg 13). In other words, software necessarily must be master-planned, not approached with the disorder of the mob or duende. "Master planning feeds off the desire for order, a desire born of our fear of failure, our fear of death" (Gabriel, 2000, pg 11).

The early assumptions were that computing would be interesting only to engineers and such people. These assumptions are proven false by the prevalence of computing we find today, but they prevented programming itself from getting into the hands of the masses. "Today almost every business and human pursuit is built on computing and digital technology. Artists... tightrope walkers... carpenters... dentists... depend on computing, and most of the people I mentioned want to have a say in how such software works, looks, and behaves. Many of them would program if it were possible" (Gabriel, 2000, pg 16). Like Christopher Alexander's wish that the design and construction of buildings be put into the hands of those living in and using them, so part of the vision of mob software is the putting of programming into the hands of the users. Just as homes are customizable—home owners can put in new shelves and repaint or, if they are more ambitious, perform a serious remodelling—so OS/FS programs can be tailored to the needs and preferences of the users by the users themselves. The ITS operating system mentioned above is an example.

Nevertheless, Gabriel sees mob software beginning to take off. The first bit of evidence is the OS/FS movement, particularly in that it has produced high-quality and successful products. "Open source... provides our first view into mob software... By now we know about quite a few mostly successful open-source—or baby mob software—projects: Linux—which has won a number of awards for customer support and artistry—Mozilla, Jikes, Emacs, GCC, Apache, Perl, Python, sendmail, and BSD" (Gabriel, 2000, pg 18). The list includes two operating systems, a web server, an email server, a web browser, a text editor, and four programming language systems.

Gabriel is very quick to point out, however, that the free software movement does not go far enough. "Unfortunately, the open-source community is extremely conservative, focusing solely on the need to build up slowly a parallel open infrastructure next to the proprietary ones already in place" (Gabriel, 2000, pg 25–26). True mob software would be willing to explore a more varied and creative realm of software and would be less concerned about maintaining stable versions of the code. "Open-source projects tend to be convergent: Each project is aimed at a particular artifact in the end. Mob-software projects tend to be divergent: The goal is to build variety and diversity so that the competitive forces and natural selection in the user realm can take over" (Gabriel, 2000, pg 26). Implied but unstated is that only engineers and the like, unfortunately, are involved in OS/FS work.

Perhaps a better example of the work of the mob in computing technology, in Gabriel's view, is the variety of uses (and users) of the World Wide Web. "The World Wide Web was first envisioned as a way to publish and cross-reference scientific papers. ... But the technology was accessible to real people, and the Web grew almost boundlessly" (Gabriel, 2000, pg 29). The Web is also a medium through which Gabriel sees a gift economy to be viable alongside the commodity economy.

As a final (non-computing) example, Gabriel describes Levittowns, planned communities built by the development company of Levitt and Sons after World War II. These homogeneous,

mass-produced homes represented the opposite of Christopher Alexander's timeless way. "I might have argued that . . . Levittown metaphorically epitomized what I find frightening and discouraging about contemporary software design and construction" (Gabriel, 2000, pg 31). Instead, the mob living in first Levittown personalized their homes so thoroughly that their uniformity is no longer easy to recognize. Gabriel has a similar hope for the mob to change the landscape of software.

4.3 Critique

Hackers and how we program. I find what Gabriel says about the "physicist's model" of programming credible in my own experience. As an undergraduate, I was told that when approaching a large programming task we were to "design, design, design *first*." A recent graduate whom the department had hired to develop some software for them had spent five months designing the software and one month coding it. "Figure it out, code it up, compile it, run it, throw it away," as Gabriel puts it (Gabriel, 2000, pg 13). Eagerly coding away before figuring it out, my professors would warn, would mean making design mistakes I would pay for later.

In graduate school I found myself not following their advice. When I reflected on what I was doing, it seemed I was taking a "code first" approach. The process would be code, debug, understand, document. It bothered me a little. I felt like I simply did not have the discipline to design first and to test and document as I went along as I was taught to do. I once had a conversation with another graduate student, an acquaintance of mine, who happened to be a physicist. He was doing some coding as part of his research or a class he was taking, and mentioned studying the algorithms he would be implementing. Do you always need to take so much time studying other people's algorithms? I asked. "I don't feel comfortable starting to code until I understand completely what the algorithm is doing," he replied. Would you ever attempt to learn an algorithm *by* programming it? "Recipe for disaster," he said.

