Dynamic Link
Christian Perspectives on Software Development Issue 3 2011–2012

His Requirements

As this journal was being assembled, I added a part time job to my normal duties as a college professor. It's a brief engagement I accepted to keep in touch with industry and to keep my skills current, but I soon realized again why I have a passion for systems analysis: It is the direct engagement with someone who has a vested interest in the outcome.

"Vested interest" is only part of it. The initial interview with stakeholders started like most with a few questions for clarification, a few ideas on the final outcome, and then it happened as it often does. After a question was answered, I offered my usual assertion as a question about a requirement: is that always true? That moment with a person who would actually use the software turned into a period of exhilaration as an exception (requirement) was discussed. A fresh bolt of energy entered the conversation.

"Exhilaration?" A "bolt of energy?" Both may sound too strong in describing what can happen in a business meeting, but I would be willing to offer even stronger terms. Why? It's because this is the moment that an analyst lives for. It is the moment where partners make a discovery. Somehow, after being buried under mounds of initial requirements, the back and forth discussion was the spark behind the combustion that blasted away the weighty assumptions and perceptions that buried an insight, a jewel, that was in need of being found: the truth.

I hope we can discover "the truth" in discussions and articles about the Christian faith as it applies to software development. This journal marks the third time where people have unequivocally said that their faith is part of what they do when developing software, whether that role is as the analyst, software engineer, project manager or an expert user.

Dr. Joel Adams discusses the stewardship opportunities in developing code to leverage green technology. Dr. Steven VanderLeest explores the need for Christian engineers to have a source of devotions about their craft. Dr. Victor Norman reflects on the value of beautiful code as an extension of his faith and service to God, and this is complemented by Mr. Bruce Abernethy's reflection on our need to create as a reflection of His image. Mrs. Dorinda Beeley talks about how she answered the call to bring the truth to many through her work of supporting information technology for missionary organizations.

Finally, this journal closes with a summary of the Dynamic Link 2011 Conference. This conference was designed to bring students and working professionals together for a day of discussions about the Christian perspective on areas of software development. In a sense, these thoughtful discussions are held to understand His requirements for "the truth."

I believe the articles in this journal and the conference are opportunities where the participants acknowledge that God is at the center of everything we do to include the software systems we create.

Patrick M. Bailey, MS
Associate Professor
Calvin College Computer Science Department

Table of Contents

CALVIN
College

Department of Computer Science
http://www.cs.calvin.edu

Carbon Footprints and Computing: Efficiency as Stewardship of God's Creation

Joel C. Adams, Ph.D.
Department of Computer Science
Calvin College

Introduction

In Genesis 1:26, God told the first humans to "be fruitful, multiply, and subdue the earth." This "subdue the earth" phrase, in conjunction with other scriptural passages (e.g., Psalm 8: 5-8), provides humanity with the authority to make use of God's creation. However, since it is God's creation (e.g., Psalm 24:1, 1 Corinthians 10:26), humans are not licensed to abuse the creation. Instead, we are to act as caretakers or *stewards* of God's creation.

In most parts of our world, electricity is generated by the combustion of fossil fuels, usually coal or natural gas. Fossil fuels are carbon-based and combustion is an oxidation process, so this combustion produces carbon dioxide (CO2) gas. When CO2 is released into the atmosphere, it traps heat, which is generally thought to be a Bad Thing. This leads to the notion of *carbon footprint*: the amount of carbon—formerly sequestered in the fossil fuel—that the use of a given device or process releases.

Computers and Carbon Footprints

Computers, regardless of whether they are desktops, laptops, tablets, or smart phones, are powered by electricity. As devices become increasingly mobile and powered by batteries for long periods of time, computer manufacturers are increasingly sensitive to the power consumption and carbon footprints of their devices, to the point that some devices are now "smart" enough to shut down idle components, ranging from storage devices to particular circuits within the computer's central processing unit (CPU).

If we think about how computing devices consume electricity, it should be evident that a given device consumes differing amounts of power at different times. At any given time, the device occupies one of the positions shown on the continuum below:

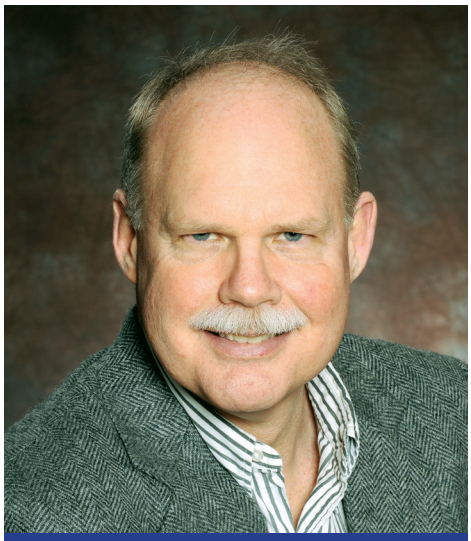
(A) If the computer is off and disconnected from a power source, it is consuming no electricity, putting it at one end of the continuum.

(B) If the computer is off but it is connected to a power source, it consumes a trickle of electricity to keep its battery charged, its internal clock running, etc.

(C) If the computer is on but is in sleep mode, it consumes a somewhat larger amount of electricity to maintain the memory states of whatever programs have been launched (i.e., the operating system, at the very least).

(D) If the computer is on and not in sleep mode, but has no user programs running (i.e., it is idle), it consumes a much larger amount of electricity than when it is sleeping, moving it much further down the continuum.

(E) As a user launches programs on the computer, those programs use more and more of the computer's resources, consuming energy. On average, each additional program launched moves the computer further down our continuum.



Joel Adams

(F) If a computer has sufficiently many programs running that all of its devices—CPU, memory, storage units, network adaptors, etc.—are continuously busy, it consumes a maximal level of electricity, placing it at the opposite end of our continuum from (A).

