Software Development as Spiritual Metaphor

Richard G. Epstein
Department of Computer Science
West Chester University of Pennsylvania
West Chester, PA 19383

Introduction

In every era human beings have sought spiritual meaning in their work. What is the spiritual meaning of being a farmer, a carpenter, or a shoemaker? The major spiritual traditions have seen the physical world as an outer manifestation of an inner realm beyond space and time, the realm of the spirit, of the sacred. The patterns in the outer world were seen as symbolic (poetic, theatrical, artistic) expressions of that inner reality. Occupations, such as being a farmer, could be viewed as carrying metaphoric representations of that inner reality.

This paper explores the possibility of finding spiritual metaphors in software development, an important occupation in the contemporary world. In particular, we will view the work of the software developer as an outer manifestation of the inner reality of character building, the attempt to perfect the human character, what some authors have called "work on the soul" [1]. We will find that if we look at work in software development as a metaphor for this kind of spiritual work (the work of perfecting human nature), then we will find many symbolic meanings. It may even be the case that work in software development carries more elaborate and comprehensive spiritual meanings than work in traditional occupations, such as that of the farmer, the carpenter, or the tailor. Given the nature of the world economy that we are building, an economy in which software is becoming ubiquitous, this kind of investigation into software development as spiritual metaphor may prove a means of encouraging people from different cultures to understand the work of perfecting human nature in a new way. At the very least, those who work in information technology will be able to avail themselves of a set of spiritual symbols and references that may prove both humorous and inspiring.

This paper will look at some of the basic concepts in software development and we will explore how these concepts can be recast as symbolic allusions to the work of perfecting human nature (the work of "character building")." While the author is especially interested in drawing upon the intellectual framework provided by the world's spiritual traditions (especially Islam, Judaism, Christianity, Buddhism and Hinduism), it is also the case that some theories of ethics (like the virtue ethics of Aristotle and Plato) derive from the idea that ethical people are people who have refined their character. Hopefully, philosophers and those who are schooled in one or more or the world's spiritual traditions will enjoy this paper both for its humor and its (hopefully, provocative) insights into what software developers are doing on a profound, symbolic level.

The particular concepts from software development that we shall recast as spiritual metaphors include desirable software qualities (which turn out to be desirable qualities for human beings as well), software processes (which turn out to be symbolic of particular approaches to spiritual discipline), significant principles for developing quality software (which turn out to express fundamental truths on the human level) and issues in software security (which easily translate into metaphors relating to the struggle between the spiritual aspirant and those who would derail the aspirant from his/her spiritual journey). The author believes that the Universe (or, God) is creating software at this point in human history precisely because we

human beings can learn a lot about ourselves, our thought processes, our ethics and our morals, from a study of software systems and how they are developed. The author views the Universe as the greatest of the holy scriptures, a profoundly revealing scripture that many people are ignoring.

Signs

Every spiritual tradition refers to the outer world as a symbolic manifestation of the inner world. (This view of reality has some resonance with Platonic thought, which did influence the thinking of theologians in Judaism, Christianity and Islam.)   For example, an important concept in Islam (and in Sufism) is that everything in the manifest world is an *ayat*, or a sign, of the reality of God.   According to the Islamic tradition, God spoke the following words through the Prophet Mohammed, I was a hidden treasure and I loved to be known, so I created the Universe. God was hidden, unmanifest (what is called the *Ayn Sof* in Jewish mysticism), and God wanted to go through the process of Self-discovery, so God created this universe and perhaps other universes. The manifest world is God's attempt to know and to understand God's own qualities, God's own nature. Creation is an act of Self-Revelation and according to the great Sufi teachers, like Muhaiyaddeen ibn Al-Arabi and Jalal-a-din Rumi, it is through the human being that God knows God's Self.

Some Christian mystics expressed the same idea with the phrase *liber mundi* or book of the world [1].   According to this concept, nature is a holy scripture that needs to be studied with the kind of devotion and creativity that one uses to study the traditional holy books. Nature is a holy book and, as such, needs to be read with the great creative power of the human imagination. This creative imagination will enable the human being to see the symbolic meanings behind everything. What is seen on the outside is a poetic, theatrical and artistic expression of the ultimate truth beyond space and time. We shall refer to the idea of the book of the world repeatedly in this paper. Hindus use the word *leela* to express the idea that the Universe is a Divine drama, a drama in which God is both the playwright and the actor who plays all the roles. Although Buddhists don t use theistic language in their description of reality, they distinguish between that which has no birth and no death (nirvana) and the world of birth and death, of impermanence (samsara). In Buddhist terms, the world of samsara is a continuous, precious, and meaningful lesson plan arising from the ultimate Truth.

In viewing the physical world as being filled with spiritual meaning, as being a book that needs to be read and understood, we might ask about the spiritual meaning of a particular type of work. If we see a farmer working in the field, we might ask about the inner reality that this work expresses. As we watch the farmer sowing seeds across a field, and we might realize that we sow seeds of various kinds in our inner beings, the seeds of love and compassion, or the seeds of hatred and jealousy. Several months later we might see the same farmer reaping the harvest, and we realize that the seeds that we have sown will yield their own kind of harvest. Acts of love and compassion will have one kind of consequence, and acts of hatred and jealousy will have another. When the farmer pulls out weeds from the ground, we can view that as a reminder that we, too, must identify the weeds that are growing within us. We need to separate the weeds from the wheat. If we do not pull out our own weeds (for example, the weeds of hatred and bigotry), then those weeds might very well crowd out the abundant wheat harvest that can sustain us through a long and hard winter.

When the author was first exposed to this kind of symbolic thinking quite a few years ago, he started to think about the spiritual meaning of his work as a professor of Computer Science. He thought about it for a while and he realized that he spent a lot of time meeting with students who would come to his office with "bugs" in their programs. His work was to identify the problems and to help the students to fix their bugs. It occurred to the author that his work as a Computer Science professor was an outer manifestation of an inner reality — the reality of repairing things, of fixing flaws and imperfections. His work of fixing bugs was an outer manifestation of an inner reality, the reality that the Universe (according to the author's view of things) is constantly attempting to fix things, to repair the damage that is being wrought by human ignorance, stupidity and violence. When we look at the civilization that we currently inhabit, it is obvious that there is a lot of work that needs to be done in this direction.