My thoughts pulled me in two directions. Part of me thought, here is a man who is disciplined enough to program the right way while I am just making trouble for myself. On the other hand, I was not just implementing algorithms I read in a book. As a researcher in computer science itself, not just a field that used computing, I was inventing my own algorithms. I had tried the "design first" (not quite the "design, design, design first") approach, and it did not work—I truly could not anticipate the problems the design would need to address until I actually attempted to code up a solution. But even algorithms I read in journals were simply too complicated to understand just by reading and thinking. My practice was to learn algorithms by coding them. Perhaps it was because I was a computer scientist and code was my everyday currency of thought. Or perhaps Gabriel was right (if this is what he meant): when you are doing something new and ambitious and when you are really inspired, you just want to hack.

Software and architecture. Much of Gabriel's argumentation in this theme depends on an analogy between software and architecture. The use of this analogy is not new. In fact, architecture provides much the standard vocabulary for talking about software development. It is no surprise that a community like OOPSLA so readily appropriates the work of an architect like Christopher Alexander. However, the validity of the present argument depends on the analogy holding at specific points.

Of course the analogy breaks down somewhere, and one place is the potential (and, usually, actual) complexity of software that come from its flexibility and detachment from physical reality. As Fred Brooks puts it, "[Software is] such a tractable medium. The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air. . . " (Brooks, 1995, pg 7). "The floor plan of a building helps both architect and client evaluate spaces, traffic flows, views. . . . The reality of software is not inherently embedded in space. Hence it has no ready geometric representation . . . " (Brooks, 1995, pg 185)

This makes a difference in evaluating the idea that mob software would make programming more accessible to the masses, that someone could modify his or her spreadsheet program just like a Levittown resident spends his Saturday afternoon on a home improvement project. No one would mistake the modifications I have made to my home for something done by a professional. For example, there is the towel rod in our bathroom whose wall plate I bent so badly that the brackets are not flush against the wall. It still holds towels. By contrast, any programmer knows that a small tweak to a piece of software could crash the entire system, and you will be at a loss to know even whether the mistake is in the modified part itself or in the way that the part interacts with the rest of the system.

Even if the analogy does hold, Gabriel's vision of the work of the mob may not match how the "mob" worked in architecture. For example, one way in which Gabriel says the free software movement does not go far enough is that it mirrors proprietary software's command-and-control system by having "small core teams led by module owners who are strict gatekeepers" (Gabriel, 2000, pg 26). Gabriel himself quotes Christopher Alexander on cathedral building that although the individual workmen made their own mark on the building, "At any given moment there was usually one master builder, who directed the overall layout" (Alexander, 1979, pg 216). In some cases, the identity of the master planner is preserved. The Rheims Cathedral guidebook says, "It was Jean d'Orbais who undoubtedly conceived the general plan of the building. . . . This is one of the reasons for the extreme coherence and unity of the edifice" (Eschapasse, 1967). The cathedrals may not even be monuments to a mob working apart from detailed plans. Christopher Alexander's denial that "some great architect created these buildings, with a few marks of the pencil, worked out laboriously at the drawing board" is called into question by the existence of architectural plans for Gothic cathedrals very similar to the plans architects draft today (Böker, 2005). The precedence for mob activity within the analogy is incomplete.

Hacker lifestyle. The most questionable comparison, however, is not between software and architecture. It is the suggestion that the early hackers provide any sort of a model for "the ways that people work effectively" (Gabriel, 2000, pg 1) or an alternative to the "error prone, hectic, family-destroying, health-degrading, night-haunting" (Gabriel, 2000, pg 14) mode of work that the commodification of software has forced on programmers. Even a superficial read of *Hackers* reveals a near-universal theme of characters who sacrifice the rest of their lives for their hacking. When one hacker's "computer widow" left him, "it was a jolt that many Homebrew members—those who had convinced a woman to marry a computer addict in the first place—would experience. 'I would say the divorce rate among computer scientists is almost 100 percent—certainly in my case,' Gordon French later said" (Levy, 1994, pg 235–236). In fact, the "way of hacking you like" (Gabriel, 2000, pg 1) with which Gabriel tantalizes his audience embodies the very stereotype that keeps

the masses away from computing. This is my consistent experience in efforts to recruit students without prior programming experience to computer science courses.