From this, we might be tempted to conclude that the amount of electricity used by a running, non-sleeping computer depends on the number of programs that are currently running on it, and this may well be the case, on average. However, the *behavior* of the running programs is even more important than their number. That is, one resource-intensive program can keep all of a computer's components in continuous use, and thus consume the maximal level of electricity; while a dozen simple programs all doing nothing but waiting for the user to interact with them would consume a lower level of electricity. The position of a computer on our continuum thus depends mainly on the *behavior* of the programs it is running.

It follows that the carbon footprint of a given computer can vary from zero when it is at one end of our continuum (i.e., off and disconnected from power) to some maximal amount when it is at the other end of our continuum (i.e., on, and running programs whose behavior keeps its devices continuously busy).

Computer Programs and Carbon Footprints

Since the carbon footprint of a computer depends on the behavior of the programs running on it, let us turn our attention to those programs.

Prior to 2006, most computers had just one processing "core" in their CPU chips. Such chips could perform just one action at a time, so clever computer scientists devised operating systems that would "time-share" that core among multiple programs, like people time-sharing a waterfront cottage. The speed at which the computer performed this time-sharing created the illusion that all of the programs were running simultaneously.

In those days, CPU use was a zero-sum game: every CPU cycle consumed by one program was a cycle that was unavailable to another program, making it important that programs be as time-efficient as possible. Since a program is only as efficient as its underlying algorithm, and algorithms are independent of implementation details like programming language and hardware platform, algorithm efficiency received a great deal of attention. Computer scientists devoted much time and effort to crafting algorithms and data structures that could be used to solve common problems efficiently, and they devised formalisms like "big-oh notation" to measure and compare their efficiency (See the insert for a short explanation of big-oh).

To illustrate, consider the problem of sorting a list of n values into ascending order. Many different algorithms have been devised that solve this problem correctly. Some algorithms can solve the problem in $O(n^*lg(n))$ time-steps, while other algorithms take $O(n^2)$ time-steps to solve it. Since $n^*lg(n) < n^2$ for positive values of n, the algorithms that solve the problem in $O(n^*lg(n))$ time-steps are more time-efficient (i.e., faster) than those that solve it in $O(n^2)$ time-steps.

Or, consider the problem of locating a particular item within a sorted list of n items. We could solve the problem using an $O(n)$ algorithm called sequential search, or we could solve it using an $O(lg(n))$ algorithm called binary search. Since $lg(n) < n$ for positive values of n, binary search is the better choice.

> From a practical perspective, a program's design can greatly affect its energy consumption, and hence its carbon footprint.

Big-oh notation thus provides us with a convenient way to talk about the time-efficiencies of different algorithms for the same problem, and to estimate how efficient an algorithm is. A program that uses an $O(n^2)$ algorithm to sort a list of n values is inefficient because it uses more steps to solve the sorting problem than are necessary—we know that more time-efficient algorithms exist. In general, if the best possible algorithm A1 for a problem requires time $O(f(n))$, and we instead use an algorithm A2 for that problem that requires time $O(g(n))$ where $f(n) < g(n)$, we are not solving the problem as efficiently as we might.

Since a computer program is just the expression of an algorithm in a programming language, the same big-oh notation that describes an algorithm's efficiency can be used to describe the efficiency of a program that implements that algorithm.

Big-oh time-efficiency can also be used to compare two programs' carbon footprints. That is, if we have two programs P1 and P2 that solve the same problem, P1 solves it in time $O(f(n))$, P2 solves it in time $O(g(n))$, and $f(n) < g(n)$, then program P1 solves the problem faster than P2, using less electricity than P2, and with a smaller carbon footprint than P2, making it preferable from a creation-stewardship point of view.

From a theoretical perspective, a program that solves a problem using an optimal algorithm should have a minimal carbon footprint, since it using a minimal number of time-steps and hence a minimal amount of electricity. By contrast, a program that solves a problem using a non-optimal algorithm will have a larger-than-necessary carbon footprint, making it less desirable from a creation-stewardship point of view.

From a practical perspective, a program's design can greatly affect its energy consumption, and hence its carbon footprint. Some simple examples include:

- Mobile devices like smart phones and tablets provide value through apps for services that let their users communicate and stay "connected". However, the design of an app can greatly affect its energy consumption (and hence the battery-life of the mobile device). Services designed to minimize communication traffic generally use less energy and have a lower carbon footprint than functionally equivalent services that communicate continuously or in regular traffic bursts, especially across cellular networks.

- On a multicore computer, a program that uses a parallel algorithm (i.e., one that makes use of all of the processor's cores) should solve its problem more

quickly than a program that solves it using a sequential algorithm (i.e., one that uses just one of the processor's cores). If it can solve the same problem in less time, then the parallel program may use less electricity and have a smaller carbon footprint than its sequential counterpart.

These are just two of many ways in which program design can affect a device's energy consumption; space limitations prevent us from presenting more.

A program's carbon footprint thus depends on the algorithm it uses to solve its problem and how efficiently its design uses the underlying hardware.

Conclusion

We have seen that computing devices have carbon footprints, whether they are desktops, laptops, tables, or smart phones. We have also seen that the carbon footprint of any particular device can vary along a continuum, and that its position on that continuum at any given time depends on the behavior of the programs that are running on it at that time. Finally, we have seen that a program's design can affect its carbon footprint, and that a program's big-oh time-complexity provides a means of comparing the carbon footprints of similar programs.

In a world in which energy consumption equates to the release of carbon into the atmosphere, minimizing programs' carbon footprints is a worthy goal for computer scientists and software engineers, especially for Christians seeking to be good stewards of God's creation.

Joel C. Adams, PhD (adams@calvin.edu), is professor and chair of the Department of Computer Science at Calvin College. He has twice been named a Fulbright Scholar, has designed and built several Beowulf clusters, and has authored numerous books and technical articles. He also enjoys reading, watching movies, playing a 5-string fretless bass, and watching his sons play soccer.

Haven't Heard of Big-Oh?

