Spring 2021

Understanding God and His Redemptive Work in Computer Science

Mike Wallinga
Northwestern College - Orange City, mwalling@nwciowa.edu

Understanding God and His Redemptive Work in Computer Science

Dr. Mike Wallinga

10/15/2020

Introduction

Computer science as a discipline and computer scientists as a group tend to exhibit a technical, skills-focused disposition. Curriculum standards demand students acquire a long list of mathematical and programming abilities. Students both expect and are excited by the opportunity to learn the ins and outs of cutting-edge technology. Furthermore, programming a computer is an exercise in exactness and specificity. Is the semicolon in the right place? Are all of the parentheses matched? Does this Boolean logic or loop control statement contain an off-by-one error? Failing to state all of the required commands fully and exactly will result in a program that works incorrectly, incompletely, or not at all. It is no surprise that so much classroom time is spent on the technical details involved in writing code.

This emphasis leaves little room for contemplative, philosophical reflection, yet the guiding principles of Northwestern College include "seeking opportunities for growth and reflection that integrate faith and learning" and "discerning and developing unique gifts in service to Christ, the church, and the world Christ loves and redeems" [31]. How can this be accomplished in computer science? What can computer scientists learn about God through engaging their discipline? How can computer science be used to pursue God's redeeming work in the world?

Programming as an Act of Creation

Although computer science encompasses more than writing code, coding is a foundational aspect of the discipline. Students begin their program of study with introductory programming courses, and programming remains a vital part of nearly every course, research project, and paper. Thus, it is fortunate - one might say, providential - that coding provides a window through which to view and understand God.

Creating Something from Nothing

This window into understanding God is illustrated foremost by God's power as creator of the universe from nothing mirrored by computer scientists' ability to create their own virtual worlds from nothing via coding. Genesis 1 describes how "in the beginning when God created the heavens and the earth, the earth was a formless void" and John 1:3 says "all things came into being through him, and without him not one thing came into being" [49]. When a computer program is prepared to run, its code - the combination of data and instructions that define every detail of that program's execution - is loaded into the computer's memory, effectually creating a complex, well-defined world where nothing had previously existed.

Coding is a mixture of both technical and creative expertise, a melding of the mechanistic and the artistic. Technical skill by itself will not produce a successful program; an elegant design is required. In his seminal work *The Mythical Man-Month*, Fred Brooks explains the pleasure derived from building efficient and useful systems from such a complex, malleable tool as a computer. Brooks declares, "as the child delights in his mud pie, so the adult enjoys building things, especially things of his own design," and posits that "this 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" [7]. Writing for *Dynamic Link*, Calvin College's publication of Christian perspectives on software development, senior architect Bruce Abernethy admits, "while there are many compelling aspects to the craft of software design and development, the major satisfaction I can testify to, over and over again, is to be part of the work that is done to design and build something that has never existed before" [1].

Like Brooks, Professor Keith Vander Linden of Calvin College directly connects this creative aspect of computer science to our status as image-bearers of God [45]. Donald Knuth, Professor Emeritus of the Art of Computer Programming at Stanford and widely heralded as the "Father of Computer Science," takes this connection even further. Not only does the creative aspects of programming grant us an appreciation of God and His creation, but it provides us with "at least a glimmer of extra insight into the nature of God for that very reason, because creating a program often means that you have to create a small universe" [28]. Knuth continues, "many of today's large computer programs rank among the most complex intellectual achievements of all time. They are absolutely trivial by comparison with any of the works of God, but still they're somehow closer to those works than anything else we know" [28]. Covenant College's Dr. John Hunt agrees, "since starting with fundamental items of 1's, 0's, and Turing instructions is as close as we are going to get to creating *ex nihilo*, designing software is as close as we are going to get to reflecting God as the creator" [25].

The assertion that programmers create their own small universes is best seen in the design paradigm of object-oriented programming (OOP). In OOP, programs are created by defining *objects*, structures that model unique identity, state, and behavior. One of the distinguishing attributes of OOP compared to other programming paradigms is the tendency to model a real-world entity as a combination of both its attributes (or state) and its actions (or behavior) [29]. In this way, object-oriented programmers define their creation, not only in terms of what it *is* but also in terms of what it can *do*. In many cases, an object's behavior involves interacting with other objects of varying types. Defining the results and consequences of complex interactions between objects in a virtual world is like defining the laws of nature for our physical world. The programmer starts with a blank, empty memory space, and defines all of the entities that will inhabit that space, including their characteristics, their abilities, and their interactions.

A classic example of defining the laws of nature in a computer program is John Conway's Game of Life [20]. The game is a cellular automaton that requires no input beyond its initial state; a "player" provides the starting layout of the environment and watches the subsequent evolution according to a set of predefined rules. The subsequent state of a given live cell is determined by its neighbors; too many neighbor cells causes the cell in question to die due to overpopulation, while too few neighbors results in death due to underpopulation. A cell with the appropriate number of live neighbors continues to live in the next generation, while a dead cell can become a live cell if it is surrounded by an appropriate number of live neighbors (mimicking reproduction). The specific requirements are unimportant for the purposes of this discussion, but serve as an illustration that a programmer has the power to define his or her own rules in the creation of a virtual world.

The Power to Redefine Creation

God not only possesses the power to create an entire universe from nothing and define its laws of nature, but also is able to redefine and make exceptions to those rules when it suits His purposes. Examples found in scripture include the parting of the Red Sea in Exodus 10 (which followed a number of supernatural plagues imposed upon the Egyptians that also clearly defied the usual laws of nature), and the story of the sun standing still "in the middle of the sky and delaying going down about a full day" in Joshua 10 [49].