In summary, this calls into question Gabriel's enthusiasm for mob software. There is insufficient evidence that the masses want to participate in the way he supposes, or that they would be happy with the software they would produce if they did. Moreover, the model he proposes—the masses customizing their software in the way they customize their houses—fails at certain points, and the examples of past mob software he cites are unlikely to appeal to the masses.

5 Response

How Christ's lordship influences the production of software is hard to see or articulate—if there is any way. This is why I find Gabriel's ideas worthy of attention. How should a Christian respond to this? More generally, how should a Christian regard himself or herself in the field, and how readily should we participate in the goals and projects of the community?

5.1 What Christians can affirm

One reason I have not been able to ignore this essay is because there is so much I find attractive in it. I would also like to see coding appreciated for its art and beauty, and to see a wider range of people learn to program. (One motivator, I suppose, is job security—for an academic program like the one I am in to survive, we need a wider variety of students to take a second major in computer science, or a minor, or even a computer science course or two.)

The craft of coding. The mental activity that I here want to make obedient to Christ is not the general computer use in which nearly everyone in modern society participates. Nor is it as wide as the field of computer science, in which I teach. Our present focus is specifically on software production. In fact, more specifically than that, we are concerned with the task of programming or coding, which is only one step in the software development process (other activities include specification, design, documentation, testing, and maintenance). This is the activity that Gabriel is trying to exalt. In the current, gloomy situation, Gabriel says, "the focus is on architecture, specifications, design documents, and graphical design languages. Code as code is looked down on. . . . We find little or no code education" (Gabriel, 2000, pg 17)

Those who have never programmed will have trouble conceiving the joy many of us find in crafting computer code. (Students who have suffered through a required introductory programming course which was not designed for them and for which they were unprepared are likely even more skeptical.) It is, however, a universal experience of those who have gotten deeply into programming. Fred Brooks, a Christian in the field of computer science, attributes the fun of programming to "the sheer joy of making things. . . the pleasure of making things that are useful to other people . . . the fascination of fashioning complex puzzle-like objects of interlocking moving parts and watching them work . . . the delight of working in such a tractable medium" (Brooks, 1995, pg 7). Joseph Weizenbaum considers extreme cases of this to be a new mental disorder, "the compulsion

to program," which he compared with compulsive gambling (Weizenbaum, 1976, pg 115–122).

What is programming? Is it a leisure activity, like gardening, where one can find some quiet solitude, put in hard work but at an unregulated pace, and reap results after patient labor? Is it a cultural activity, like music or cooking, where one can tune and showcase his or her talents for his or her own enjoyment or others'? Or is it a technological activity, like engineering, in which tools are used or fashioned to make life's mundane tasks easier? Programming is all of these things, and one who has experienced its thrill can only hope to see more people find it enjoyable and useful. Levy's hackers—Gabriel joins Levy in lionizing them—understood coding this way, their "Hacker ethic" including such propositions as "You can create art and beauty on a computer" (Levy, 1994, pg 43) and "Computers can change your life for the better" (Levy, 1994, pg 45).

Like other cultural activities, programming is worth examining in asking what it means to be human. Put positively, the unlimited creativity of computer programming is an expression of our following God's example. Quoting Brooks again, "I think [a programmer's] delight must be an image of God's delight in making things, a delight shown in the distinctness and newness of each leaf and each snowflake" (Brooks, 1995, pg 7). Don Knuth said, "I think people who write programs do have at least a glimmer of extra insight into the nature of God. . . because creating a program often means that you have to create a small universe" (Knuth, 2001, pg 168).