This view of God or the Universe as a force that is trying to pull human civilization into a higher state of consciousness, a more compassionate and loving and creative consciousness, is wonderfully expressed by the contemporary Sufi teacher Pir Vilayat Khan in his book "Awakening" [2]. Pir Vilayat is the son of Hazrat Inayat Khan, the Indian Sufi who is credited with introducing Sufism into the West about eighty years ago. Pir Vilayat Khan (like his father) was profoundly respectful of all religious traditions and he continues to interact with leaders within the Jewish, Christian, Hindu and Buddhist communities. Because this paper focuses on the spiritual journey as involving the perfection of character, the development of virtue and compassion, we will use this vision of the Universe (or God) as a power that is trying to pull humanity into a more peaceful and compassionate future.

Our purpose in this paper is to explore the many symbolic meanings (or, "signs") lurking in software development. A fundamental aspect of software development is to perfect things, to get rid of the bugs, and that will be our primary focus, because the emphasis in this paper is on software development as a metaphor for character development. However, there are many other spiritual meanings in the work of the software developer. This absolutely must be the case if our spiritual traditions are to survive into a future in which computer technology will be ever more ubiquitous. If technology is spiritually meaningless, then aren't we building a civilization that will be spiritually meaningless as well?

The Basic Goal in Software Development

Let us now consider the spiritual metaphors at play in software development. Suppose we consider a typical software developer at work. Just as someone who observes Professor Epstein trying to fix bugs in a student's program might surmise that this is an outward manifestation of a basic cosmic process (for example, the process by which the Universe is trying to pull human beings toward a more peaceful and compassionate future), so we might consider what our typical software developer is fundamentally trying to accomplish. The fundamental goal of the software developer (at least, ideally) is to make the customer happy, or to satisfy the customer. The customer has a new computer application that needs to be developed and the customer is asking the developer to understand the requirements and to build such a computer program.

But, who is the customer, at least metaphorically? Adopting Pir Vilayat Khan's evolutionary language (and the great Sufi poet Jalal-a-din Rumi used this kind of evolutionary language almost eight hundred years ago), let us say that the Universe (God, the Buddha nature, the True Self) is trying to pull the human race forward in its evolutionary development, towards

a state of greater awareness, perfection, love and compassion. So, the Customer is a metaphor for the Universe and the application that the Customer (note the capitalization of this word, which we will use throughout the rest of this paper) wants the developer to create represents a new development in the evolution of human consciousness, something that has not existed before (at least in that person). So, in trying to satisfy the Customer, the software developer is working on creating something new, something that has not existed before, something that represents another step forward in the development of human society, in human consciousness.

As the software developer performs the outer work of building this new computer system, we shall view this as a metaphor for the inner work that this same software developer must perform in order to perfect his or her character. This inner work is being done at the urgings of the Cosmic Customer (the Universe). This inner work is the work of perfecting the human character, of becoming more aware and moral and compassionate. This is the inner work that the outer work (the work of designing and coding and testing) symbolizes.

It is interesting to read the "book of the world" with this kind of understanding. For example, if we read an article like Peter Denning and Robert Dunham's "The Missing Customer" [3], we soon discover that their article is filled with provocative spiritual meanings. Denning and Dunham argue that one of the most important issues for software professionals is to develop an appreciation for the customer and the fundamental importance of satisfying the customer in the whole enterprise of software development. So, let us suppose that we are observing the world of software development with the intention of understanding it as metaphorical, as being a holy book with deep, spiritual meaning. Then, Denning and Dunham's observation that software developers often do not appreciate the importance of satisfying the customer, and that software developers may not even have the skills to communicate and interact with the customer, is a provocative spiritual assertion. One can easily arrive at the spiritual meaning of this assertion by converting "customer" to "Customer" in the previous sentence. Denning and Dunham's article can be viewed a symbolic reference to the fact that human beings (in general) are not aware of the importance of attuning themselves to the Universe (this Cosmic power that is trying to draw humanity forward, away from barbaric violence, into harmony and peace and compassion for "the least among our brothers"). Human beings are not aware that our purpose in life is to make the Customer happy. Furthermore, a metaphorical interpretation of Denning and Dunham's article would be that human beings do not even know how to communicate with the Customer (the Universe). We don't even know how to determine what the Universe wants or what the Universe is trying to tell us.

In fact, there is even more to Denning and Dunham's article if we read it in *liber mundi* terms, using the language of the Christian monks, because Denning says that the educational institutions (in particular, colleges and universities) are resistant to adopting the view that they should be satisfying the customer. He quotes from a faculty document at his former university to this effect. Reading that document in *liber mundi* terms we see that many of our institutions of higher learning are resistant to communicating the idea that human consciousness should be evolving, that we should be resisting hatred and violence and injustice. In short, many institutions of higher learning don't care about the Customer either. As you can see, the *liber mundi* makes for good reading.

Desirable Software Qualities

A software system should have the properties of being:
- correct
- reliable
- robust
- user-friendly
- portable
- interoperable
- reusable
- extensible
- maintainable
- efficient
- secure
- ethical and socially responsible

As we observe a software developer trying to incorporate these qualities into his or her software, how can we interpret these efforts in *liber mundi* terms?

As it turns out, these desirable software qualities are also desirable qualities within human beings. Thus, we can say that the software developer's attempt to incorporate these features into her software is symbolic of her attempts to perfect her character along the same lines. The Customer would definitely like to see human beings evolve in the direction of these inner qualities. A human being who develops these qualities is certainly a human being who is awake, compassionate, and moral. In other words, desirable qualities for software in the outer world are metaphors for desirable human qualities in the inner world. In the rest of this section we will examine how each of these desirable software qualities translate into human terms.

For software, the quality of being correct means that the software satisfies the customer's requirements, or makes the customer happy. The quality of being reliable means that the software satisfies these requirements consistently over time. The quality of being robust means that the software is correct and reliable, even under unusual circumstances.