If the comments about big-oh are new to you, hopefully this explanation will help you understand. Big-oh is a notation used to approximate the upper bound of the scale of work needed to accomplish a task. Let's say that you are part of a group of people, and you (alone) have to greet each person. We refer to the size of the group with the letter "n" (number). Since you wouldn't greet yourself, you will greet the total size of the group less yourself. That would be exactly "n minus one" or n-1. Now when n becomes very large, that "minus one" becomes negligible, so big-oh drops that term and describes the scale of the work as O(n) (It starts with a capital "O", and it's short for "on the order of" which is why we call it big-oh!) That is the order of how much work you must do to accomplish the task—about n units of work.

Now imagine someone said that everyone in the group must take a turn and give their business card to each member of the group, and it must be done one at a time. Let's work with a small number for this example—three people in the group (i.e. n is three). The first person gives his business card to the second and third person. Then the second person gives her card to the first and third person. Finally, the third person gives her card to the first and second person. (This is very close to how things work in a computer program—one thing at a time!) Basically each person (3 or n) had to visit the other two (the original 3 less him or herself or n-1). So, if everyone followed the instructions, there were six specific moments where someone gave someone else a card which is the same as saying three times two. In this example, if we substitute the number three with "n" the exact amount of work done is "n multiplied by n minus 1" or $n(n-1) = n^2-n$. When n is very large, the "–n" term is insignificant compared to the n^2 term, so big-oh drops the "-n" and describes the scale of the work as $O(n^2)$.

Technology Devotion:
Why Christian Engineers and Scientists need a Devotional Life

Steven VanderLeest, Ph.D.
Department of Engineering
Calvin College

Peculiar Pursuit

It is a rather unusual calling: writing devotions for Christians working with technology.

As a Professor of Engineering at Calvin College, I regularly start class with devotions. At a Christian college, this is not so unusual, though even here it is perhaps remarkable to do so in a technical class such as Introductory Electronics or Computer Architecture. I hope my classes encourage students to develop not only technical ability but also spiritual discipline. Furthermore, I hope to integrate the two so they are not simply two worthy pursuits laid side-by-side, but a whole that is deeper and richer than the parts imply.

Is there an audience?

In a hallway discussion with a colleague, we each noted that we had developed a collection of topics we regularly used for devotions in our engineering courses. This led to work on a book.

We have approached a number of publishers, many of whom liked the idea, but questioned the market appeal. They were doubtful that many scientists or engineers would be interested in reading religious material about their discipline. They suspected that most people working with technology did not see any application to their faith (or vice-versa).

This is our point and also our conundrum—our book would challenge and encourage Christians working with technology to connect their faith with their occupation. Will we be able to stimulate interest in an audience that doesn't appear to recognize this need?

In this article, I hope to argue for the necessity of connecting these aspects of our lives and the benefits of taking this approach.

Why would publishers suspect there is no need? Perhaps most technologists are atheists. No self-respecting scientist or engineer would admit to religious aspects of reality, at least not publically. At most religion is a private, individual affair, but it has no place in the objective world of technical development. Perhaps most Christians are anti-science. No true believer would admit that science holds any ultimate truth, at least not publically. At most, science has some value for certain occupations, but it has no standing to make claims about the ultimate origins or purpose of life and God's creation.

I reject both propositions. Many scientists and engineers are Christians. Many Christians work in technology areas. While many of us hold faith convictions, we may in practice tend to separate out our work by going about our daily business with little thought about how God fits in. Meanwhile, in theory, we believe God rules over all—including our work.

> I hope my classes encourage students to develop not only technical ability but also spiritual discipline.

Even if we grant that one could simultaneously be a Christian and an engineer (or scientist or computer scientist), we could still hold them as separate, isolated roles. Not only might one consider religion to be a private affair, but in the US, one could also point to the enshrined separation of church and state. Here too I beg to differ: separation perhaps, but not exclusion. Not isolation. I believe Christianity has something distinctly helpful to say about technology development. Our worldview and values shape the culture and society around us. Christian faith provides important normative guidelines that can shape technology.

Steven VanderLeest

Technology is no exception to God's rule

Christ's rule extends over every square centimeter of the creation. Theoretically we believe this. In practice, we tend to give technology and science an exemption to God's sovereignty, not intentionally but by omission. It doesn't occur to us that God might have something to say about our work. It may be that our strong dependence on science and math has lulled us into thinking science is the ultimate objectivity. The seductive self-interest of the scientific method has convinced us of its claim to be the ultimate arbiter of truth.

Self-interest? What about the vaunted objectivity of science? The scientific method may proceed by virtue of a disinterested experimenter, but science itself claims to be master of its domain. Further, it tends to over-reach, claiming that one can best understand reality through science and that what is real is defined

by what can be measured by science.

Technical folks are particularly susceptible to the problem of compartmentalization because we do it for a living. Good programming practice includes abstraction and modular design. In order to understand and control, we break down big problems into step-by-step solutions. We avoid control and data coupling across large, complex systems because the interactions are difficult to predict and bound.

While a divide and conquer approach to technical problems is an effective tool to deal with complexity, this method can also lead to problems. When we examine the parts in isolation, we miss essential behaviors and interactions. Compartmentalization of life and reality itself misses the essential and foundational characteristic of God's sovereignty. This is the same sin as boxing up worship for Sunday only, leaving faith behind on Monday morning.

Bridging the gap between technological work and faithful worship

Technology and faith are not mutually exclusive and putting them together produces an interesting fabric. Because the connections are not always obvious, I write my devotionals to help tease out some threads that demonstrate the relationship. This section enumerates a few aspects of the interplay between the two.

Faith guides technology

Faith convictions ought to guide our technological pursuits. Consider a couple examples.

First, God made us with the ability to develop technology. Technology is part of what makes us human. We were created to create—as much *homo faber* (man the maker) as *homo sapiens* (man the wise). Part of the *imago dei* is the human ability of creativity. We reflect the creator when we invent, design, and develop. We were also created to steward. Part of the call to stewardship is to care for and cultivate the creation. **As stewards, we unwrap the gift of creation by thoughtful development of culture and society.**

While preserving creation is also part of that call, this does not mean keeping it in a static, so-called pristine condition that shows no mark of humankind. That would be like burying the one talent without investing it (like the "wicked, lazy" servant of Matthew 25:26). Instead, we should help the creation to flourish, nurturing new development and growth while protecting beauty and grandeur. Creativity is thus a tool of stewardship.