Programming language theory provides an analogous case of computer scientists' power to define - and then redefine - the rules of their created worlds. When designing a new programming language, designers have the ability to provide *overloading*. This means that the same arithmetic operator (for example, a plus sign) can be used to name multiple functions. In the Java language, the plus sign indicates traditional arithmetic addition when used with numeric variables, but indicates concatenation when used with strings. In other words, "Hello" + "World" = "Hello World" but 2+2=4, not 22.

Language designers often extend this overloading capability to other programmers. This means that a programmer can redefine the behavior of a given operator via custom functions, beyond a language's built-in rules of behavior [38]. Thus, a programmer could choose to redefine the plus sign to indicate exponentiation, so that $2 + 3 = 8$ (because 2 raised to the third power is 8). Such a choice would defy convention and make the code more difficult to read, so it is rarely done in practice, but the fact remains that programmers have an extraordinary ability to redefine standard rules to fit their whims within the confines of their created universe.

Limitations of Computing vs. a Limitless God

While programming provides computer scientists with the ability to create new worlds from seemingly nothing, thus enhancing their appreciation of God as an all-powerful creator of the universe, this analogy should not be mistaken in any way for equality; such a comparison would be foolish and full of hubris. Despite their technical and creative prowess, programmers are absolutely unable to reproduce the limitless power of God, even within their small, independently created fiefdoms. Furthermore, programmers are consistently short-sighted, unable to anticipate the consequences of their own creations, and often find themselves bound by the limitations that God Himself has imposed on our world.

Limitations in Software Design

Beyond the act of creation itself, one of the most mind-boggling, awe-inspiring aspects of God is the hypostatic union, or the doctrine that Jesus Christ was both fully human and fully divine. Within Jesus, these dual natures exist in unity in a single hypostasis, or individual existence. This doctrine was affirmed by the Council of Chalcedon in A.D. 451, although the issue was reconsidered and reaffirmed multiple times afterwards, including the second Council of Constantinople in A.D. 553, the Lateran Council in A.D. 649, and the third Council of Constantinople in A.D. 680-681 [47].

A similar thorny issues arises in the object-oriented programming concept of *inheritance*. Inheritance allows new classes of objects to be defined as extensions of existing classes [38]. In other words, if I create a class derived from an existing class, the new class *inherits* the attributes and behavior from the parent class, while being able to add new attributes and behaviors of its own. For example, in a computer program modeling a university, we might create a base class to model a person, because

there are some attributes that all people associated with the university share, regardless of their role. Everyone has an ID number, a first name, a last name, a mailing address, and so forth. From that base class, we may create two derived classes, one for employees and one for students. An employee object has everything that a generic person has, but adds attributes unique to that role, such as a job title and a salary or wage. A student also has everything that a generic person has, but adds custom attributes such as their choice of major, their grade point average, and so on. Additional classes could be derived from the employee class, subdividing into faculty and staff classes, since those types of employees differ in some notable attributes, such as tenure status or the accrual of paid time off and sick leave.

For better or worse, reality rarely fits into such rigid predefined categorizations all the time; there are bound to be exceptions that cross these boundaries. In some cases, these exceptions happen without issue. For example, an employee may decide to enroll in the university's graduate program to complete an advanced degree; this employee now inherits attributes from two branches of the program's inheritance hierarchy. This person now has both a job title and a major field of study, as well as both a salary and a grade point average, and all of these attributes can coexist without conflict. But, what happens when an employee assumes both faculty and staff responsibilities? This employee now inherits attributes from two branches of the hierarchy, but some of these attributes are in direct conflict with one another. Is this employee eligible for tenure or not? Does this employee accrue vacation time or sick leave? Does this employee need to maintain working hours during spring break?

In programming terms, both the employee-turned-student and the faculty-staff hybrid employee are examples of *multiple inheritance*, and specifically examples of a *diamond inheritance hierarchy*, so named for the shape of its associated inheritance diagram [11]. Picture a graph with a node at the top of a page labeled "Employee," with nodes branching diagonally downwards to both the right and left labeled "Faculty" and "Staff," respectively. An employee that inherits from both the faculty and staff classes would be placed at the bottom of this diagram, with diagonal edges drawn to connect it to both the Faculty node and the Staff node above it, thus forming a diamond.

Multiple inheritance can be a powerful tool in program design, but a diamond-shaped hierarchy can introduce incompatibilities and conflicts, as described by the faculty-staff hybrid example. Because of this, designers of object-oriented programming languages must weigh the pros and cons of supporting multiple inheritance and decide how to handle the diamond situation. Java takes a conservative approach to this issue and disallows true multiple inheritance. Instead, Java includes a separate type of object definition called an interface, and derived classes in Java are allowed to inherit from only one base class and any number of interfaces. In this way, Java avoids the complications associated with the diamond-shaped hierarchy by specifying only one "first-class" inheritance (the inheritance from the base class) [35]. If there are any conflicts between the base class and an interface, the base class always takes precedence. Other languages provide full support for multiple inheritance, but deal with diamond-shaped conflicts differently. Python will default to the base class that appears first in the derived class's header code, in essence giving one base class precedence over the other [38]. Trying to define a derived class with diamond-shaped ambiguities in Eiffel will cause a static semantic error at compile-time, so the program will not run at all until the ambiguity is removed [38]. C++ allows such an ambiguous class to be defined, but still recommends that programmers avoid it [11]. Instead, C++ will produce a static semantic error message only when a programmer attempts to *use* a class member that is ambiguous [38].