In the *liber mundi* ("book of the world") these qualities translate to the idea that the human being should satisfy the basic moral and ethical principles of the Customer (the Universe, God, the Buddha nature, the True Self), and that the human being should do this in a consistent manner, and even under difficult circumstances. These are demanding qualities and the software developer's efforts to achieve them are symbolic of our efforts to achieve these qualities in our own lives. It may be relatively easy to be ethical and moral when the circumstances are not too demanding, but the third quality (robustness) demands that we achieve these qualities even when we face unusual and difficult circumstances. We must have within our character the ability to respond to unusual and perhaps unexpected inputs (like a hurtful insult from a loved one or a friend or from a reviewer of a paper that we have written) without violating the correctness requirements of the Customer. For example, striking someone back in anger for a hurtful insult would be an example of violating the quality of robustness. Robustness demands that we have the ability to process unexpected insults without violating the Customer's requirement that we control our anger. When a human being behaves in an unusually hurtful manner without any external provocation that is an example of a violation of reliability. Reliability requires consistent behaviors within the guidelines of the "requirements." In fact, it is possible to explore the human ramifications for each of these qualities (and the other qualities that we shall describe

in the following paragraphs) at some length, but the limitations of space will not permit that in this paper.

In software, the quality of user-friendliness has to do with the manner in which the system interacts with human users. The system should not demand unusual and time-wasting behaviors from the users. The system should not make it easy for users to make mistakes and get into a "dead end" situation. The system should present the user with clear instructions and guidelines for interaction with the system. The system should have a sensible and pleasing appearance.

Obviously, user-friendliness is a human quality that we could discuss at great length. Let us first examine just a few of the ways in which human beings might violate this particular quality. Human beings should be considerate of others and not make excessive demands of others. A person who is always late for meetings is not being user-friendly. A person who wastes other people's time with endless gossip is not being user-friendly. A person who closes down a friendship because his friend made a few incorrect keystrokes is not being user-friendly. A person who does not communicate clearly and with patience and with deep listening is not being user-friendly. A person who forgets to bathe and whose appearance is sloppy and disheveled is not being user-friendly. A user-friendly person is one who interacts with others with compassion and with deep listening. He actually hears what the other person is saying and appreciates what is being communicated. He reacts to those communications in an appropriate manner and is flexible and responsive to a variety of inputs.

The qualities of portability and interoperability in software systems are related. Portability usually refers to the ability of a program to operate in a variety of environments, for example, on top of a variety of operating systems (e.g., Linux and Windows). Interoperability refers to the ability of the software application to interact with a variety of existing applications. An example of interoperability would be the ability to import information from a database into a spreadsheet, or the information from a spreadsheet into a PowerPoint presentation.

In human character terms, portability is the trait of being able to move to a new environment without crashing. The new environment might be a new cultural environment or a new work environment or a new social environment. For example, when a business person is asked to relocate to a new country (for example, from the United States to Japan or visa versa) if that business person can function well in both environments, we would say that this business person has displayed the property of portability. Interoperability refers to the ability to interact with different types of people, perhaps people from different ethnic, religious, and cultural backgrounds, or people from different socio-economic backgrounds. So, portability and interoperability are character traits that have to do with flexibility, the ability to function under radically new circumstances and to interact effectively with people whose life experiences might differ radically from one's own life experiences.

Reusability refers to the ability of a software component to be reused in a new program so that the developers of the new program do not have to "reinvent the wheel." In human terms, reusability has to do with sharing knowledge that one has gained so that other people do not have to go through the process of obtaining that knowledge "the hard way." Almost all knowledge sharing can be viewed as the human analogy for software reuse. For example, when parents try to teach their children appropriate forms of behavior based upon their own life experiences, this is a kind of reusability. They are transferring what they have learned to their children to help their children avoid some of the difficulties they themselves have encountered in their lives.

For software, extensibility refers to the ability to add new components to a piece of software so that it will exhibit new functionality. Extensibility is a double-edged sword. Extensibility might mean that we could add a new component to a piece of software that makes it less reliable, less user-friendly, or less secure. However, as a positive trait, extensibility means that we can add new functionality that improves the performance of the software from the customer's perspective.

In human terms, extensibility refers to the ability to acquire new knowledge and new behavior patterns. As with software, this new knowledge or these new behavior patterns, can either be for good or for ill. In terms of striving for perfection, extensibility means that this new knowledge and these new behavior patterns will be in the service of the other desirable qualities (like user-friendliness). For example, if a human being acquires a new behavior module that allows that human being to curse and to be rude, that would not be consistent with the goal of striving for perfection of character. But, other thought and behavior modules that human beings acquire can be consistent with the goal of perfecting human character, of becoming a moral and compassionate human being.

Software has the property of maintainability if it can be (relatively) easily modified either to correct defects or to satisfy the changing requirements of the customer. Maintenance may also be required to improve system performance or to handle anticipated changes in the environment in which the software is operating. Maintainability is an especially important software quality because so much effort is devoted to software maintenance.

In liber mundi terms, maintainability refers to human flexibility in many different dimensions. First of all, it is a highly desirable human character trait to be able to fix one's own defects. If one has a character flaw (for example, if one has the tendency to be rude or inconsiderate), then being able to fix that flaw is a positive character trait. The spiritual journey fundamentally has to do with this ability to continuously monitor one's emotions, thoughts, and behaviors with the intention of transforming the hurtful emotions, thoughts, and behaviors into compassionate and helpful emotions, thoughts, and behaviors. Furthermore, sometimes we have to change because the Customer changes our circumstances. For example, the Customer might decide to force us to lose our employment, in which case we had better have the property of maintainability, so we can respond to these new circumstances. Perhaps the Customer has decided to challenge us with a health crisis. Again, maintainability, the ability to adjust to these new requirements, is a positive character trait. Maintainability in human terms also has to do with refining our behavior in order to perform better and more efficiently, and also to have the ability to anticipate future circumstances so that we will have appropriate responses to those anticipated events.

Software is efficient if it performs its functions in a reasonable amount of time. Clearly, this is also a desirable human character trait. The Customer might want us to accomplish something in our lives and it is best not to keep putting it off. Eventually, death comes and it will be too late to make the contribution that we were mulling over for so many years. Also, efficiency is important in our interaction with other human beings. In that context, efficiency is an aspect of user-friendliness. If we agree to do something for a colleague and then fail to do that within a reasonable amount of time, then we are lacking the quality of efficiency.

Software needs to be secure. That means it must not be vulnerable to attacks from malicious parties who might want to interfere with the correct functioning of that software. Software security goals include confidentiality, integrity, and availability. A software system should ensure that only authorized users get access to information, that the information that is

stored is correct and can only be modified by authorized personnel, and that the software system and its information be available to authorized parties at all times.

