Computing Technology: Created, Fallen, In Need Of Redemption?

Joel C. Adams
Department of Computer Science
Calvin College

Abstract

If we accept the premise that computing technology is a part of the created order, then a variety of creation-related biblical themes become applicable to computing technology.
In this paper, we explore some of the implications of these themes for computing technology, particularly the themes of creation-fall-redemption, the cultural mandate, and stewardship. We also explore two developments in computing technology – the evolution of user interfaces and the evolution of programming languages – in the light of these themes.

1 Introduction

God has provided us with (at least) two sources of information about Himself:

- The Bible, also known as God's *special revelation*, and
- His creation, also known as God's *general revelation*.

In this paper, we explore different aspects of computation as revealed in God's revelations. In section 2, we begin with an argument that computing is a part of God's creation. In section 3, we apply various creation-related biblical principles to the endeavor of computing, and illustrate their application to operating systems and programming languages. Section 4 presents our conclusions.

2 Computation and God's General Revelation

Boyle, Keppler, Maxwell, Newton, and Pascal were just a few of the many scientists motivated by *theism*, believing that they could gain a better knowledge of their Creator by discovering the laws that underlie His creation. Christian physicists, chemists, geologists, and biologists may thus seek to understand the workings of different aspects of God's creation, by discovering the laws the govern them. Can Christian computer scientists make a similar claim?

For those who view the computer as a mere instrument, the phrase 'Computer Science' may appear to be a misnomer: there is no 'Microscope Science' or 'Telescope Science' or any other science named after an instrument. Why is there a *Computer Science*?

The root of the issue lies in the term 'computer', which has several definitions. A computer that sits on one's desk is indeed an instrument. This leads many people to mistakenly believe that Computer Science is the study of such instruments, and thus not a proper science at all.

Computer scientists, however, study a different kind of computer. To the computer scientist, *a computer is an abstract model of computation*, such as the *finite state machine* or Alan Turing's *Turing machine*. Years prior to the existence of physical computers, Turing and others used these models to discover problems that were impossible to solve using any computer, whether it be an abstract model or a physical system.

Interestingly, making minor changes to some of these unsolveable problems produced solveable problems. With the identification of each change that made an uncomputable problem computable, another *point* was identified separating computability from uncomputability in the problem space. With sufficiently many of these points, a *boundary* began to take shape. This boundary is an impermeable barrier between what can and what cannot be computed. As such, it has the force of a *law* with respect to computational systems.

The work of the computer scientist is to add to our knowledge about this and other computational laws. Like a theoretical physicist, a computer scientist constructs a mathematical model of reality, and then explores what is possible and/or impossible within that model. Actions that prove to be impossible within the model are impossible to implement on any actual hardware or software system to which the model applies; and actions that prove to be possible within the model can be implemented on actual systems.

Here is an anecdote to make this abstract discussion more concrete: When the author was a graduate student at the University of Pittsburgh, that university's Information Technology group decided that too much of the campus mainframe's CPU time was being wasted by freshman programs that contained infinite loops. The IT group decided to address this problem by writing a software preprocessor that, when a freshman submitted their program to be run, would first scan the program, determine whether or not it contained an infinite loop, and only run the program if it did not contain an infinite loop. The IT group spent six person-months trying to solve this problem and made no headway. They eventually decided to consult a faculty member from the Department of Computer Science. The CS faculty member they contacted immediately recognized that the problem they were trying to solve was a restatement of a problem that had been proven to be uncomputable fifty years earlier. In computer science circles, this problem is called the *Halting Problem*, and no hardware or software system that will ever be built will be able to solve it. This 'computational law' implied that no hardware or software system would ever be able to determine whether or not a freshman program contained an infinite loop. If the Pittsburgh IT department had known of this 'computational law' and understood its application to their problem, they would not have wasted six person-months in a futile effort to circumvent it.

In this respect — *because it has discoverable laws that cannot be violated in the real world* — Computer Science is a science.