In synthesizing all of these design choices, the point becomes clear: human intellect is unable to create a satisfactory solution to a diamond-shaped multiple inheritance problem that does not require

a compromise at some point. In some cases, the possibility is explicitly disallowed, while in others precedence rules give preference to one definition over another. In the most flexible implementations, multiple inheritance is fully allowed as long as ambiguities and conflicts do not arise, but when they do, they create an error state. No existing programming language can completely model Jesus Christ's dual natures of being fully human and fully God; no matter how sophisticated our tools become, programmers are unable to capture the awesome complexities and mysteries of God.

Limitations in Capabilities

Although it is easy to feel otherwise in this age of mind-boggling computing power, Vander Linden admits that our ability to create and calculate has a ceiling imposed by God. "The nature of software is itself governed by laws of computability, which dictate capabilities of and limits on the theoretical power of computation. These meticulously fashioned systems of law bear witness to God's power and creativity" and illustrate our inability to progress beyond what God has granted us [45]. Adams continues, "Christian computer scientists can thus work to discover more about God by discovering computational laws, since doing so will advance our knowledge of this part of God's creation" [3].

For decades, the computer processor design industry enjoyed exponential increases in clock speed and performance (often mistakenly conflated with Gordon Moore's law, which actually addressed the rate of increase of *transistor count,* not speed). However, even the best electrical engineers cannot defy the laws of physics, and in the first decade of the 21st century, the heat generation associated with increased clock frequencies became problematic. Improvements in chip design impressively reduced the required voltages for many processors, mitigating the concerns over heat, but also stalling the previously steady increase in clock frequencies for more than a decade [6]. This has led engineers to focus on increasing computational power by adding more processor cores; even the least expensive general-purpose computers today have at least two cores, if not four, and the largest currently available consumer products include sixty-four cores!

However, this shift in processor design introduced new complexities in algorithm design and coding techniques. Previously valid assumptions in sequential programming about the order of execution of commands and the use of cache memory are often invalid in a parallel program. Thus, simply taking an existing sequential algorithm and translating it into a parallel one is not necessarily the best choice; in many cases a completely new algorithm is required. Unless a parallel program is carefully designed, it will be subject to erroneous updates to data or out-of-sync communications [6]. Even if these design considerations are taken into account, there is a theoretical limit to the speedup that results from parallel computation, defined by Amdahl's Law [27]. In practice, the theoretical limits are overly optimistic because they do not account for the communication overhead incurred between computational units, and obtaining efficiency above 90% is considered an achievement [6]. As is the case with countless other scientific achievements throughout history, the last several decades of processor design has shown that human ingenuity can overcome seemingly insurmountable obstacles with creative solutions. It has also shown that no matter how impressive it may be, human ingenuity is bound by the physical limitations God has placed on creation.

These physical limitations on computing also are seen in an inability to represent the infinite. While God is infinite, and the God-given laws of mathematics include infinity, computers are unable to deal with arbitrarily large numbers because they are subject to physical constraints. The constraint arises from the number of binary digits (bits) used to represent numbers; in the case of integers, a choice of n bits allows for a range of 2^n numbers. When a calculation results in an answer that falls

beyond that range, we say that *overflow* has occurred. Overflow results in some curious errors, such as the addition of two positive numbers resulting in a negative answer, or two negative numbers summing to a positive answer [46]. Michael Scott writes, "computer arithmetic is not as orderly [as mathematical arithmetic]. The problem is that numbers in a computer are of limited precision" [38]. To demonstrate this reality, Scott provides an example involving three integers between two and three billion evaluated in a 32-bit computing architecture. The expression $b - c + d$ can be evaluated safely left-to-right, but the mathematically equivalent expression $b + d - c$ will create an overflow error, because $b + d$ will produce a value that cannot be represented in 32 bits [38]. Over time, computer architectures have moved from 8 bits to 16 bits to 32 bits to 64 bits, greatly increasing the upper bound for computation, and someday we may enjoy 128-bit-capable machines. Yet, no matter how advanced the hardware becomes, an upper bound will exist. Computers, and thus computer scientists, will never be able to fully model infinity or comprehend this attribute of God.

Limitations in Understanding

Not only are programmers unable to fully reproduce the grand scope of God's creation, they are limited in their knowledge of their own creations, unable to fully comprehend their small-scale universes. In sharp contrast to God's omniscience, it is common for programmers to fail to understand the full effects that their code will have on their program, even though they both created the world and defined its rules. Returning to Conway's game of Life, it is famously a deterministic system, which has led many people to study the system and discover interesting starting configurations that lead to surprising, unanticipated behaviors. One of the most famous of these is R.W. Gosper's glider gun, first discovered in 1970. Gosper's arrangement was the smallest known "gun" for forty-five years after its publication, and was the first known finite pattern with unbounded growth. This proved that there can exist initial Life patterns that grow infinitely, a fact that was originally conjectured to be impossible by Conway himself [22]. Knuth cites this anecdote as an example of our limited knowledge in comparison to God's infinite wisdom, noting that "it's abundantly clear that a programmer can create something and be totally aware of the laws that are obeyed by the program, and yet be almost totally unaware of the consequences of those laws" [28].