The human implications of software security are enormous, and we will devote an entire section of this paper to these issues. In some sense, software security is a symbolic manifestation in the outer world of the human inner reality that many people fall prey to the messages of hatred, violence, and injustice that swarm about us, like worms propagating through the Internet. All issues in computer security are powerful symbols of the human spiritual reality. We human beings must protect ourselves from attacks from malicious parties who would steer us away from the path of morality and compassion, love and tolerance. The messages of hatred and intolerance circulating in this world are very much like computer viruses and worms, and the intolerant messages that some of our parents or cultures might have left in our brains when we were growing up are very much like Trojan horses. We, as human beings, are responsible for protecting the confidentiality of some of the information that has been shared with us. We are responsible for protecting the integrity of the information that we have. We must not distort our memories (although some of the viruses that infect us may try to distort those memories for malicious ends). Finally, we must be available to those who deserve and expect our attention and help. Yet, we are vulnerable to viruses and worms that are tantamount to denial-of-service attacks, encouraging us to withdraw from human interaction and from a life of serving others.

Software should have the qualities of being ethical and socially responsible. A software system should be designed so that it does not cause unnecessary harm, and so that it protects the privacy of individuals. A software system should be designed so that it does not harm the environment or the social fabric of a just and harmonious society. Obviously, human beings need to be ethical and socially responsible. This has been expressed in various ways in the preceding paragraphs. Ethicality and social responsibility manifest in a variety of ways, including what we called user-friendliness, correctness, maintainability, portability, interoperability, and security.

The author feels that he has just touched the tip of the iceberg in terms of how desirable software qualities translate into human terms. Hopefully, the reader is convinced that the book of the world, including this current chapter that is filled with computer technology, is indeed a good read!

Software Development Processes and Principles

A software development process (or, process model) is a defined methodology for developing software. There are literally dozens of such processes and they differ in many interesting ways. In *liber mundi* terms a software development process is a metaphor for a spiritual practice. The spiritual literature contains many dozens of practices that one may fellow, including meditation and prayer practices, practices that involve working for social justice and peace, practices that are designed to help practitioners transcend the world of appearances, practices that are intended to help practitioners develop their creative imagination, and practices that are designed to help human beings perfect their character. Again, our focus in this paper is on this last kind of practice, on practices that are designed to help human beings become more virtuous and ethical, more compassionate and caring. So, the world of software development provides dozens of metaphors for how to engage in a spiritual practice, and we shall explore several of these. In particular, in the following sections we shall examine the life

cycle (or, waterfall) model, eXtreme programming, and Open Source software development as metaphors for spiritual practices (or, disciplines).

A software development process is an agreed upon methodology for developing software. It is described in terms of specific steps and methodologies. We shall view software development processes as metaphors for the various disciplines that human beings might apply to themselves in order to perfect their human natures. The starting point for this metaphor is the fundamental principle, which we have already discussed, that the goal of software development is to make the customer happy. In spiritual terms, we are viewing the Customer (the Universe, God, the Buddha Nature, the True Self) as a vast intelligence that is trying to pull the human race forward in its evolutionary development. The Customer is trying to pull human beings to a higher level of spiritual development, a level of development that is marked by an awareness of the interconnectedness of all beings, an awareness that is deeply rooted in love and kindness, compassion and tolerance. So, we shall view each software process as suggesting a particular approach (or collection of approaches) for how we human beings might attune ourselves to that inner pull, the pull of the Customer."

Before exploring the spiritual lessons hiding in specific software processes, we will discuss four basic principles of software development. These are:

- Principle #1. Make the customer happy.
- Principle #2. Four eyeballs are better than two (the "many eyeballs" principle).
- Principle #3. Remove defects as early as possible
- Principle #4. No software process is appropriate for all development environments.

These are significant, widely held principles in a field (software development, or software engineering) that is filled with creative ideas as well as controversy. The controversy about how best to develop software in the field of software engineering symbolizes the fact that there is great controversy among the spiritual communities as to which spiritual practice is the most efficacious for an individual person or group of persons. Despite the controversies about how best to develop software, almost all software engineers agree with these four principles. We have already discussed the fundamental principle that the goal of the software developer is to make the customer happy, so let us move on to the other three principles.

The many eyeballs principle. This second principle states that two eyeballs are not enough to create defect-free and correct software. According to this principle, all code that is put into production must be viewed by at least four eyeballs and preferably more than four. In other words, all code should be subject to some form of peer review, such as a code inspection process. Many software development experts have warned against the seemingly talented developer working in isolation without permitting his or her work to be reviewed by others. This is one characteristic of what Steve McConnell [4] calls a "problem programmer."Steve McConnell believes that programmers who refuse to have their code reviewed by others should be fired.

In *liber mundi* terms, the problem programmer symbolizes the person who is introverted to a dangerous degree, the loner in the extreme sense of that word. The loner refuses to accept feedback and criticism from others. The loner is not open to new experiences and new learning. Obviously, such a person will have difficulty perfecting his or her character because such a person is reluctant to engage in those kinds of experiences that can provide important life lessons. So the spiritual meaning of the many eyeballs principle is that people who seek to perfect their character should be part of a community of people with similar spiritual goals (such

as making the Customer happy). If you want to make the Customer happy, you cannot work in a cubicle all by yourself.

Remove defects as early as possible. The third principle that almost all software developers (or, engineers) accept is the idea that defects (or, "bugs") need to be removed from the software as early as possible. The longer a defect remains in a piece of software, the more costly it will be to remove it. Ideally, defects should be found and removed before testing, because testing is only a filter. It will only find a certain percentage of the defects.

Obviously, this principle corresponds to an important spiritual reality. On the journey of becoming a moral and ethical person, it is important to remove our defects as early as we can. If as a teenager we have a certain defect (like the tendency to lie), then it is best to transform that defect into the virtue of truthfulness during the teenage years, rather than waiting before one is on trial before a jury for corporate fraud in one's latter years. The longer a defect remains in our character, the more difficult and costly it will be to remove it.