2.1. Did God Create Computation?

If computer science *is* a science (as this author believes it to be), then it does not necessarily follow that computation is part of God's creation. One might argue that computation *transcends* God's creation: that its laws existed before God's creative act and that God Himself is subject to the laws of computation.

Christianity has long held to the doctrine that God's power is infinite. For example, in response to the question *What is God?* the Westminster shorter catechism replies:

God is a spirit: infinite, eternal, and unchangeable in his being, wisdom, power, holiness, justice, goodness, and truth.

It is impossible for our finite minds to fully comprehend the concept of infinity. However it seems to us that if God is indeed infinite in power, then his power cannot be limited by the laws of computation, any more than Jesus' power to walk on water was limited by the law of gravity. So if we agree that God is infinitely powerful, then we believe that the laws of computation do not limit his power.

So the laws of computation do not limit the activities of an infinitely powerful God, but they do limit the activities of the creatures in His creation. This suggests (but in no way proves) that the laws of computation are a part of His creation.

For the rest of this paper, we will suppose that computation as we know it *is* a part of God's creation. Then God has established the boundaries between what can and cannot be computed. At such boundaries, God is in effect saying to us, "You may compute this far but not further." As we discover such boundaries, more of God's decisions (and henceforth nature) as the Creator of computing become known to us.

2.2. Computation and Computing Technology

Let us define *computing technology* to be real-world hardware and software systems (constructed by computer and software engineers, respectively), as opposed to the abstract *computation* studied by computer scientists via their computational models. Since people build them, real-world systems are the creation of mankind. Since people are a creation of God, and computing technologies are a creation of people, computing technologies are also a part of God's creation.

There is thus a distinction between a *computer scientist* who explores the boundaries of computability, and a *computer engineer* who builds hardware systems or a *software engineer* who builds software systems. Like Boyle, Keppler, Maxwell, Newton, and Pascal, Christian computer scientists can discover new information about their Creator by studying His creation. Christian engineers are using the creative natures that are theirs as image-bearers of God. It is thus our position that computation (studied by computer scientists) and computing technology (created by engineers) are different, but that both are parts of God's creation.

The discussion in the next section applies to both computation and computing technology. However the phrase 'computation and computing technology' is tedious to read and write very many times. We will henceforth use the term *computing* as a collective shorthand for these different aspects of God's creation.

3 Computing and God’s Special Revelation

God’s special revelation does not mention computing. However it does say a great deal about His creation, of which we have argued computing is a part. If computing is indeed a part of God’s creation, then

1. Biblical teachings about God’s creation can inform our understanding of computing; and
2. A Christian can — in loving obedience to God’s commands — embrace work as a computational scientist or engineer.

In this section, we examine a few biblical principles and how they can be applied to computing.

3.1. *Created*: Computing should serve humanity

In Genesis 1:28-30, God gives the people He has created a mandate to “subdue” and “have dominion over” His creation. This passage is known as the *cultural mandate*, and it establishes the basic relationship between people and the creation: people are to “have dominion over” the creation.

If computing is indeed a part of God’s creation, then this passage establishes that people are to “have dominion over” computing. This implies that in the human-computer relationship, the humans are to work at being in control of the computers.

The relationship between a computer and the average user is largely determined by the *human-computer interface* (HCI) presented by the machine, which is a part of its operating system. In many (most?) operating systems the proper human-computer relationship is inverted: a person must invest in significant training to acquire special machine-oriented skills and knowledge before they can use the machine. As a result of this requirement, the machine effectively dictates the terms of a person’s interaction with it, and so dominates the person.

Virtually all operating systems’ HCIs require some special knowledge of their user, and so the proper human-computer relationship is skewed to a greater or lesser extent in nearly all computers. The degree to which the relationship is skewed in a particular operating system is proportional to the amount of special knowledge that operating system requires. Operating systems that minimize the need for special knowledge thus enable their users to more easily fulfill God’s mandate.