In summary, the act of programming a computer serves as an analogy that helps us understand God as our all-powerful creator. As image-bearers of God, we are able to exercise our creative and intellectual gifts in such a way that we create our own virtual worlds and define the laws that govern them. However, this is clearly an imperfect analogy that cannot fully represent God's awesome power. Although it may first appear to be true, programmers do not actually create something from nothing; the foundational laws of mathematics and the limiting laws of physics and physical reality were created by God, who has both granted our capacity to create and placed constraints upon that capacity. We lack the ability to fully model God's creative power, and even if we could, we lack the intellectual capacity to comprehend the full consequences of our creations.

Computer Science in a Fallen World

Furthermore, unlike God's creation, the creations of programmers are far from inerrant. Everyone who has used a computer for more than a few minutes has experienced an error message, a bug that produces an incorrect answer, or a system freeze that requires a reboot. These shortcomings and flaws are due to living in a fallen world; even though it exists in a virtual, digital realm, computing is part of God's creation and is not immune to the effects of sin [3]. Like all other man-made creations, our computer programs are destined to fall short of God's perfection.

Vander Linden identifies four potential distortions of computing, caused by fallen people living and working in a fallen world, that prevent us from reaching an ideal vision of computing [45]:

- **Making gods of ourselves.** Like a modern-day Tower of Babel, computer science tempts us to marvel at our own achievements, to "make a name for ourselves" (as stated in Genesis 11 [49]), and to consider our intellect and ingenuity to be so great that, given enough time and resources, there is no problem so great that we cannot solve it on our own accord.
- **Making a god of computing.** Another temptation is to consider computer technology an inherent good and an end unto itself. Much like people who drive their cars off of unfinished bridges because of erroneous GPS directions, we can place too much faith in technology and assume it to be right and good [24].
- **Making a god of technical achievement.** In a discipline focused on skill acquisition and an industry rife with one-upsmanship, many are tempted to both find their self-worth and feel superior over others on the basis of their technical prowess and achievements. Programmers who succumb to this temptation may find themselves empathizing with the author of Ecclesiastes: "I considered all that my hands had done and the toil I had spent in doing it, and again, all was vanity and a chasing after wind, and there was nothing to be gained under the sun" [49].
- **Making a god of money.** A common motivation for people to pursue a career in computing is financial stability and reward. Considering that seven of the top ten largest companies in the world by market capitalization are technology companies (Microsoft, Apple, Amazon, Alphabet (Google), Facebook, Alibaba, and Tencent [43]), and computing-related jobs consistently rank highly on lists of the best-paying entry-level positions [23], this is hardly surprising. However, as Matthew 6 reminds us, we are not to store up for ourselves earthly treasures; financial gain as a primary motivator is neither satisfying nor glorifying to God [49].

When our thoughts and actions are compromised by these distortions, our technological creations are adversely affected, and those creations in turn affect us, creating a sinister feedback loop. Dr. Derek Schuurman has held professorships at Redeemer University College, Dordt College, and Calvin College, given an invited chapel talk at Bethel University, and written numerous publications on faith and technology. In Dordt's *Pro Rege* publication, Schuurman writes, "Technology is not neutral - it profoundly shapes us in unexpected ways... Every technological artifact is created with some kind of bias: it opens up some possibilities, while simultaneously closing down others" [37]. Hunt agrees, saying "God has a purpose to ALL that He has made. Nothing is value neutral; rather all things may be used and understood either properly, for the glory of God, or improperly, without acknowledging Him" [25].

Schuurman cites several examples of technological bias influencing our lives [37]. Search engines decrease our need for memory and recall. Robotic automation changes the nature of modern work and employment opportunities. Social networks redefine our understanding of companionship and interaction, and increasing digital communication causes a decrease in our capacity for empathy. Always-connected mobile devices increase expectations for working and "on-call" hours. The list could go on.

When our quest for automation is driven solely by a desire for greater efficiency and profit margin, we lose sight of the well-being and financial stability of displaced employees. When we view the quality of our person-to-person interactions in terms of how many "likes" and "retweets" we garner, we lose sight of people as relational beings and instead view them as instruments of self-validation. When the convenience of constant mobile connectivity leads to an inability to disconnect from our work, we lose sight of God's mandate of Sabbath rest. In short, when our development and adoption of technology suffers from the distortion of sin, we lose sight of God.

There are few instances of this distortion as poignant or as egregious as the current excitement over so-called "Big Data" and the faith placed in software programs implementing complex predictive models. In her best-selling and award-winning book, *Weapons of Math Destruction*, Cathy O'Neil writes, "the math-powered applications powering the data economy were based on choices made by fallible human beings. Some of these choices were no doubt made with the best intentions. Nevertheless, many of these models encoded human prejudice, misunderstanding, and bias into the software systems that increasingly managed our lives" [32].

It is difficult to overstate the power these fallible algorithms exhibit. Alistair Croll calls Big Data "our generation's civil rights issue," stating that "personalization is another word for discrimination," citing discriminatory examples in housing, bank lending policies, variable credit limits, and insurance rates [13]. In her book, O'Neil addresses faulty predictive models in college admissions, borrowing money, prison sentencing, and hiring and laying off employees [32]. Harvard professor Latanya Sweeney's research uncovered that searching for racially-associated names led to a disproportionate increase in targeted advertising for criminal background checks [42]. St. George's Hospital Medical School built a computer program to automate its admissions process, intending to reduce variability and increase objectivity, but instead introduced a substantial bias against women and minorities because the historical data used to build the model contained primarily white male candidates [30]. Increasingly, it seems every aspect of our lives has been touched by artificial intelligence algorithms, and often with dire side effects.