No software process is appropriate for all development environments. The fourth and final principle that nearly all software engineers accept is that no particular software process is appropriate for every environment. Certain types of applications (for example, safety-critical applications) may require the "big up-front design" approach of the life cycle model. This requires a tremendous amount of effort "up front" to gather the requirements from the customer and to design the system architecture and components before any code is written or tested. On the other hand, there are other types of applications (e.g., Web applications that are not safety or finance critical) that are best served by following an agile process like eXtreme Programming. Even the main creators of eXtreme Programming (folks like Kent Beck [5]) emphasize the need to customize eXtreme Programming to one's particular development circumstances.

The spiritual message in this particular principle of software development would seem to be that different life situations and different cultures and different personality types will find different spiritual practices more appropriate for their individual circumstances. If the Universe is trying to teach us about human nature through the manifestation of computer technology, then one of the messages the Universe seems to be communicating to us is that we need to be tolerant of different approaches to spirituality. While we need to acknowledge the spiritual tradition (software development process) that represents our ancestral tradition (what we were taught about software development at school and at work), nonetheless, we need to be open to the lessons offered by alternative traditions (processes). There is so much that we can learn from one another (both for finishing this Web app and for continuing our journey towards a kind and gentle heart). That being said, let us now look at the spiritual meaning of three important approaches to developing software: the life cycle (or waterfall) model, eXtreme Programming, and Open Source software development.

The Life Cycle Model

The life cycle (or, waterfall) model views software development as progressing somewhat linearly through the phases of analysis (requirements gathering), design, implementation, testing, and maintenance. The life cycle model can be viewed as a metaphor for an approach to spiritual development that begins with a long period of introspection (the requirements gathering phase). The subsequent phases of the process (design, implementation, and testing) are intended to make sure that the customer's requirements are being satisfied. Eventually, the product is released to the customer and will need to be modified if defects are

found or if the environment in which the software is functioning has changed. In the following paragraphs we will explore the various phases of the life cycle model (analysis, design, implementation, testing, and maintenance) and we will interpret each phase as a phase in a spiritual practice that is intended to help one to perfect one's character.

In the world of software, analysis is about finding out what the customer wants. This can be a very complicated and time-consuming process. It involves a lot of communication with the customer (if it is done correctly) and the result is a written agreement (even a business contract) documenting what the customer will get when the project is finished.

In *liber mundi* terms, the life cycle model best fits a highly disciplined approach to spiritual development that begins with a long period of introspection and meditation and prayer, during which the individual uses spiritual practices in order to establish some kind of communication with the Customer. In some spiritual traditions, this will require an intense period of introspection, perhaps with some kind of inner dialogue with God, for the purpose of determining what it is that God wants from this person. In other spiritual traditions (for example, Buddhism) the dialogue might be with holy beings (bodhisattvas), or with one's own teacher. Even the great Greek philosophers (like Plato and Aristotle) stressed the need for this kind of introspection in order to determine one's fundamental values, one's fundamental moral principles. Beyond that, these spiritual practices are about trying to determine the nature of reality and the purpose of the individual's very existence. Analogous to the requirements document that results from the analysis phase of a software project, this period of intense introspection might lead to written documents (perhaps in the form of journal or diary entries) or at least conscious decisions about these various dimensions of human experience, including fundamental values and truths. The fundamental output of the analysis phase for the spiritual aspirant is a profound understanding of what the Customer wants, what truly constitutes a moral and ethical and worthwhile life.

Once the analysis phase is finished, the life cycle model of software development progresses to the design phase. During the design phase decisions are made about the basic software architecture that will be needed to achieve the requirements determined during analysis. The software design phase progresses from high-level architectures to more and more detailed decisions about how to effect the eventual implementation. The result is an understanding of the components of the system and how they will interact.

In *liber mundi* terms the design phase is about making basic decisions about how to achieve the goals that were revealed during analysis. For example, during the analysis phase an individual might discern that her purpose in life is to be a healer. During the design phase that person might decide that she wants to become a psychiatrist, a healer of the mind, so she decides that she must go to college and eventually to medical school. The design phase on the spiritual journey is of this nature — making basic life decisions in order to realize the objectives (or requirements) that were revealed during the analysis phase.

We should mention at this point that all software processes involve some aspects of analysis, design, implementation, testing, and maintenance, but most of the alternative processes (such as the eXtreme Programming process that we shall discuss in the next section) are highly iterative (rather than linear) in the approach they take to achieving these ends. In a highly iterative spiritual process, analysis and design (gathering requirements and making life decisions) might occur on a daily basis. Therefore, there are radical differences between software processes both as software processes and as metaphors for spiritual practices. In this paper we can only

touch upon some of the differences between these radically different approaches to spiritual development.

In the life cycle model, the implementation phase involves transforming the design into actual code in a specific programming language. In spiritual terms this corresponds to actually doing the work that is required to bring the design to life. In the case of the young woman who decided that she wanted to be a psychiatrist, the implementation phase would correspond to her actually attending college and getting into medical school, and getting her M. D. so that she could become a psychiatrist.

The testing phase is the next phase in the software life cycle. This involves taking the code that was developed and to look for defects in the code. The software is tested in a special test environment, not in a real-world context (at least when the software vendor is ethical about the testing process). In spiritual terms the testing phase involves checking to see if the implementation strategy is actually producing what the Customer wants. For example, it might be that after finishing medical school, our psychiatrist discovers (during her internship at a local hospital) that clinical practice is not allowing her to become the person that she wants to become (according to her original analysis of her life's purpose). She might decide that becoming a university professor would be more consistent with the Customer's requirements, so she might pursue a doctorate or seek a teaching position at a research-oriented medical school.

The maintenance phase in the software life cycle involves modifying the software as required due to a variety of possible circumstances. These include fixing defects that are found during operation, changing environmental conditions, changing requirements coming from the customer, or the need to tune system performance in a variety of ways. In spiritual terms, maintenance corresponds to allowing continuous feedback from the Customer in order to perceive changes in the requirements or to perceive defects in one's character or in one's understanding of the person that one should be. It might be that our psychiatrist finishes medical school, practices psychiatry for many years, but in so doing learns many new expectations from the Universe. She discovers new ways in which the Universe wants her to help heal humanity. Perhaps she volunteers to help inmates at a local penitentiary. Perhaps she continuously seeks to perfect the healing arts that she has learned, so that she can serve her patients with ever more love and compassion and competence.