Put negatively, the power a programmer has over the computer appeals to the rawest form of our sinful nature, the temptation to be like God. Weizenbaum explains,

> *The computer programmer, however, is a creator of universes for which he alone is the lawgiver. . . . [U]niverses of virtually unlimited complexity can be created in the form of computer programs. . . . No playwright, no stage director, no emperor, however powerful, has ever exercised such absolute authority to arrange a stage or a field of battle and to command such unswervingly dutiful actors or troops.*
>
> *One would have to be astonished if Lord Acton's observation that power corrupts were not to apply in an environment in which omnipotence is so easily achievable. It does apply. (Weizenbaum, 1976, pg 115)*

Levy similarly describes the attitudes of the hackers (though without noting any danger): The computer "had made them masters of a certain slice of fate. . . . *Like Aladdin's lamp, you could get it to do your bidding*" (Levy, 1994, pg 45–46). The ancient appeal of sorcery is that one could exercise control over supernatural powers. The analogy between programming and sorcery is not lost on the computer science community. One of the most respected introductory programming texts "is dedicated, in respect and admiration, to the spirit that lives in the computer" (Abelson and Sussman, 1996, pg v) and says in its opening paragraph, "We are about to study the idea of a *computational process.* Computational processes are abstract beings that inhabit computers. . . . People create programs to direct processes. In effect, we conjure the spirits of the computer with our spells" (Abelson and Sussman, 1996, pg 1). (One of the authors of that text is one of Levy's hackers.)

The Christian Programmer. How can we characterize a programmer with commitment to Christ? The easiest way is to take those things which are true for a Christian in any labor or activity, and

see how they apply to coding and participation in the software community.

The First Commandment, according to the Westminster Shorter Catechism, teaches us "to know and acknowledge God to be the only true God, and our God; and to worship and glorify him accordingly" (Westmisnter Shorter Catechism, 1647). To acknowledge and glorify God certainly includes avoiding all danger and temptation to making a god of oneself. Likewise, for any ability we may have or accomplishments we may produce we must thank God and acknowledge that they belong to him. Gabriel says, "software skill is a gift from an unknown and unknowable source" (Gabriel, 2000, pg 30). Though in context his concern at least partially is to distinguish between taught and innate skill, we must resolutely say that software skill, like all others, is a gift from God, and God is knowable.

Similarly, the Christian programmer must avoid hubris. Though it may sound strange to think of geeks this way, a sort of machismo pervades the hacker culture and the OS/FS movement, a commerce of pride and glory. Eric Raymond describes how the volunteers who have contributed to Linux are compensated by "the intangible of their own ego satisfaction and reputation among other hackers" (Raymond, 2000, pg 22). Discourse surrounding many OS/FS projects in online forums is frequently acrid and prideful, unaccepting to newcomers or those perceived to have less ability. Gabriel acknowledges that for Mob Software to flourish, there needs to be "a culture similar to the current open-source culture, but ... much more accepting of users, especially novice users" (Gabriel, 2000, pg 28). A Christian's witness for the gospel is a more critical reason for exemplary public behavior.

In addition to pride, the Christian programmer will find himself or herself in a culture that encourages and glamorizes addictive and compulsive behavior. We have already spoken of Levy's hackers and Weizenbaum's compulsive programmers, who "cannot attend to other tasks ... when [they are] not actually operating the computer. [They] can barely tolerate being away from the machine. ... The compulsive programmer spends all the time he can working on one of his big projects" (Weizenbaum, 1976, pg 118). Margolis and Fisher, likewise, describe the "person in love with computers, myopically focused on them to the neglect of all else, living and breathing the world of computing," (Margolis and Fisher, 2002, pg 65). The image of such people, they say, turns many people, especially women, away from the field of computing. Whether for work or leisure, the Christian programmer must exercise the same cautious moderation in his or her programming as he or she does towards all other things. The Westminster Larger Catechism considers it to be part of our obedience to the Sixth Commandment ("you shall not murder") to engage in "a sober use of meat, drink, physic, sleep, labor, and recreations" (Westminster Larger Catechism, 1647, QA 135) and to avoid "immoderate use of meat, drink, labor, and recreations" (Westminster Larger Catechism, 1647, QA 136).