What is the cause of these awful, if unintended, consequences? O'Neil posits, "this happens because data scientists all too often lose sight of the folks on the receiving end of the transaction... Their feedback is money, which is also their incentive... And the victims? Those folks are collateral damage" [32]. Once again, it is clear that computer technology, programmers, and their creations exist in a fallen state and suffer from the distortion of sin.

God's Redeeming Work in the (Digital) World

So, computer science provides an understanding of God as the all-powerful and perfect creator, but due to sin and the fallen state of the world, that glimpse is imperfect, incomplete, and flawed. As we read in 1 Corinthians 13:12, it is though we see through a glass, darkly [49]. What can be done about this? Several Christian computer science scholars point to God's cultural mandate to Adam and Eve in Genesis 1:28 [49] as evidence of our responsibility to redeem computer science:

- Vander Linden interprets the command to "subdue" the earth as "to control and bring order" to it; when used properly, computer systems can be a useful tool for doing so, and for enhancing and extending God's creation [45].
- Hunt includes computing in the cultural mandate because it "affects all aspects of culture, including understanding and shaping both our physical and intellectual environments" [25].
- Adams finds studying computer science to be an act of obedience, asserting that "if computing is indeed a part of God's creation, then men and women who work to increase mankind's understanding and mastery of computation are obeying God's command with respect to this part of His creation" [4].

Furthermore, Adams and Vander Linden both connect Genesis 1:28 with God's further instruction to Adam and Eve in Genesis 2:15 to "work and take care of" the garden of Eden [49], acknowledging that although we are directed to have dominion over the earth, we are but stewards of it. As Psalm 24 reminds us, "the earth is the Lord's, and all that is within it" [49]. Thus, Adams explains that

"where the cultural mandate tells us that we are free to make use of God's creation, the doctrine of stewardship tells us that we are not free to abuse his creation" [4]. This includes computing.

Schuurman alludes to the repeated use of the words *all things* in Colossians 1 as proof that redemption is about literally *everything*, including computer science, since "all things have been created through him and for him... so that in everything he might have the supremacy" [49]. He also invokes the Kuyperian phrasing, "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!'." Furthermore, Schuurman references 2 Corinthians 5:16-21, in which we are called to be agents of God's reconciliation and actively participate in the renewing of God's world [49], as a scriptural reason to strive for the redemption of computing technology [37].

What Does Redemption in Computer Science Look Like?

How, then, can computer scientists and software engineers go about the redeeming work that we are called to do as followers of Christ?

Good Work as Redemptive Worship

First, good work itself glorifies God. From Colossians 3:23, "whatever you do, work at it with all your heart, as working for the Lord, not men" [49], it is clear that when we use our God-given gifts to the best of our ability, it is pleasing to God. This is as true of coding and algorithm design as any other task. Joel Adams reiterates, "men and women whom God calls to be computer scientists, and who pursue this vocation in a way that is holy and pleasing to God, can thus do so as an act of worship" [4].

But how does worshipping via good work bring about redemption and reconciliation? In Genesis 1, God repeatedly looked upon His creation and pronounced it good [49], before it was tarnished by sin. If the work is performed with whole, honest, sincere effort, and that work makes progress towards an end that is holy and pleasing to God, computer scientists and software engineers will be improving the state of world, inching it ever-so-slightly closer to its original perfection. As Adams puts it, "Christian computational engineers can apply [their] knowledge in the construction of systems that improve on existing ones, with the goal of producing systems that God would think of as good" [3].

Schuurman lists several norms that should guide our technological activities: cultural appropriateness, transparency, stewardship, delightful harmony, justice, caring, and trust. He notes that "all these norms can be summarized by Christ's call to love the Lord our God and to love our neighbor as ourselves. These norms do not dictate exactly *how* to act, but they point a way forward [37]. Being mindful of these norms will help ensure that computer scientists' work is holy and pleasing to God, and that they will be a redemptive agent of change in the world. Schuurman continues,"these norms are not exclusive; they work together and help lead to flourishing and *shalom*. We need to remember that the meaning of technology ought to be service to God" [37].

Following Ethical Standards and Pursuing Social Good

Certainly, Christians do not have an exclusive claim on performing good work that improves the quality of the world and society. Numerous professional organizations consider honest, ethical work that benefits society as an essential trait of successful computing professionals. Even if these organizations do not name their goals in Christian terms, they have recognized the need to adhere

to similar principles in order to avoid harming society with abusive practices. As Schuurman states, "efforts to pursue technology without attention to norms will lead to consequences; creation will ultimately push back" [37].

The primary professional organization for many computer scientists is the Association for Computing Machinery (ACM). The ACM's Code of Ethics, to which all members must voluntarily comply, was revised in 2018 to address the newest developments and issues arising in the field [2]. The code includes general ethical principles, such as to avoid harm, be honest and trustworthy, and respect privacy. It addresses professional responsibilities, such as maintaining high standards of quality and conduct. In its third section, the code addresses leadership principles, such as ensuring the public good is of central concern in all computing work, and managing personnel and resources to enhance the quality of working life. While the ACM's code of ethics may not directly reference scripture or use the same language as Schuurman's norms, it is certainly pointing in the same direction. Software engineering projects that are designed and implemented with strict adherence to the code will undoubtedly improve society and bring creation closer to its redemptive ideal.