A second example of faith guiding our technical work is recognition of our responsibilities related to the technology we develop. The scientific patina of our technology misleads us into thinking technology is objective, unbiased, and neutral. It is not. The technology we develop reflects the values and desires of the human creators, even when the designers intended to be objective. Technology is always a means to an end. The problems we choose to solve and the tools we develop as solutions have biases—at the very least towards the goals we had explicitly in mind, but additional biases also sneak in without our conscious intent.

Technology and faith are not mutually exclusive and putting them together produces an interesting fabric.

Technology itself does not have moral agency. Consider a hammer developed for pounding nails but used by a criminal to strike and kill a victim. The hammer is not responsible for murder—only the criminal is accountable. But most cases are not so simple. Technology embodies responsibility and bias. Consider a medicine that is developed for curing disease by a manufacturer using careless methods and unsanitary conditions, resulting in deaths. The medicine is not responsible for murder—but where is the wielder of the weapon? Who is responsible? Is it the manufacturer (and possibly others in the "chain of custody" of this product)? Responsibility traces a thread from user to seller to manufacturer to designer. Each bears some accountability for the results produced by technology. Carelessness, negligence, cutting corners, or failure to recognize consequences can all result in harm from technology. While technology has no agency—it cannot act on its own—it embodies the volition of its creator and user. The harm or good that technology produces is a telltale sign of this bias. That bias implies our responsibility.

As Christians in technology, the call to care for our neighbor implies that we take responsibility for our products seriously. Technology is a powerful tool that amplifies human vice: a human with a gun is much more dangerous and likely to kill than one without. Technology also amplifies human virtue: a human with a telescope can see farther and is much more likely to develop insights about distant space objects and appreciate the astronomical scale of God's creation. Each time we develop new software or a new device we let a powerful genie out of the bottle. Christians in particular should pause to reflect before releasing the genie. Pause to feel the weight of responsibility to cultivate creation through technology development; pause to feel the weight of the call to care for our neighbor.

The best technological designs can glorify God and serve his kingdom by demonstrating responsible and appropriate design practices through the embodiment of virtues such as love, caring, humility, justice, mercy, and stewardship. We need these virtues in full measure because our tools also raise some of the most vexing ethical questions in society today. In the last century we have developed multiple

technologies that can wreak global destruction such as nuclear or biological weapons, or even grey goo (the nanotechnology doomsday scenario first envisioned by Eric Drexler). We are temptingly close to creating life through genetic cloning, artificial intelligence, or downloading our brains into machines. These technologies pose deep philosophical questions that Christians need to tackle with scriptural principles to guide our thinking. The power of the technology we create stands in stark contrast to the fallible and frail humans that created and use the technology. The virtue of humility curbs the hubris technology so easily promotes.

Technology enlivens faith

While our faith ought to guide our development and use of technology, the reverse is also true. Technology can guide development of our faith, helping us understand and worship our God.

Because tool-making is in our blood, we tend to build mental models of the world through technical metaphors. Think about all the shorthand phrases we use daily to represent complex ideas through technical analogy: pushing his buttons, turning the crank, driving her crazy, or I'm just a cog in the wheel. Why shouldn't we use technical metaphors to help us understand and discuss our faith too?

Scripture itself uses the imagery of potter and clay to describe God's sovereignty. The metaphor of tools applies to us: we are not only the tool-maker, we are also the tool. The computer produces garbage out from garbage in, providing a metaphor for the damage we suffer from spiritual debris that clutters our lives.

> The best technological designs can glorify God and serve his kingdom by demonstrating responsible and appropriate design practices through the embodiment of virtues such as love, caring, humility, justice, mercy, and stewardship.

Technology done well can help us worship God. It gives us devices to understand creation better. It gives us tools to cultivate the natural resources God gives us. Technology done well can help us serve God by serving our neighbor. It gives us devices to heal wounds. It gives us tools to help humans and all creation to flourish.

Epilogue

My colleague and I are continuing work on a book of devotions. In order to provide evidence to publishers that there is indeed an audience, I have also been writing a blog on the connection between technology and Christian faith, called *Deus Ex Machina* (God in the Machine). Every couple weeks I post a new devotional, you can find them on *http://www.calvin.edu/weblogs/deusexmachina/*.

Steve VanderLeest is a Professor of Engineering at Calvin College, Vice-President of R&D at DornerWorks (an embedded systems engineering company), and a partner at squishLogic (an iPhone app development company). His publications span technical areas such as computer performance measurement and safety-critical design methods, technology philosophy topics such as technological justice and responsible technology, and technology education topics such as design and entrepreneurship.

Code's Creative Spirit

Bruce Abernethy
Senior Architect, CQL Inc.

This summer I walked past a boy who was busy drawing in a sketchbook with his friend beside him—he was clearly quite involved with his current effort. His friend saw me, smiled and waved me over as if to say "Take a look at this."

I started over to meet these total strangers. As I got closer the friend said something to the effect of "He just saw this scene a few minutes ago and is busy drawing it out." I am no art critic, but the picture that was unfolding using just a basic pencil on a wire-bound notebook looked as if a black and white photograph had been taken—but more than that. He also captured the dynamics and motion, and even emotion, of the scene. I believe that he has no memory of this chance meeting on a summer day, but this amazing display of skill and creativity has had quite an impact on me ever since.

What is it about engaging in the act of creation that is so compelling to people everywhere? Whether it is kids building sand castles or adults doing crafts or restoring old cars—people who are engaged in actively creating any kind of project are often the most happy and fulfilled people I have ever talked to. And that definitely includes people involved in the design, architecture and development of computer software.

Since I was quite young I was impacted by the reading and re-reading of the story of Creation, particularly the idea that how we as humans were created intentionally by God and in the "Image of God". Among the many implications of this reality are that we take on many of the communicable aspects of God which includes wisdom, knowledge, goodness, freedom, grace, perfection, beauty, and, I believe, creativity. While God is ultimately the Creator and Sustainer of all things, we are important parts of this creation as we act as the makers and fashioners—the "hands and feet" if you will—of God's ongoing plan.