Nevertheless, as with other aspects of work and culture, this is not reason for the Christian to withdraw. The period between Christ's ascension and return, when Christians are "in the world" (John 17:11) yet not "of the world" but "chosen out of the world" (John 15:19), is comparable to that of the Israelites in exile: God's holy people, living among the heathen, awaiting deliverance. Jeremiah wrote to the exiles telling them to settle and participate in the culture in which they found themselves. "Build houses and live in them; plant gardens and eat their produce. ... [S]eek the welfare of the city where I have sent you into exile, and pray to the LORD on its behalf, for in its welfare you will find your welfare" (Jeremiah 29:5 & 7, ESV). Nothing here is unique to the one

involved with software production, but the principle applies if this is the work which one's hands find.

Similarly, there is a general need for Christians to work with an exemplary level of excellence, both for imitating God and promoting a good name for ourselves and the gospel. The Christian who is involved in programming, whether commercial or as part of the OS/FS movement, must have courtesy and integrity which will gain notice. Software development is a domain in which there is much room for shoddy work as well as opportunities for good work to stand out. A Christian programmer should consider it a duty that any software which will be used by others would have good design, thorough testing, and adequate documentation.

A final arena where ethical concerns affect the Christian programmer is that the Christian should look for opportunities to use his or her craft to support the work of the gospel and to show mercy to those in need. Karl-Dieter Crisman has pointed out that many participants in the OS/FS movement believe they are contributing to social good in the sense that the software they produce is available without fee and so is accessible to those with fewer resources, especially the developing world, and for this reason Christian participation in the OS/FS movement is consonant with Christian stewardship (Crisman, 2008, pg 11–12). The Christian programmer will want to use his or her abilities and opportunities so that Christian love for the needy will outshine any secular concern for social good.

5.2 Critique

Gabriel's vision of mob software attains many of the deseridata I have described above. Its realization would spread the appreciation of coding to many people. Excellence would be encouraged. The willingness to help and share would encourage the development of software for social good. This paper has uncovered various points in which Gabriel's vision is unrealistic and his claims unproven, though not all of these observations damage the thrust of his argument or proposal. Instead, I submit the following summary of why a Christian should not participate in the movement as Gabriel presents it.

First, it is founded on a presupposition that God is not the orderer of the universe, specifically Stewart Kauffman's ideas of order emerging from chaos as a fundamental law of the universe. He sees this as something to replace the ancient—especially Christian—understanding of God as a source of sacredness and meaning. Gabriel uses this assumption to disdain authority and any deliberate organization. If this proposed view of nature is completely wrong, then one would have to conclude that mob software simply would not work; that is, it predicts a certain result based on an incorrect idea of how the world works. Even if aspects of Kauffman's model were established and redeemed—meaning that order arising from chaos is a consistent pattern in God's providential ordering of the world—the Christian would need to observe God's glory and acknowledge him in the process.

Second, the vision puts excessive hope in a human endeavor, essentially a false gospel. The story of the Tower of Babel teaches us the futility and rebellion of humanity working in concert: Under the threat of being scattered across the face of the earth, humans try to make a name for themselves in pride and confidence of their collective ability. God foils their attempt to dethrone him (Gen 11:1–9, ESV). In mob software Gabriel expects "the spirit of xenia [to] raise to the

highest status everyone with a stake in the community. Users participate in design at all levels of scale, and projects are begun specifically to address the needs or wants of a particular user community. . . . Resource scarceness created by artificial boundaries no longer exists, and in an era of abundance, excess thrives" (Gabriel, 2000, pg 28–29). Christians cannot share his optimism. We know fallen nature itself will interfere with any predicted altruism.

Finally, Gabriel's proposal is imbibed with a non-Christian spirituality. At the end of Gabriel's presentation at OOPSLA, he bowed and placed a small box on the stage, which to me appeared to be an offering to the duende. To achieve great things, the programmer is expected to invoke a dark spirit from folklore which rewards haphazard recklessness. This must conflict with the Christian programmer working dutifully and responsibly, prayerfully acknowledging a personal God who has apportioned talents as he has willed, the author of order and peace.