Other organizations, even those that are not directly involved in the creation of software systems, recognize the increasingly large role played by software, algorithms, and technology, and they address those issues in their own codes of conduct. For example, the Association for Institutional Research's Statement of Ethical Principles are meant to "guide [institutional researchers] as [they] promote the use of data, analytics, information, and evidence to improve higher education" [5]. The statement addresses multiple issues related to technology, including privacy, confidentiality, and data stewardship.

In addition to existing professional groups adopting new, computing-focused ethical standards and codes of conduct, brand new independent nonprofits have been created to promote and advocate for the ethical use of data and technology. One such organization is Data & Society, founded in 2014 with a mission statement to "study the social implications of data and automation, producing original research to ground informed, evidence-based public debate about emerging technology" [18]. Recent research by Data & Society has covered artificial intelligence, technology's impact on labor and health, and online disinformation [19]. The organization is currently hiring full-time positions and also offers an annual faculty fellowship program.

The call for computer scientists to behave ethically and improve society is not just happening at the professional level. Increasingly, higher education is heeding the call, as well. In the ACM's education-focused journal, *Inroads*, Goldweber, Kaczmarczyk, and Bluementhal write, "as computer scientists and educators, we have an obligation to reflect deeply regarding our mission... in today's world it is insufficient to present computer science as either a purely abstract mathematical discipline or little more than a collection of skills leading to a high paying job... we need to step up and start preparing our students to change the world" [21]. The authors represent Xavier University and Regis University, but they are far from alone in promoting a movement that has become known as "Computing for Social Good." Other universities with established student clubs, organizations, or courses include Stanford [39], Brown[16], Swarthmore [14], Stony Brook University [8], the University of Illinois [12], and the University of Washington [15][17].

Even though these organizations are not explicitly Christian, their goals and values align with God's command to be agents of reconciliation in the world, making them of particular interest to Christian computer scientists. Calvin College professor Harry Plantinga noted that "professional responsibility and ethics are not only found among Christian computer engineers... However, the motivation for subscribing to ethical standards among Christians is different and perhaps stronger. The professional

may find more reason to go out of her way to serve well when motivated by gratitude to God" [34]. Christian computer scientists have a great opportunity to positively influence the world when they participate in secular organizations, as long as that organization's goals and value are not in conflict with their faith. Invoking Matthew 5:13-16 [49], Adams notes that "Christians with computational expertise - whether scientists or engineers - can bring much-needed 'salt and light' to computational research and practice... Christians with a strong ethical sense should be in the vanguard of those making decisions" about technology [3]. A Christian computer scientist or software engineer will find fulfilling, redemptive work by participating in these organizations.

Serving Christian Organizations

Plenty of opportunities exist for computer scientists and software engineers who prefer to use their gifts to serve explicitly Christian organizations. The Christian mission field has found many ways to utilize computer technology, and is becoming increasingly technology-dependent.

JAARS [26] is a non-profit organization providing technical support and solutions in the areas of aviation, technology, multimedia, and training for Bible translation to related organizations such as Wycliffe [48]. JAARS views its role as an essential part of modern Bible translation: "computers, specialized software, reference libraries, and online backups now enhance everyday translation work. With Internet access, translators in remote locations can consult with linguistic specialists far away for help with difficult translation issues. In this digital age, technology is mandatory—and it's transforming the way Bible translators work" [26]. JAARS's FieldWorks translation software has doubled the speed of translating the New Testament into previously untranslated languages, and in some cases, has made translation possible at all [40]. Representatives from JAARS and Wycliffe have visited the Northwestern Computer Science department numerous times in the past to share how technology is used in the mission field and recruit interested students. Most recently, Wycliffe's Jim Leamer spoke in CSC171QR Computer Science I in October 2018. Wycliffe also shares personnel needs with Northwestern computer science faculty via email, and those opportunities are shared with students.

Pioneers [33] is another missions organization that makes heavy use of technology to support missionaries. Most of the Pioneers IT staff is based in their US home office in Orlando, Florida, although support positions are also placed internationally. A Northwestern computer science alum, Jennifer Marks, serves as their global technology advisor, and has been stationed both in the United States and abroad.

In addition to existing mission organizations, brand new Christian groups have been founded to embrace and take advantage of modern technology. One of the most prominent of these is Code for the Kingdom, a "a weekend hackathon and ongoing ecosystem where global issues are tackled from a Christian perspective" [10]. The first Code for the Kingdom hackathon took place in the San Francisco Bay area in 2013. Since then, fifty events have been held in cities around the globe, including Austin, Seattle, Bangalore, Dallas/Fort Worth, Orlando, Raleigh, Nairobi, Los Angeles, London, Jakarta, Guatemala City, Atlanta, Albuquerque, Denver, Nashville, Warsaw, Manila, Chicago, and Bogota. During the hackathons, participants "write code and create technology to help release the oppressed, teach God's Word, heal the sick, feed the hungry, clothe the naked, and support the church and the body of Christ" [10]. One of the organizers for the Nashville hackathon, Adam Murray, is also a senior web developer for World Vision. He described the hackathon as valuable because "programming has a huge potential for bringing people together. At our events, people who want to serve their communities bring their ideas, and we work together to improve and

develop them... it shows that our collaborative work as developers gives us an opportunity to serve those around us" [36].

Research and Theory as Redeeming Work

Because many mission-based organizations rely on computers for their daily operations, there will always be a need for information technology and programming skills, such as network administration, database management, and web development. However, computer scientists and data scientists with more advanced, theoretical, and research-focused aspirations can use their gifts to further God's redemption of creation, as well. Northwestern alum Dr. Michael Holm earned a PhD in mathematics from the University of Nebraska-Lincoln and now serves as the chief data scientist for Covenant Eyes. In his role, he develops artificial intelligence for image-recognition software using convolutional neural networks. Image recognition and classification is an active area of research in computer science, and Dr. Holm is applying his theoretical knowledge and research skills to help Covenant Eyes' Screen Accountability software identify pornography [9].