In the above paragraphs we described the life cycle model in linear terms. In reality, the life cycle model is not perfectly linear. There are feedback loops to earlier stages (including, renegotiating requirements) as the reality might dictate. Furthermore, there is more and more interest in alternative development models (almost all of whom are highly iterative in their approach to software development). Nonetheless, the life cycle model introduces the basic terminology of software development (the ideas of analysis, design, implementation, testing and maintenance).

One particular implementation of the life cycle model is Watts Humphrey's Personal Software Process (PSP) [6]. Without getting into the details, the PSP is almost blatant in its openness to interpretation as a metaphor for self-improvement (or, character development). There are two basic aspects of the PSP: time management and defect management. The PSP requires a tremendous amount of data-gathering relating to how the developer spends his time day by day, hour by hour, minute by minute. This data-gathering improves the developer's ability to develop period plans (to plan out how he will spend his time week by week) and product plans (to plan out the phases involved in product development). In addition, the PSP

requires careful tracking of defects and it emphasizes in a powerful and convincing way the importance of finding those defects as early as possible (before the first compile!).

The spiritual metaphors for the PSP are the ideas of careful observation (of mindfulness) and of careful record keeping (e.g., keeping a journal). The person who is following a PSP-like spiritual practice is carefully noting her emotions, thoughts and actions and is recording them in an appropriate journal or diary. She realizes that the only way that she can improve her spiritual state (her character) is by careful observation of how she reacts to events, how she spends her time, with special attention being paid to defects of character that need to be transformed. She realizes that the longer she allows a defect to remain in her character, the more difficult it will be for her to track down and remove that defect.

eXtreme Programming

The second software process that we will examine is eXtreme Programming (XP). XP is called an agile process and among agile processes it is the most influential. The basic problem with the waterfall model is that the customer doesn't get to experience the final product until late in the life cycle. One goal of XP is to give the customer many intermediate versions of the software, which allows for more interaction between the customer and the software as the software is being developed in a series of iterations. Another aspect of XP is that a customer representative should be a member of the development team, allowing for continuous interaction between the developers and the customer. There are about a dozen basic practices in XP. We shall examine some of these along with their possible interpretation as spiritual metaphors.

The first XP principle, as mentioned above, is that the customer must be on site at all times. The customer is viewed as an intrinsic part of the development team. In *liber mundi* terms the idea is that the individual who is trying to satisfy the Customer must be in constant communication with the Customer, refining his or her understanding of what the Customer wants on a continuous basis. This kind of interaction with the Customer implies a high degree of mindfulness and wakefulness. Whereas the life cycle model seems to suggest a spiritual discipline that involves a long preliminary period of introspection, the XP model seems to suggest a spiritual discipline based on "real-time" interaction with the Customer, with the spiritual aspirant incrementally advancing in his or her understanding of the requirements day by day, perhaps hour by hour or minute by minute.

Another aspect of requirements gathering in XP is the use of "user stories" to specify what the customer wants rather than a formal requirements documents. In XP the customer is asked to commit her requirements to index cards in the form of short statements of, say, two or three sentences. In a typical project about eighty of these index cards (user stories) are collected and then they are prioritized in terms of their importance to the customer. The huge formal requirements document that might be the result of the life cycle approach to software development might be seen (in one metaphorical interpretation) as corresponding to a holy scripture, like the Bible or the Qu'ran. This is what the Customer wants! Do it! However, the XP approach is to gather what the Customer wants in small sayings or parables, and then to prioritize those sayings according to the importance that the Customer assigns to them.

Project planning in XP is described in terms of short iterations two to three weeks in length. The overall plan (in terms of the iterations and their goals) is mapped out at a release planning meeting. After each iteration (or perhaps after a few iterations), the developers have software with some functionality that they can offer to the customer (a release). The customer

does not have to wait for many months (or perhaps years) for the life cycle to come to its conclusion in order to obtain software that does something. In spiritual terms, XP corresponds to a process of spiritual development (or character development) that is highly dynamic and energetic. That is, the spiritual aspirant realizes that he must attain certain character goals in short iterations, in a short period of time, with constant interactions with the Customer (by introspection, prayer, or meditation). Every moment is an opportunity to produce something that might be of interest to the Customer.

Another fundamental practice of XP is pair programming. This relates to the 'four eyeballs are better than two' principle we discussed earlier. In pair programming, when applied literally (which is difficult for some organizations), every line of code that will eventually get to the customer (i.e., production code), is produced by a pair of programmers working at one computer. One programmer does the typing and is called the driver. The other programmer looks on and is on the alert for defects and flaws and for possible ways of improving the driver's approach to the code. (For some reason, the XP advocates don't seem to call this second programmer the 'back seat driver'.) If we interpret XP as a metaphor for a spiritual practice, then it would seem that XP requires that every practitioner have a spiritual partner or friend to help them with their practice, someone who will provide constant and insightful feedback on their behaviors and attitudes. In other words, the best way to satisfy the Customer is to have a partner who will constantly alert you to the possibility that you are going off track. The partner is especially important in terms of preventing defects (character flaws) from entering into the product that the Customer will eventually get. The empirical data suggests that pair programming is a very powerful weapon in terms of quality assurance, that is, in terms of preventing defects from being injected into the product. So, it would seem that XP, taken as a spiritual discipline, can make a very strong argument for having a spiritual partner or friend as one advances on one's spiritual journey.

Another principle of XP is that the programmers must write the unit tests before they write the actual code. That is, they must write programs that will be used to test the code modules that they plan to produce. The argument is that when programmers write the unit tests first, this gives them a deeper insight into what that particular module (or, chunk of code) is supposed to do and how it is to perform. One spiritual interpretation for writing the unit tests first is that the spiritual aspirant, before embarking upon a particular course of action, should do some sort of contemplative practice in which the aspirant anticipates how that course of action will play out. So, if the spiritual aspirant decides that the Customer wants him to speak to a person in a position of power (like his boss), then he will first do a contemplative practice which will allow him to play out the possible ways in which this confrontation might unfold. Then, when he actually confronts his boss, there is a greater likelihood that the aspirant will have devised a strategy for handling possible negative reactions from the boss. The contemplative practice allowed the aspirant to run through possible scenarios of this nature.