It is interesting to note write these ideas in the light of trends observable today. Computing technologies such as the MacOS operating systems, Microsoft’s “plug-and-play” standard and Windows operating systems, and newer Linux environments like KDE and GNOME each reduce the amount of special knowledge required of a computer-user. The creation of these technologies thus represents progress towards achieving the human-computer relationship suggested in Genesis 1. This can be seen as a convergence of God’s general revelation (computing practice) with His special revelation (biblical principles).

3.2. *Created*: We are to be stewards of computing

In Genesis 2, we read that God placed Adam and Eve in the garden to care for it. In Psalm 24, we read that "The earth is the Lord's, and all that is within it." These passages and many of Jesus' parables indicate that we are but *stewards* of God's creation, meaning that God has put us here to care for what belongs to Him.

This doctrine of stewardship moderates the cultural mandate in the following way: 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*. If computing is indeed a part of the creation, then the cultural mandate implies that we are to be sovereign over computing; and stewardship implies that we are not to abuse computing.

The question of what constitutes "computational abuse" can make for interesting discussion. For example, do any of the following qualify as computational abuse?

- Creating or forwarding a computer virus?
- Using a computer to circumvent copyright law?
- Seeking to break encryption techniques?
- Overclocking a computer's CPU?
- Downloading a program to use your CPU's spare cycles (e.g., SETI@home, or distributed.net)?

Would any of your answers change if computational intelligence (as defined by the Turing Test) ever equals or exceeds human intelligence?

3.3. *Fallen*: Computing is in need of redemption.

The Bible also makes the following three points about God's creation:

1. Following his creative act in Genesis 1, God pronounced His creation "good."
2. Following humanity's original sin in Genesis 3, God cursed His creation.
3. God will eventually liberate His creation from His curse.

Put simply, the creation has been changed (for the worse) since God created it, and God will one day redeem it. This theme of the need for creation to be redeemed is a thread spanning the breadth of the Bible. Some of the places it is mentioned are the following passages:

- In Isaiah's 11th chapter (and his description of the "new heavens and earth" in chapters 55 and 65), he uses imagery that describes a created order quite different from the one we know: "*The wolf will lie down with the lamb, the leopard will lie down with the goat, the calf and the lion and the yearling together; and a little child will lead them. The cow will feed with the bear, their young will lie down together, and the lion will eat straw like the ox. The infant will play near the hole of the cobra, and the young child put his hand into the viper's nest. They will neither harm nor destroy in all my holy mountain, for the earth will be full of the knowledge of the Lord as the waters cover the sea.*"

- In Romans 8, Paul discusses this more generally, saying that the creation "*waits eagerly for the sons of God to be revealed*", "*was subjected to frustration*" and "*will be liberated from is bondage to decay and brought into the glorious freedom of the children of God*."

- In Revelation 21, John describes "*a new heaven and a new earth, for the first heaven and first earth had passed away*" and he relays to us that God Himself will dwell among His people and will "*wipe every tear from their eyes. There will be no more death or mourning or crying or pain, for the old order of things has passed away*."

So unless computation is somehow outside of the created order, it would seem that it too is "*cursed*" and "*subject to frustration*" and will in the new earth be "*liberated from its bondage to decay and brought into the glorious freedom of the children of God*". It would thus appear that computation can be better than it is at present — something that God would describe as "*good*."

This raises many questions: How will "*good*" computation be different from computing as we know it? Does the phrase "*liberation from bondage to decay*" mean that entropy will no longer exist so that physical systems will no longer deteriorate? Will chips no longer burn out? Will storage media no longer degenerate resulting in data loss? Will networks no longer lose data packets? What does the phrase "*subject to frustration*" mean when applied to computation? Will algorithms that have been proven optimal on this earth turn out to be in "*bondage to decay*" and sub-optimal in the new earth? Will limitations like the impossibility of solving the Halting Problem vanish? Are our very abilities to apply logic and prove theorems "*frustrated*" and in need of liberation? Such questions make for an interesting discussion (that is necessarily speculative for now).