Bezalel and Oholiab—The Creative "Makers" of Judah

In Exodus 35 Moses told about when "[…] the LORD has called by name Bezalel the son of Uri, son of Hur, of the tribe of Judah; and he has filled him with the Spirit of God, with skill, with intelligence, with knowledge, and with all craftsmanship, to devise artistic designs, to work in gold and silver and bronze, in cutting stones for setting, and in carving wood, for work in every skilled craft." (ESV)

I believe that God does fill, equip and enable gifts and talents among his people. That includes the ability, creativity and motivation to complete great works. People sometimes ask why I still try to do some kind of programming, every day, after more than 30 years of "playing with" computers. 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 never existed before, work that is done well, and work that meets a need for other people. The Roman poet Juvenal talked about *cacoethes scribendi* which is loosely translated as the "insatiable desire to write"—I believe this is experienced today, by many inspired coders, designers and makers.

Attracted to Beauty and Good Design

Paul reminds us in Philippians 4:8 that "...whatever is true, whatever is honorable, whatever is just, whatever is pure, whatever is lovely, whatever is commendable, if there is any excellence, if there is anything worthy of praise, think about these things." (ESV)

Why are people attracted and excited by one piece of software or hardware over another one? Why do some people have a nearly religious attachment to their iPhone, love playing with the Wii, never want to give up their TiVo, never leave home without their BlackBerry, and have hundreds of lifetime hours into Tetris? Why was the world-wide web the app that made the Internet a must-have utility? Why do I still have a Macintosh SE/30 and Newton on a prominent shelf in the basement and make a point to "light them up" a few times a year to make sure they are still OK? In a nutshell it is because these devices and services seem to have something inherently "right" or good about them—they are excellent at what they are created to do.

I believe that God does fill, equip and enable gifts and talents among his people.

Beautiful Code

Software developers see this in well written code. When you have written enough code you develop a sense when you see some really excellent or even elegant code segments. When one of these segments or algorithms is good enough that it needs to be remembered, it is often given a description or name such as a "binary search", "bubble sort" or "traveling salesman" and others. One of the great benefits of the Open Source movement is that it has made the source code for a number of very well written and large scale software systems available for anyone to review and use. While non-programmers may think it is an odd pastime, the regular "pleasure reading" of a variety of different Open Source projects can be educational and inspiring. It is interesting that you never really know which of this code is written by Christians or not, but common grace enables wonderful contributions to all who are created in the image of God.

To software "artisans" who take the time to learn, practice, and understand the craft of software development, there is an awesome beauty and order in a solution that is done well.

Inspiration from Creation

Psalm 111:2 "Great are the works of the LORD, studied by all who delight in them."

Many of the great pieces of art and music are inspired by the Creation around us. I am reminded of the incredible power of recent videos of the Tsunamis, the beauty in the colors sunset, or awesome time-lapsed sequences of the International Space Station orbiting the Earth—all the best artists and computer generated animations cannot come close to the "real thing".

Recently there has been an interesting trend in hardware and software design to revisit "Natural User Interfaces" like motion, touch, gestures and voice. The popularity of the Nintendo Wii game console came largely as it enabled natural movements and full-body motion—enabling it to outshine and outsell competitors in the market with far superior graphics and sound. The "touch, swipe, and pinch" interface of the iPad have attracted nearly 30 million users to use a tablet after years of disappointing adoption of devices that required a keyboard or stylus to complete many operations. Now the Microsoft Kinect controller for Xbox and personal computers is enabling complex interactions between people and machines that were never before possible using even more natural motions, gestures and voice commands. Humans are naturally compelled and pleased when they can interact with hardware and software in a way that seems natural to them. Drawing inspiration from Creation can enable people to do far more things, far more quickly and easily, than they ever could before.

Share with Others

Another very rewarding part of being a software developer is the opportunity to give back and participate in the larger community of software developers. Residents of West Michigan can attest to the amazing phenomenon that has come with having the ArtPrize competition these past years with an amazing array of art and visitors from around the world. Perhaps more often heard (especially from kids) is something like "That doesn't look too hard—I could even do that …" Exactly! Great work inspires others to do great things.

While learning experiences for software developers can include classic training and national conferences, I believe there can be a richer experience in settings like user groups, "bar camps", and "open spaces" where all participants are encouraged (or required) to be both consumers and presenters.

We had a great experience attending the Maker Faire in Detroit this year. Nearly every kind of hardware, software, invention, art, performance, experience, or competition that is being worked on throughout the world was present at the exhibition. But more than that, the makers and inventers themselves were available to show and explain their works and in many cases attendees were able to get "hands on" with skills and technologies that they may not have experienced before—where else can an eight-year old be allowed or encouraged to learn to solder LEDs on to a lapel-pin that they can take with them when they go (and for only $1).

For Good Works—To the Glory of God

Continuing in Exodus 36, "Bezalel and Oholiab and every craftsman in whom the LORD has put skill and intelligence to know how to do any work in the construction of the sanctuary shall work in accordance with all that the LORD has commanded." And Moses called Bezalel and Oholiab and every craftsman in whose mind the LORD had put skill, everyone whose heart stirred him up to come to do the work. (ESV)

Bruce Abernethy

An important part of our life as Christians is to complete and be part of good works to bring glory to our Creator. If we use the talents and gifts we have been blessed with, to create beautiful and admirable works, and are ready and willing to share what we have learned and accomplished with others—people will notice. God is the Creator of all that is good and beautiful, and the giver of gifts and talents. Man is the maker or fashioner and should use their skills and accomplishments not for their own glory, but to draw people to God. If we as software craftsmen develop our skills and creativity, do our jobs right, work as we are called, share freely with others around us, and are ready to "give the reason for the hope" that we have when others ask (1 Peter 3:15-16), then we will have answered the call and done our part in introducing others to our amazing Creator.