Other aspects of XP have to do with the way that members of a development team are to interact and communicate with one another. An XP development team is supposed to begin each workday with a stand-up meeting. No one is allowed to sit down at such a meeting, the theory being that when people sit at a meeting they tend to talk too much, or drift off and not pay attention. In a stand-up meeting the developers (and customer) stand in a circle and each developer reports on his or her progress on the project. The result is that the team should have a pretty clear idea of how the team is doing with respect to the current iteration and what problems are arising and might need to be addressed.

In terms of a spiritual practice, the XP stand-up meetings emphasize the need of starting each day with an assessment of where one is in spiritual terms, either in one's own life, or in the context of a family or community. If one is living in a family or in a community, everyone gets an opportunity to report on where they stand on their journey and if problems are raised, the family or community addresses those problems immediately, not sweeping them under the rug.

Usually (but not always) the life cycle model is implemented in a hierarchical team structure, where there is a hierarchy of power. You might have upper level managers and middle level managers and team managers, with software developers at the lowest level of his power hierarchy. There are many possible spiritual interpretations for the organizational structures in the world of software development. One spiritual interpretation for the hierarchical team structure might be in terms of a hierarchy of powers in the spiritual universe, which some religious traditions allude to (in terms of God and archangels and angels). Another interpretation is institutional, that the hierarchy represents a church with its leadership structure. The XP model is highly democratic. Everyone participates on equal terms. Even the Customer is just a member of the team. Some might say that an XP stand-up meeting is somewhat like a Quaker meeting, but this is not exactly true. At an XP meeting everyone is expected to give a report on where they are, whereas at a Quaker meeting one only speaks when moved by the Holy Spirit.

One final XP principle that we shall discuss (there are others which we did not discuss, but which can also be cast in spiritual terms) is the forty-hour workweek. Many organizations have difficulty implementing this perfectly, but the idea behind this is that software developers who work more than forty hours per week will not be able to produce high quality software products because they will be stressed out and over-worked. Developer burnout and stress are factors that lead to the introduction of defects into a software product. A good software developer has time for relaxation, family, and fun. Seen as a spiritual metaphor, the forty-hour workweek principle bears the message that a spirit of fun and light-heartedness is important for the spiritual journey. Someone who devotes every minute of their life to spiritual progress, to spiritual attainment, without a smile on their face, is probably not following a path that leads to peace and happiness. The author heard an anecdote about Thich Nhat Hanh, the much-loved Zen teacher (and New York Times best-selling author) who visited a Zen center in the United States. The members asked him what he thought of their spiritual practice and he said, "You should smile more."

Closed Source versus Open Source

Open Source is an increasingly important approach to software development. However, some software engineers are of the opinion that Open Source is not a well-defined software process. It lacks the rigorous specifications of processes like PSP and XP. Open Source is a general philosophy concerning the creation and distribution and sharing of software. The basic issue that we shall focus on is whether the source code should be open (freely available) or closed (closely guarded proprietary information that will not even be available to clients). In particular, what are the spiritual meanings behind these two philosophies concerning how software is created and distributed.

Let us give a brief description of how Open Source code development works. There are many variations on this and different approaches to intellectual property rights with respect to Open Source software [7]. In Open Source software development, an individual or group of individuals come to see the need for a specific tool or application. They develop a prototype for

that tool which represents the starting point for the evolution of the product. That prototype is freely shared over the Internet and people who come to see the value of the core concept can then work with the software, adding functionality, fixing defects, improving performance, and so on. There is a central authority that determines what gets incorporated into the tool or application and what doesn't.' The 'many eyeballs principle' that is so important for software development was perhaps first stated in graphic terms by Eric Raymond, a leading advocate of the Open Source software, who said, 'Given enough eyeballs all bugs are shallow' [8]. One claim made by the Open Source community is that when defects are found in a viable (that means, widely-used) Open Source tool or application, it is fixed with great speed because many experts around the world make an effort to remove the defect.

In the early days of computing, software vendors tended to share the code with their customers. This was before the days of the Internet when sharing code was not so easy. These days, closed source code is closely guarded as proprietary information by the software vendors. Customers get the executable, but not the original source code.

In spiritual terms, one might say that Open Source software development represents spiritual disciplines that are democratic and open, whereas closed source development represents approaches to spiritual development that are closed and elitist. After all, there are mystical schools that have very strict rules and requirements for admission. The spiritual aspirants in these schools do not share what they learn with the broader human community. There are other spiritual schools that are eager to share what they have learned with their fellow human beings. As this paper is being written the SCO lawsuit has become big news. SCO is a company in Utah that is suing IBM because IBM is distributing the Open Source operating system, Linux. SCO claims that it holds the copyright to some of the Unix code that has been incorporated into Linux. The history of religion has had many examples of this nature, where a religious organization that believes it has the sole right to spiritual truth challenges those who wish to take a more democratic view of things. Many religious traditions have been subject to this kind of conflict (between the insiders and the outsiders) in the course of human history.

Computer Security

We have already discussed security as one of the desirable goals for high-quality software. Computer security is an enormous topic. It intrinsically interfaces with the fields of computer ethics, computer law, and many highly technical problems in computer technology. The very language of computer security is rich with spiritual metaphors and, indeed, it would be easy to write a very long narrative on this one subject. In this paper, our intention is just to touch briefly upon just a few of those metaphors. Our focus will be on the basic terms and concepts in computer security, and the spiritual meaning that these ideas bear.

Dorothy Denning has presented a fundamental model for information warfare, which is at the heart of many discussions in computer security [9]. Information warfare involves an attacker and a defender. The defender is trying to protect information assets, including hardware, software and data. The attacker is either trying to get access to those assets or to damage them in some way, either by destroying them or by modifying them with malicious intent. The defender is trying to assure that the information assets retain the properties of confidentiality, accessibility, and integrity. We discussed the spiritual meaning of confidentiality, availability, and integrity briefly in our earlier discussion of desirable software qualities.

In this paper we will examine the spiritual meaning of the computer security issue solely in terms of the Universe (God, the Buddha nature, the True Self) being a power that is trying to pull human beings forward in the evolution of their consciousness. The very language of this presupposition (the existence of such a power with such an intention) implies the existence of countervailing powers or influences, energies that are trying to prevent the human community from evolving in this manner. We usually equate these powers with "darkness" or "evil." These countervailing powers want to pull humanity backwards into the darkness of tribal hatred, brutality, injustice, and hatred.