To make such questions even more difficult to answer, Genesis 2 tells us that prior to God's cursing His creation, He "*took the man and put him in the Garden of Eden to work it and take care of it*." From this, we can infer that God's creation still required care and work on the part of its stewards, even when God saw it as "good". While the exact nature of this work and care is not detailed, the implication is that the creation's need for development and maintenance pre-dates God's curse. If the effect of redeeming creation is to remove its curse and return it to its original state, then it would appear that computing will still require development and maintenance following its redemption (although perhaps in some less "thorny" way than is currently required.)

3.4. *Fallen*: People are in need of redemption.

A further complication is that in addition to the creation being cursed, original sin has affected the people who use computers. Much of the theology in the Bible deals with how our spiritual state has been affected by original sin, as described in Genesis 2 and 3.

The Bible also teaches that original sin has affected people's physical (Genesis 3) and mental (Romans 1) states. This implies that the negative aspects of the human-computer relationship are not solely the result of flaws on the computing side, but also stem from flaws in human beings. Both humans and computers must be redeemed in order for the human-computer relationship to be what God would call "good."

Some negative aspects of computing clearly stem from humanity's flaws. Buggy software springs to mind, as does malicious software (viruses, worms, trojan horses, etc.). Perhaps in the new earth, people will not write (or at least release) buggy software. Perhaps viruses, worms, and the like will all be benign.

We suggested earlier that the specialized knowledge required by an operating system's HCI largely determined the degree to which a computer dominates its users. Perhaps this is too simple. Perhaps our degraded mental abilities also contribute to this problem. Perhaps after we have been redeemed, operating systems will still require specialized knowledge of a user, but it will require much less effort for the average person to acquire such knowledge.

3.5. *Redemption*: Building "good" systems

With his death and resurrection, Jesus began the redemption of ourselves and the creation. Christians are called to be *imitators* of Jesus: to continue his work of redemption and to work for the advancement of His kingdom until he comes again. Christians with computational expertise – whether scientists or engineers – can bring much-needed "salt and light" to computational research and practice. For example, as computing power increases, the temptation to surrender our responsibilities to technology will also increase. Christians with a strong ethical sense should be in the vanguard of those making such decisions.

Christian computational 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. Christian computational engineers can apply such knowledge in the construction of systems that improve on existing ones, with the goal of producing systems that God would think of as "good."

The history of computing shows marked improvement in many areas. Two of these areas are operating systems and programming languages. The history of these areas shows progress: we believe that to the extent they correct the flaws of their predecessors, the evolution of operating systems and programming languages represents progress towards systems and languages that God might call "good."

Operating Systems. Computers once cost a fortune and were extremely difficult to operate, so that only a select few were able to make use of them. Today, the average person can own and operate their own computer, thanks to an unprecedented decline in prices and vastly improved operating systems. The improvement to operating systems came as the result of a slow-but-steady series of changes.

The earliest operating systems were *resident monitor systems* that did little more than load programs and the translators and libraries necessary to process those programs. *Batch systems* improved computation by increasing the CPU utilization and decreasing the average turnaround time for programs. *Multiprogramming systems* improved computation by using memory more efficiently, improving CPU utilization further, and decreasing the average turnaround time even more. *Time-sharing systems* improved computation by allowing for interactive execution, making possible a host of new interactive applications. *Virtual memory systems* improved computation by making it possible to write and execute programs whose size exceeded the size of physical memory. The development of *microcomputer systems* democratized computing, by distributing the power of computing out of computer centers onto people's desktops. *Multitasking systems* allowed a person to work at one task on their computer at the same time as the computer processed another task. *Multithreaded systems* improved user interfaces, by letting applications separate I/O and processing. *Networking software* turned standalone number crunching machines into communication devices, allowing people to connect and communicate in ways never before possible. *Real-time systems* made it possible to process data in real time, bringing about multimedia applications that were undreamed of a decade earlier. The progress in operating systems from the early days until today has been nothing short of remarkable (though there remains much room for improvement).