5.3 Application

Much of this has been autobiographical. Hearing Gabriel's talk made me realize the need to consider how the way we think of the world affects the way we program and teach programming, and whether it limits the way we can participate in software projects. The need to evaluate Gabriel's claims led me through a forest of different fields which all teach us something about software production in one way or another. This paper is something of a report on that journey.

In the spring of 2009, I taught the senior capstone course in our department for this first time, titled "Social and Ethical Issues in Computing." One paper we read as a class was entitled "How Computer Systems Embody Values" (Nissenbaum, 2001). Though an interesting title, more than one student expressed deep skepticism that computer and software systems could reflect the values of the people and organizations that developed them (the contents of the article, in fact, did not explain it or adequately convince the students). I believe, however, that we do see values and beliefs reflected in the way we program and what we program, but that the reflection comes only in subtle ways, aspects like the purpose and motivation for the software and the human interaction around the development of the software, much more so than the design or coding itself.

There are parts of the OS/FS movement to which Christian programmers can make good contributions and which they can channel for their own priorities. A cultural appreciation for coding and other aspects of software can develop, and the benefits modern society has obtained through computing technology can be shared with others. However, my experience with Gabriel's mob software proposal—first hearing it originally presented and then tracing out its details—has taught me that the Christian programmer must be aware of the sources of the ideas around him or her and weigh carefully where they will lead.

Acknowledgements. I thank David Cook and Bob Brabenec, who served as my official readers for this paper as my faith-learning project at Wheaton College. I came to David with the vague idea of how I should respond to Gabriel's essay, and he helped me shape my ideas into something coherent. I also thank Cary Gray and Terry Perciante for kindly reading my outlines and drafts, and providing feedback. Cary in particular offered corrections on the technical and historical material and pointed me towards useful references.

References

Abbate, J. (1999). *Inventing the Internet*. Cambridge, MA: MIT Press.

Abelson, H. and G. J. Sussman (1996). *Structure and Interpretation of Computer Programs* (second ed.). Cambridge, MA: McGraw Hill and the MIT Press.

Alexander, C. (1979). *The Timeless Way of Building*. New York: Oxford University Press.

Alexander, C., S. Ishikawa, and M. Silverstein (1977). *A Pattern Language*. New York: Oxford University Press.

Böker, J. J. (2005). *Architektur der Gotik : Bestandskatalog der weltgrössten Sammlung an gotischen Baurissen*. Salzburg: Pustet.

Bratt, J. D. (Ed.) (1998). *Abraham Kuyper: A Centennial Reader*. Grand Rapids, MI: Eerdmans.

Brooks, Jr., F. P. (1995). *The Mythical Man-Month* (Anniversary ed.). Addison-Wesley.

(2005). *The Confession of Faith and Catechisms*. Willow Grove, PA: The Committee on Christian Education of The Orthodox PresbyterianChurch.

Crisman, K.-D. (2008). Open source software and mathematics: Pedagogy and principle in the Christian classroom. Third-Year Faculty Integration Paper at Gordon College.

Epstein, J. M. and R. Axtell (1996). *Growing Artificial Societies*. Cambridge, MA: MIT Press.

Eschapasse, M. (1967). Reims Cathedral. Caisse Nationale des Monuments Historiques, Paris. Quoted in Brooks Brooks (1995).

Free Software Foundation (2007). The free software definition. http://www.fsf.org/licensing/essays/free-sw.html (7/16/2008).

Gabriel, R. P. (1989). Lisp: Good news, bad news, how to win big. Keynote address at EuroPAL conference. http://www.dreamsongs.com/WorseIsBetter.html (7/25/2008).

Gabriel, R. P. (2003). Master of fine arts in software. http://www.dreamsongs.com/MFASoftware.html (7/18/2008).

Gabriel, R. P. and R. Goldman (2000). "Mob Software: The Erotic Life of Code". Dreamsongs Press. http://www.dreamsongs.com/Files/MobSoftware.pdf (7/16/2008). Originally distributed in booklet form at the OOPSLA 2000 conference. Pagination differs slightly between the original published form and the electronic version. Our citations use page numbers from the electronic version.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

García Lorca, F. (1998). *In Search of Duende*. New York: New Directions. Compiled by Christopher Maurer. Originally published 1955.