Within this spiritual framework (that is, in *liber mundi* terms), we might consider the defender in the information warfare model as representing the human being (and, at the macrolevel, those members of the human community) who are trying to respond to "the pull," those who are trying to move forward in their spiritual development, who are trying to perfect their human natures. The attackers, on the other hand, represent the forces of darkness that are trying to keep human beings from evolving in this manner. The attackers are not so much on the outside, as on the inside. They represent the malicious software (thought patterns, habit energies) within each human being that tries to deter that human being from spiritual progress. This malicious software may take the form of hateful messages passed on from generation to generation, or hurtful habit patterns and thoughts that the human being holds on to. According to this metaphorical interpretation of the computer security situation, human beings are being assaulted by many attackers (often called demons in the spiritual traditions). These attackers are trying to get the human race to turn back from the spiritual path.

There are two dimensions of computer security that we will especially focus on. One is the fact that software vendors have been somewhat irresponsible in the deployment of computer technology. This is one aspect of what James Moor called the "ethics gap" in his paper on character development within the computer technology community [10]. Vendors often create software packages, some of great import, that have vulnerabilities that attackers can exploit. But, this is an exact analogy to the human situation. The human being has many vulnerabilities that the attackers can exploit. All human beings who are trying to evolve in the direction of virtue and compassion will find themselves attacked by a legion of demons that are trying to exploit their particular vulnerabilities. Many of these vulnerabilities derive from cultural prejudices and learned responses to life's events. So, the vulnerabilities in real-world software are symbolic of our human vulnerabilities, our human tendency to submit to the messages of hatred, greed, and violence.

The second dimension of computer security that is rich with spiritual meaning is the world of computer crime. Every kind of computer crime has some kind of spiritual metaphor within it. These include identity theft, malicious hacking, violations of privacy, stealing intellectual property, and creating malware (malicious software). We as human beings have vulnerabilities, and the computer criminals (on the spiritual level) have many tools for exploiting these vulnerabilities. In the realm of malware they design extraordinarily sophisticated attacks to exploit specific vulnerabilities, and these attacks are becoming ever more damaging in their impact. For example, viruses and worms represent the spiritual reality of how demonic ideas spread within the human community, causing people to live in fear and to perpetuate injustice upon those who are powerless. Sometimes our own parents or teachers might plant a Trojan Horse or two in our software, giving us the tendency to react to situations in ways that are contrary to what the Universe requires, namely that we live with compassion and love for our fellow beings.

In a distributed denial-of-service attack the attacker tries to gain control of thousands or even millions of computers that will act as "zombies" in a concerted attack on a target. The popular 1981 movie, "My Dinner with Andre," used the term "zombie" to describe people who are living in a dream world, people who are dominated by fear. So, the spiritual meaning of a denial-of-service attack might be an attempt by the forces of evil (which the author is defining as the forces which are trying to pull us backwards in terms of our spiritual evolution) to demoralize human beings so that they do live in fear, and do not act with full awareness and appreciation of the life that is right in front of them.

Computer security is likely to be a major issue in the realm of computer ethics and computer technology for the coming decade (at least). In practical terms, it brings to the fore the kinds of issues that computer ethicists have been discussing for quite a few years, what James Moor called "the ethics gap." In spiritual terms it is a theatrical presentation by the Universe of the basic human problem, that we are vulnerable to fear and exploitation. In spiritual terms the solution to the computer security problem is for each of us to strive to develop integrity, what some have called "authentic presence," in order to protect ourselves from the attackers who spread malicious ideas and who wish to bring human spiritual progress to a halt.

Conclusions

This paper was an attempt to show that the world of software development is literally filled with spiritual metaphors and lessons. According to the spiritual view of things (which we have captured with the phrase *liber mundi*, or "book of the world") everything that manifests in our life bears lessons and teachings. We need to learn how to read this book of the world, for doing so will greatly enrich our lives. Software development (and, more broadly, computer technology) is an important reality in the book of the modern world. The software development chapter in the book of the modern world can be read as being rich with spiritual lessons and insights. This paper was an attempt to explore just some of those insights. In particular, we explored how desirable software qualities correspond to desirable qualities for the human being, and how different software processes correspond to different approaches to spiritual practice. We also explored some of the spiritual metaphors present in the (somewhat scary) world of computer security. The basic goal of this paper was to convince the reader that the computer technology chapter in the book of the world is a really good read.

References

1. Moore, T., Care of the Soul: A Guide for Cultivating Depth and Sacredness in Everyday Life, Harper Collins, New York.
2. Khan, V. L., Awakening: A Sufi Experience, Tarcher Putnam, New York.
3. Denning, P. J., and Dunham, R., "The Missing Customer," Communications of the ACM, March 2003, pp. 19-23.
4. McConnell, S., "Problem Programmers," IEEE Software, March/April 1998, pp.128-126.
5. Beck, K., eXtreme Programming explained: EMBRACE CHANGE, Addison-Wesley, Boston, 2000.
6. Humphrey, W. S., Introduction to the Personal Software Process, Addison-Wesley, Reading, Mass, 1997.

7. Wu, M., and Lin, Y., "Open Source Software Development: An Overview", *IEEE Computer,* June 2001, pp. 33-38.

8. Raymond, E. S., <u>The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary</u>, O'Reilly and Associates, Cambridge, Mass., 1999.

9. Denning, D. E., <u>Information Warfare and Security</u>, Addison-Wesley, Reading, Mass., 1999.

10. Moor, J. H., "If Aristotle Were a Computing Professional", <u>Computers and Society</u>, September 1998, pp. 13-16.

About the Author

Dr. Richard G. Epstein is a Professor of Computer Science at West Chester University of Pennsylvania in West Chester, PA. His main interests are software engineering, computer ethics, and the social implications of computing. These interests are reflected in his book, <u>The Case of the Killer Robot</u>, and in many of his stories and plays. Professor Epstein won the ACM SIGCAS Outstanding Service Award in 2003 for his contributions to computer ethics and the social implications of computing. The "Wheel" article and many of his stories and plays are available at <u>www.cs.wcupa.edu/~epstein</u>. He can be reached at <u>RGEpstein@aol.com</u>.