Programming Languages. The history of programming languages provides a similar story of remarkable improvement. In early machines like ENIAC, a program was hard-wired into the machine, so that making the machine execute a new program effectively required the program portion of the machine to be *rewired* – a tedious task at best.

The next generation of machines permitted a programmer to write and load programs without rewiring the machine, but such programs had to be written in the *binary language* of the machine. Writing these programs was difficult and error-prone; debugging them was even more so.

Assembly languages were subsequently developed that allowed a programmer to write a textual *mnemonic* for a given binary statement, and write a *symbolic variable name* instead of a raw memory address. A program called an assembler then translated the mnemonics into their binary counterparts and the symbolic names into addresses. The 1-to-1 correspondence between assembly mnemonics and machine instructions produced two main drawbacks: (i) programs written for one kind of computer could be transferred to but not executed on a different kind of computer; and (ii) the logic of assembly programming reflected the low-level logic of the machine, not the high-level logic of a person.

These problems were reduced by the development of *high-level languages*, which allowed a programmer to write high-level statements using logic more like that of a person. A compiler could then translate such high-level statements into as many binary statements as were needed to implement that logic. Programs written in such languages could (at least in theory) be ported, translated, and run on any machine for which a compiler existed, and programming in such languages is far more natural than doing so in an assembly language.

One of the most difficult topics for high-level language students to master is *indirection*, or the use of *pointer variables*, which store memory addresses. As an address-storing variable, a pointer a relatively low-level construct, and many computer scientists view pointers as an unfortunate carry-over from assembly languages. Pointers are used to implement *dynamic data structures* that grow and shrink as a program executes, as well as elicit *polymorphic behavior* in object-oriented languages. Object-oriented languages like Smalltalk and Java use techniques like *garbage collection* to hide their pointers, thus making it easier for programmers to program in these languages.

Modern languages like ML and Haskell have been dubbed *higher-level* languages, because they have completely eliminated pointers from a programmer's view. In these languages, a programmer can define a dynamic data structure using a recursive definition, which the language's translator then converts into a machine representation using pointers. Such languages thus allow a human programmer to think and write their programs using human-level logical, and off-load the work of determining the low-level implementation of that logic from the human to the translator, which acts as an expert system for this task.

Just as the improvement in *operating systems* has made it possible for more people to become productive *users* of the computer, the improvements in *programming languages* have made it possible for more people to become *programmers*. Relatively few people were able to become proficient machine-language programmers; more were able to become proficient assembly-language programmers; still more were able to become proficient programmers using high-level languages; and still more can become proficient using object-oriented and higher-level languages.

The evolutions of operating systems and programming languages have thus permitted more people to attain differing levels of mastery over the computer. These changes reflect progress towards the 'good' human-computer relationship in which more humans "have dominion over" the computer.

4 Conclusion

Computing is a part of God's creation and is meant to serve humanity, though humanity is not free to abuse it. Both humanity and computing are fallen and in need of redemption, so that people tend to be dominated by computers. Christian computational scientists and engineers can seek to imitate Christ by working to redeem computing. Some ways in which this can be done include:

- Improving our knowledge of the laws of computing (i.e., computing research);
- Improving the computers that people use (i.e, computing engineering); and
- Improving the people who use computers (i.e, computing education).

While awaiting Christ's return, Christian computing professionals can serve as agents of redemption by conducting research into the laws of computing, by building computing systems that better serve people, and by teaching people to use computers more effectively.

As indicated earlier, the apocalyptic passages in the Bible indicate that all things will be *fully* redeemed only when Christ returns. This suggests that while we can work toward the goal of building systems that exhibit human-computer relationships God would call "good," we must be realistic and recognize that we will only be able to build systems that God would truly call "good" when Jesus returns. *Come quickly, Lord Jesus!*