Another avenue for computer scientists with theoretical knowledge to get involved in God's redeeming work is to engage the church on matters of policy and theology as it relates to modern technological issues. In 2019, the Ethics and Religious Liberty Commission of the Southern Baptist Convention released its statement on artificial intelligence, entitled *Artificial Intelligence: An Evangelical Statement of Principles* [44]. While the SBC is to be applauded for seriously engaging with a current, rapidly-advancing technology at a theological level, the statement was criticized because it "only superficially engages the reality of artificial intelligence. It often reads as if the community of AI scientists and ethicists weren't even consulted. Most signatories are pastors and theologians, and almost none have expertise in artificial intelligence" [41]. Indeed, out of 73 signatories (including at least one with a Northwestern connection, Richard Mouw of Fuller Theological Seminary), only one is obviously a computer scientist: Michael Covington, a senior research scientist emeritus from the Institute for Artificial Intelligence at the University of Georgia. While Dr. Covington possesses impressive credentials and is undoubtedly an expert in the field, the evangelical statement would have benefitted from involving more computer scientists and artificial intelligence researchers in its formulation, if for no other reason than to better establish credibility in the academic community (including the Christian academic community). It appears that Christian computer scientists with research interests have an opportunity (and, one could argue, a responsibility) to become involved with these conversations in the future, and have a hand in shaping understanding and policy as the church wrestles with the implications of cutting-edge technology.

Conclusion

It is clear that computer science has much to offer the Christian seeking to integrate their faith and their discipline. Computer science and the act of programming can illuminate our understanding of God and His awesome power. At the same time, computer science illuminates our limitations and shortcomings in comparison to our Creator. In these shortcomings, we are able to see the negative effects of our fallen condition, and how all of creation, including computer science, has suffered as a result. By recognizing that sinful temptations and distortions negatively affect computer science and technology, we realize that God's desire for us to be His agents of redeeming work in the world applies to these digital spaces, as well. Computer scientists can participate in this redemptive process by giving their best effort and following Christian norms in all of their work, participating in organizations whose ethics and values align with their faith, applying their technical knowledge

to endeavors that will improve society, offering their technical gifts to mission-focused organizations, applying their theoretical and research skills to Christian causes, and participating in the church's broader theological discussions about technology.

As John Hunt puts it, "to see the parallels between God as a designer and man reflecting God's image as a software designer provides an opportunity to understand the gifts God has given us. These gifts allow a calling that is as profound and God-centered as any" [25]. Those seeking to understand God and carry out His directive to be ambassadors of reconciliation can find their calling and vocation in the field of computer science.

References

[1] Abernethy, B. 2012. Code's Creative Spirit. *Dynamic Link - Christian Perspectives on Software Development.* 3 (2012), 8–9. Calvin College. Available: *https://computing.calvin.edu/documents/dynamic-link-journal.html*.

[2] ACM Code of Ethics and Professional Conduct: 2018. Available: *https://www.acm.org/code-of-ethics*.

[3] Adams, J. 2001. Computing Technology Created, Fallen, in Need of Redemption? *The Proceedings of the Conference on Christian Scholarship... For What?* (Sep. 2001). Calvin College. Available: *https://computing.calvin.edu/documents/christianity-and-computing.html*.

[4] Adams, J. 2002. Why Christians Should Study Computer Science. *Christianity and Computing.* (2002). Calvin College. Available: *https://computing.calvin.edu/documents/christianity-and-computing.html*.

[5] AIR Statement of Ethical Principles: 2018. Available: *https://www.airweb.org/ir-data-professional-overview/statement-of-ethical-principles/principles*.

[6] Barlas, G. 2015. *Multicore and GPU Programming.* Morgan Kaufmann.

[7] Brooks, F. 1975. *The Mythical Man-Month.* Addison-Wesley.

[8] Cesaria, C. 2018. Computing for Social Good. *Stony Brook University Magazine.* (Dec. 2018). Stony Brook University. Available: *https://www.stonybrook.edu/magazine/2018-winter/computing-for-social-good*.

[9] Cirulis, A. 2019. XXX Interceptor. *The Classic.* (Dec. 2019). Northwestern College. Available: *http://classic.nwciowa.edu/winter-2019/standout/holm*.

[10] Code for the Kingdom: 2016. Available: *https://codeforthekingdom.org/*.

[11] Cohoon, J.P. and Davidson, J.W. 2002. *C++ Program Design, 3rd edition.* McGraw-Hill.

[12] Computing for Social Good: 2020. Available: *https://ischool.illinois.edu/research/areas/computing-social-good*.

[13] Croll, A. 2012. Big data is our generation's civil rights issue, and we don't know it. *O'Reilly Radar.* (Aug. 2012). Available: *http://radar.oreilly.com/2012/08/big-data-is-our-generations-civil-rights-issue-and-we-dont-know-it.html*.

[14] CS 93 - Computing for Social Good: 2019. Available: *https://www.cs.swarthmore.edu/~kwebb/cs93/f19/*.

[15] CSE120 - Computing for Social Good: 2019. Available: *https://courses.cs.washington.edu/courses/cse120/19wi/lectures/25/CSE120-L25-good_19wi.pdf*.