Grabow, S. (1983). *Christopher Alexander: The Search for a New Paradigm in Architecture*. Stocksfield, Northumberland, UK: Oriel Press.

Hacker (1996). The Hacker's Dictionary. http://www.ccil.org/jargon/jargon_toc.html (8/27/2009).

Hyde, L. (1983). *The Gift: Imagination and the Erotic Life of Property*. New York: Vintage Books. Originally published 1979.

Hyde, L. (2008, Winter). Gifts that keep on giving. *WSJ.*, 84–90.

Kauffman, S. (1995). *At Home in the Universe*. Oxford UP.

Kierkegaard, S. (1967). *Søren Kierkegaard's Journals and Papers* (5 ed.), Volume 3. Bloomington, IN: Indiana University Press. Edited and compiled by Howard Vincent Hong and Gregor Malantschuk. Translated by Howard Vincent Hong.

Kierkegaard, S. (1999a). *Against the Crowd*, Chapter 6, pp. 23–24. In Kierkegaard Kierkegaard (1999b). Also in (Kierkegaard, 1967, pg 305-311).

Kierkegaard, S. (1999b). *Provocations: Spiritual Writings of Kierkegaard*. Farmington, PA: Plough Publishing House. Edited and compiled by Charles E. Moore.

Knuth, D. E. (2001). *Things a Computer Scientist Rarely Talks about*. Stanford, California: Center for the Study of Language and Information.

Kuyper, A. (1998). "Sphere Sovereignty", pp. 461–490. In Bratt Bratt (1998).

Levy, S. (1994). *Hackers: Heroes of the Computer Revolution* (2nd ed.). Harmsondsworth, Middlesex, England: Penguin Books. Originally published 1984.

Madden, W. A. and K. Y. Rone (1984). Design, development, integration: Space Shuttle primary flight software system. *Commun. ACM 27*(9), 914–925.

Margolis, J. and A. Fisher (2002). *Unlocking the Clubhouse: Women in Computing*. Cambridge, MA: MIT Press.

Maurer, C. (1998). "Preface", pp. vii–xi. In Maurer García Lorca (1998).

Morris, I. (1986). Gift and commodity in archaic Greece. *Man* (N. S.) *21*, 1–17.

Nissenbaum, H. (2001, March). How computer systems embody values. *IEEE Ccomputer 34*(3), 120, 118–119.

Oxford English Dictionary (2009). *The Oxford English Dictionary*. http://www.oed.com (6/17/2009).

Oxford University Press (2009). http://www.oed.com/about/history.html (6/18/2009).

Raymond, E. (2000). The cathedral and the bazaar. http://catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/cathedral-bazaar.ps (6/23/2009).

Reynolds, C. (2001). Boids. http://www.red3d.com/cwr/boids/ (8/28/2009).

Reynolds, C. W. (1999). Steering behaviors for autonomous characters. In *The Proceedings of Game Developers' Conference, San Jose, CA*, SanFrancisco, pp. 763–782. Miller Freeman Game Group.

Schaff, P. (Ed.) (1977). *The Evangelical Protestant Creeds, With Translations*, Volume III of *The Creeds of Christendom*. Grand Rapids: Baker. Originally published by Harper and Row, 1877. Available at http://www.ccel.org/ccel/schaff/creeds3.html (6/23/2009).

Thomas, L. (1974). *The Lives of a Cell*. New York: Viking.

VanDrunen, T. (2009). Flocking pattern simulation. http://cs.wheaton.edu/˜tvandrun/skeets/skeets.html (6/16/2009).

Weizenbaum, J. (1976). *Computer Power and Human Reason*. San Francisco: W. H. Freeman and Company.

Westminster Larger Catechism (1647). The Westminster Larger Catechism. Confession of Fatih (2005).

Westmisnter Shorter Catechism (1647). The Westminster Shorter Catechism. (Schaff, 1977, pg 676–700).

Wikipedia (2009). http://en.wikipedia.org/wiki/Lewis_Hyde (7/18/2008).