Bruce Abernethy (bruce@abernethy.com, @babernethy) is a software architect, developer, a husband, a father of three, and serves in the leadership of Boy Scouts and Teen Bible Challenge. His current focus is web development, mobile web/app development, online marketing, social media and line-of-business applications. Bruce regularly speaks at user groups, regional conferences and coffee shops.

Teaching How to Write Hospitable Computer Code

Victor Norman, Ph.D.
Department of Computer Science
Calvin College

Introduction

Prior to coming to Calvin College, I worked as a software development engineer in three different companies for a total of 14 years. I loved, and still love, to write computer code, as I find writing code to be a creative outlet. At about the eighth year of my time at the first company, I began to notice I was repeatedly assigned to work on products that went to completion, but never sold any units. That is, the software and hardware were designed carefully, implemented, tested thoroughly, packaged, and marketed, but the initial design given to us software engineers resulted in a product that none of our customers wanted. This realization made me think about my role as a software engineer, and why I spent so much time and effort working so hard to create software that no one would ever use. If no one was going to use this product, why bother working so hard? Why bother working so diligently to get the design correct? Why bother reviewing the code? Why bother writing, reviewing, and implementing test plans, and then fixing bugs for weeks and months on end? Why bother reviewing my own code to make it "perfect", with excellent documentation, excellent naming, and excellent design?

From these musings, I came to two conclusions. First, as a Christian I was called to do my work diligently as an offering to God, even if no human being would ever run my code or use the product I was helping create. Second, I had to continue to create the best code I could, write the best documents I could, and document the code the best I could, even though the code may never benefit any human being.

Now I, as a Christian professor, try to teach my students these same lessons. First, the students must learn not only how to write code that produces the correct output, but also create what I now call hospitable code. In other words, I teach that not only the function, but also the form of the code matters, to me and to God. I explain to students that this is code written for two "consumers": the computer that will execute the code, and also others who will come later to review, understand, modify, borrow, and extend the code.

You may ask, Who looks at computer code after it is written? To answer this question, one needs to know about a typical software lifecycle.

Software Lifecycle

There is a saying in the software development community: Code is written once, but read a thousand times. This saying illustrates an important point: a programmer, when writing code, needs to remember that he or she is not just communicating with the computer, but also communicating with those that come later, who will have to read and understand the code.

In my experience in software development, after code is written, the programmer himself must read and debug the code. Then, the programmer's team holds a formal "code review" in which the team reviews all code in the project, looking for bugs, poor coding style, missing documentation, etc. Then, during final integration testing, stress testing, and release testing, a larger group of developers and testers may need to read and debug the code.

Even after the code has been released as a product, it will likely need to be read again. (The lifetime of a typical program is 20 years or so.) In most companies, software goes through multiple revisions—1.0, 2.0, 2.1, 3.0, etc. These revisions differ in two ways: new features are added, and existing defects are fixed. In the latter case, when code is being fixed, software developers spend many hours reading the existing code, looking for the defects. However, even when new features are added, it is often the case that programmers borrow and alter large portions of existing code to add a new feature.

So, code is read often after it has been written, and thus, the readability (or "form") of the code matters. But, what does it mean to write hospitable code? I address this question next.

Hospitable Code

Most people, when expecting guests, try to make their home hospitable by cleaning it, lighting it properly, making it comfortable, and warm. They prepare food and drink, and perhaps, entertainment. In short, they make the space welcoming. I teach my students that hospitable code gives the reader of the code this same sense of being welcomed, a sense of warmth, and a confidence that the code was created with care. Hospitable code welcomes the reader to come in and be comfortable, to enjoy the cleanliness of the code, to feel at home, and to see that the space has been carefully prepared with guests in mind.

How does one do this with computer code? A programmer has many choices to make when writing code, many of which affect the readability of the code. Let me give three examples of choices the programmer has which can affect the hospitality of the code.

- *Clear and descriptive variable names.* A line of code as simple as

  a = 92

  can be improved and be made more hospitable to the reader simply by instead being written:

  minAGrade = 92

  When a reader encounters the first line, the reader may not immediately understand the purpose of the code. This may make the reader anxious that he is al-

ready losing understanding of this code. However, if the reader instead encounters the second line, he can intuit immediately that the code uses this variable to indicate the minimum score that is an A grade. Thus, the reader is left more confident that he understands the code so far.

- *Proper in-code documentation* (i.e., comments). All programming languages (that I know of) allow comments to be written in the code. These are lines that are not executed by the computer, but are written only to communicate with a human reader of the code. The proper level of documentation in the code explains to the reader any tricky or non-obvious steps or structures in the code.

- *Consistent indentation*. All modern programming languages contain control structures that cause the code to execute code conditionally or repeatedly. The coding structures can become nested within one another. For example, here is a piece of code that is not indented:

```
foreach elem in aList {
if (elem.score < 60)
{ newList.append(elem);
aList.remove(elem);
}
}
```

Understanding this code is difficult. However, if I indent the code consistently and carefully, it becomes much easier to see that the code removes all elements from aList have scores less than 60, and adds them to newList.

```
foreach elem in aList {
    if (elem.score < 60) {
        newList.append(elem);
        aList.remove(elem);
    }
}
```

If one does not indent the code consistently, then the reader will find it difficult to determine how control flows through the code. This difficulty can undermine the reader's confidence in understanding the code's logic.

Why Christians Should Write Hospitable Code

In my classroom, I emphasize to students that they must get in the habit of writing hospitable code, because it is the Christian thing to do (and it is the only way to get a good grade in my class). I explain that a Christian should write hospitable code for these reasons:

Hospitable code is code written with others in mind.

I have argued above that a programmer writes code not only for a computer to execute, but also for others to read, modify, and re-use. Thus, the Christian computer programmer should write code keeping in mind that this code needs to serve others in the community. The programmer should have a servant's attitude, looking toward the needs of others. This is a Christian attitude, clearly demonstrated by Jesus Christ, who came not to be served, but to serve. The temptation for many programmers is to think that the code just needs to have the correct functionality, and its form does not matter. The Christian should remember that both functionality and form matter to those people who come later to use this code.

Hospitable code is code can be written to serve God.