[16] CS for Social Change: 2018. Available: *http://cssc.cs.brown.edu/*.

[17] Data Science for Social Good: 2015. Available: *https://escience.washington.edu/data-science-for-social-good-projects/*.

[18] Data & Society - About: 2020. Available: *https://datasociety.net/about/*.

[19] Data & Society - Research: 2020. Available: *https://datasociety.net/research/*.

[20] Gardner, M. 1970. Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*. 223, (Oct. 1970), 120–123. Springer Nature. Available: *https://www.scientificamerican.com/article/mathematical-games-1970-10/*.

[21] Goldweber, M. et al. 2019. Computing for the Social Good in Education. *ACM Inroads*. 10, 4 (Dec. 2019), 24–29. Association for Computing Machinery. Available: *https://dl.acm.org/doi/pdf/10.1145/3368206*.

[22] Gymrek, M. 2010. Conway's Game of Life. MIT.

[23] Heath, N. 2020. Want one of the best paid entry-level jobs? Learn this 20-year-old programming language. *ZDNet*. (Apr. 2020). Available: *https://www.zdnet.com/article/want-one-of-the-best-paid-entry-level-jobs-learn-this-20-year-old-programming-language/*.

[24] Holley, P. 2015. Driver follows GPS off demolished bridge, killing wife, police say. *The Washington Post*. (Mar. 2015). Available: *https://www.washingtonpost.com/news/morning-mix/wp/2015/03/31/driver-follows-gps-off-demolished-bridge-killing-wife-police-say/*.

[25] Hunt, J. 2010. Computing and the Cultural Mandate. *Dynamic Link - Christian Perspectives on Software Development*. 2 (2010), 6–8. Calvin College. Available: *https://computing.calvin.edu/documents/dynamic-link-journal.html*.

[26] JAARS - Technology: 2020. Available: *https://www.jaars.org/how/technology/*.

[27] Kirk, D.B. and Hwu, W.-m.W. 2017. *Programming Massively Parallel Processors, 3rd edition*. Morgan Kaufmann.

[28] Knuth, D. 2001. *Things a Computer Scientist Rarely Talks About*. CSLI.

[29] Liang, Y.D. 2018. *Introduction to Java, 11th Edition*. Pearson.

[30] Lowry, S. and MacPherson, G. 1988. A blot on the profession. *British Medical Journal*. 296, (Mar. 1988), 657–658. Available: *http://europepmc.org/backend/ptpmcrender.fcgi?accid=PMC2545288&blobtype=pdf*.

[31] Northwestern College Vision for Learning: 2020. Available: *https://www.nwciowa.edu/about/vision-for-learning/full*.

[32] O'Neil, C. 2017. *Weapons of Math Destruction*. Broadway Books.

[33] Pioneers: 2020. Available: *https://pioneers.org/*.

[34] Plantinga, H. 2002. Christianity and Computer Science at Calvin College. *Christianity and Computing*. (2002). Calvin College. Available: *https://cs.calvin.edu/static/documents/christian/plantinga.htm*.

[35] Savitch, W. 2018. *Java - an Introduction to Problem Solving and Programming, 8th Edition*. Pearson.

[36] Sazo, T. 2017. Coding for the Kingdom. *The Gospel Coalition*. (Feb. 2017). The Gospel Coalition. Available: *https://www.thegospelcoalition.org/article/coding-for-the-kingdom/*.

[37] Schuurman, D. 2017. Technology and the Biblical Story. *Pro Rege*. 46, 1 (2017), 4–11. Dordt College. Available: *https://digitalcollections.dordt.edu/pro_rege/vol46/iss1/2/*.

[38] Scott, M.L. 2016. *Programming Language Pragmatics, 4th Edition*. Morgan Kaufmann.

[39] Stanford CS + Social Good: 2019. Available: *https://www.cs4good.org/*.

[40] Steed, R. 2007. Programming and the Kingdom of God. *Dynamic Link - Christian Perspectives on Software Development*. 1 (2007), 12–13. Calvin College. Available: *https://computing.calvin.edu/documents/dynamic-link-journal.html*.

[41] Swamidass, S.J. 2019. Evangelicals Take On Artificial Intelligence. *Christianity and Computing*. (May 2019). The Wall Street Journal. Available: *https://www.wsj.com/articles/evangelicals-take-on-artificial-intelligence-11557442994*.

[42] Sweeney, L. 2013. Discrimination in Online Ad Delivery. (Jan. 2013). Elsevier. Available: *https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2208240*.

[43] The 100 largest companies in the world by market capitalization in 2020: 2020. Available: *https://www.statista.com/statistics/263264/top-companies-in-the-world-by-market-capitalization/*.

[44] The Ethics and Religious Liberty Commission 2019. Artificial Intelligence - An Evangelical Statement of Principles. (Apr. 2019). Southern Baptist Convention. Available: *https://erlc.com/resource-library/statements/artificial-intelligence-an-evangelical-statement-of-principles/*.

[45] Vander Linden, K. 2002. Computing in the Image of God. *Christianity and Computing*. (2002). Calvin College. Available: *https://computing.calvin.edu/documents/christianity-and-computing.html*.

[46] Warford, J.S. 2005. *Computer Systems, 3rd edition*. Jones and Bartlett.

[47] Wilkin, R.L. 2003. *The Spirit of Early Christian Thought*. Yale University Press.

[48] Wycliffe: 2020. Available: *https://www.wycliffe.org/*.

[49] *The Holy Bible, New Revised Standard Version*.