A Christian can serve God by writing hospitable code, because the Christian is doing his or her best to create code that is readable, correct, and looks to serve the needs of others (1 Peter 4:8-11). I remind my students that writing computer code is a creative effort. (In fact, creating computer code is my personal creative outlet—it is one thing I enjoy doing in my "off hours.") In creating code, we emulate God in his acts of creation. In fact, the first characteristic we learn about God is that he is a creative being (Gen. 1:3). However, to truly emulate God's acts of creation, we must create things that are good. I teach my students that one way to create "good" programs is to write hospitable code.

Victor Norman

Creating hospitable code demonstrates integrity and God's lordship over all.

I emphasize to my students that function and form both matter, to me and to God. God judges us not only by what we do, but also by who we are. Similarly, we need to create code that is "good" throughout the creative process. We don't want to be pharisaical in our creations, creating code that is a "whitewashed tomb"—beautiful on the outside, but ugly on the inside (Matt. 23:27). I believe considering the form of computer code to be important is a truly reformed attitude, and I believe it is also acknowledges God's lordship over all.

Victor Norman (vtn2@calvin.edu) is an assistant professor of computer science at Calvin College in Grand Rapids, MI. Before becoming a professor, he worked as a software engineer for 14 years in the networking and file system industry. He currently teaches introductory programming (in python) and networking classes. He has a passion for missions, and will be taking a group of 8 students to England in January to do some programming with a missions organization for 3 weeks.

From Skeptic to Recruiter:
How a Missions Internship Changed My Life

Dorinda Beeley
LightSys Technology Services

"The laptop and printer and digital camera are essential to our ministry," the missionary shared. "Many supporters don't understand why these expenses are necessary."

It was the summer of 2000. I stood in a tiny apartment, chatting with an inner city missionary and his wife. And I didn't agree with a thing the missionary was saying. I agreed with those supporters. None of my friends had digital cameras. Why did a poor missionary need one? If you're living life on a missions budget, you don't need to splurge on technology, do you?

Computer major though I was, I saw little need for missionaries to spend supporters' money on electronics.

Fast forward three months. The hunt was on for my required computer internship when a cousin heard of the search.

"Have you thought about doing an internship with Gospel for Asia?" he asked. "We were praying this week for an IT intern."

I said I wasn't interested. Oh, I wasn't against missions. Promoting missions had been part of my church responsibilities for years. But doing computer work at a mission organization? That was just strange.

Two months later. Other opportunities closed. As a last resort, I called the IT director at Gospel for Asia (GFA). An application and several conversations later, I was accepted as an IT intern at GFA for summer 2001. And boy, did I have a few lessons to learn!

Lesson #1: Missions IT is a necessity, not a luxury.

My first job was helping with a gateway IP address change. This gave me exposure to how many computers there were in the office and how they were used. I was surprised! The staff depended on their computers for almost everything: contacting churches, processing donations, tracking finances, selecting mailings for donors, communicating with oversees offices, responding to donors, assembling the magazine, designing banners for conferences, and almost everything else.

A small, but neatly wired data center housed a phone system, routers, switches, databases, a print server, email servers, Linux firewalls, NAS, and VPN servers. They had more servers than my college did! If the network went down for some reason, the office workflow slowed to a near-halt. It was obvious that IT was not a "luxury." It was the essential electronic backbone that kept the office running.

> Missions IT is ministry.
> It is worship.
> It is prayer.
> It is about God.

Lesson #2: Missions IT is more than computers—it is ministry.

Three days a week, the staff gathered for an hour of prayer before the workday even started. The other two days, we often prayed as a department. Every Tuesday night was an all staff and families prayer meeting and it was considered crucial to be there.

Prayer was not just about the mission field. While we spent time in prayer for that, we also prayed for the building maintenance, the accounting department, the paper folding machine that quit working, and the database upgrade.

The first time an IT co-worker shared a prayer request for an IT problem, I think I laughed. But I watched the answers come time after time and I began to realize, "These people are serious about depending on God—even for the IT solutions!"

The relationship side of ministry also caught my attention. I'll admit—I was not the model of a perfect intern. While I was brought on for a specific database project, I didn't have the experience or know-how to get the project done. This left my coordinators needing to define a new project for me.

But outside of the projects (or despite the projects), I found my coordinators were most interested in *me*. They wanted to know how I was doing. They invested their time in teaching, correcting, encouraging, and exhorting me—spiritually, personally, and professionally. That befuddled me. After all, I was just here for an IT internship, right?

It took most of the summer, and even beyond, for me to realize that computers are only a part of what Missions IT encompasses. Missions IT is ministry. It is worship. It is prayer. It is about God.

Lesson #3: Missions IT is a lever, not just a necessity.

One of my internship responsibilities at GFA included software training for the staff as the office changed from one email client to another. I like training people, but I really dislike the time required to create documentation. I complained one day to my project supervisor.

"If we want the staff to be doing things that will help reach the unreached, why are we pulling them out of their offices for an hour for this training? And why am I spending 40+ hours getting this training ready? Surely there is something else I can be doing that would better use my time!"

His response showed me the bigger

picture. "People can easily waste 15 minutes a day trying to figure out how to do something on their computer. If we can provide training that saves 50 staff people each 5 minutes a day, 5 days a week—that's twenty additional hours a week that the staff have to call pastors, talk to donors, partner with missionaries. We can plant churches this way!"

That theme continued through each area of IT. The IT staff wasn't hyped over the latest techie toys or the hottest new server product on the market. Their question was: "What technology will make us most effective for the kingdom of God?"

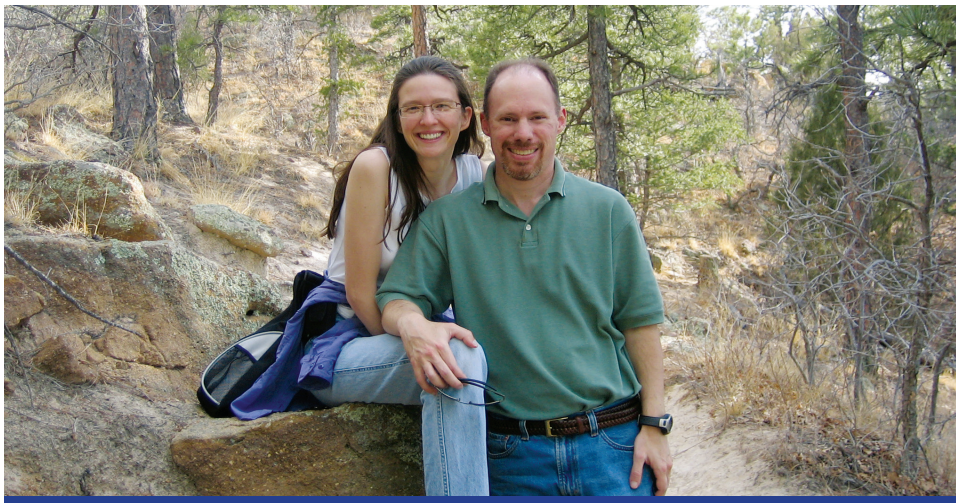
"God has given us technology as a tool," my supervisor said. "And I want to use that tool as a lever! How can we use technology to make our staff more effective in their work for the Kingdom? How can we use technology as a lever to multiply our efforts and reach more people with the Gospel?"

Computers were not just a tool that the mission used because they had them. It was a vital tool—a lever that they wanted to use to propel the Gospel forward at an even greater rate.

Lesson #4: Getting involved with Missions IT may change your life.

I left GFA at the end of the summer knowing, but not fully realizing, that my life would never be the same.

The need for more laborers for the mission field was now a part of my thoughts. 80,000 people that die every day without ever having heard our Savior's name. 300,000+ villages in India alone that have never had a Gospel witness. Missionaries that labor for 12-18 hours a day, but are besieged with still more people asking them to come share the Gospel in their towns. We needed more missionaries, more staff to support those missionaries, and more IT staff to support the infrastructure.



Dorinda and Greg Beeley

I knew those things now. And God was going to hold me accountable for that knowledge.

Lesson #5: It's all about God.

I journeyed my senior year of college asking, "Lord, what do you want me to do?" I *had* to be involved. But beyond prayer, did God want me to earn wages and financially support the missions work? Or did He want *me personally* as one of the Missions IT laborers?

There wasn't an audible voice of God or a super-spiritual "call." Just simple peace to go ahead and interview with GFA after graduation. When the door opened for me to join GFA's staff full-time, I walked through.

The last ten years have been an incredible journey—raising financial and prayer support as a state-side missionary, serving at GFA for seven years, traveling to India to see the mission work there, meeting my husband at a computers in missions conference, and now serving a variety of mission organizations through LightSys Technology Services.

The longer I'm in Missions IT, the more I see how much of a lever technology can truly be as we spread the name of Christ throughout the world. From back office servers to front line evangelism, from Bible translation software to electronic Bibles in closed countries, from secure communication to discipleship via mobile devices, from the obvious to the things one wouldn't imagine—Missions IT is one of the greatest tools we have for advancing God's Kingdom.

But it's not about us or the computers; it's about God. It's about His worth. His glory. It's worshipping Him in our colleges, our workplaces, and our lives. It's remembering that half the world is waiting for someone to tell them of the one true God so that they can worship Him too.

What part is God calling you to play? Are you willing?

Dorinda Beeley is an '02 CIT graduate of Sterling College. She's passionate about Missions IT and loves to share that passion with college students and IT professionals. Dorinda and her husband, Greg, serve with LightSys Technology Services (URL http://www.lightsys.org) providing free of charge IT support for missions. Interested in Missions IT? Looking for an internship? Contact Dorinda at Inquiries@lightsys.org.

Dynamic Link Conference 2011

Calvin College's Information Systems 371 class (IT Leadership) organized the the 2011 Dynamic Link Conference which was held on April 30, 2011 at the Devos Communication Arts and Science Building on the Calvin College Campus.

The purpose to Dynamic Link is to provide a forum where computing students and guests, which include software industry professionals, can meet to discuss the role of faith in software development.

The theme of the 2011 conference was "The Christian Responsibilities of IT Leaders." The conference opened with keynote talks, followed by an afternoon of focus groups which included lunch. The keynote speakers were Dr. Paul Jorgensen and Mr. William Noakes.

Dr. Jorgensen is a professor of computer science at Grand Valley State University. He is the author of text books in software testing and software behavior modeling. His book, "Software Testing—a Craftsman's Approach," is now in its third edition and is one of the primary references for software testing in the ACM and IEEE's Software Engineering Body of Knowledge. Prior to an academic career, Dr. Jorgensen was a manager of software testing for some of GTE's most significant projects.

Mr. Noakes is President of the Noakes Group, a private consulting company providing services and advice to "C" level executives. He also had the dual role of General Counsel and Chief Information Officer at Meijer. Prior to joining Meijer, Mr. Noakes' career included service with the Security and Exchange Commission, membership in GM's legal staff, appointments to positions of public service and an engagement as a commentator on Court TV.

The Discussion Groups

Guests and students were organized into three discussion groups in the afternoon. Prior to the conference, the students were assigned to one of the specific groups with a student leader and a guest co-facilitator. The students researched the topic for their group and prepared a recommendation and presentation. After the presentation, the students and guests engaged in discussions that resulted in a set of recommendations provided at the end of the conference. The groups, the facilitators, the topics and an abridged version of the discussion summaries by the students follow.

Group A Facilitated by Melissa Bugai (Consultant) and Matt Bushouse (student)—*Lost Opportunities: Is There a Corporate Responsibility to Attract Women to Computing Professions?*

"… As the team preparing for the discussion, we believe that women are currently not attracted to computer science, but we do not believe that any one social entity is responsible for this trend or its correction. We believe that this trend may very well change because of the increased pervasiveness of computing in ways that will appeal more to women, provided there is a concerted effort to change stereotypes and perceptions of computing as a career.

"The discussion group concluded as a whole that gender diversity is a subject in great need of redress; there is a vast untapped reserve of talent that can bring new and valuable perspectives to the field. … Perhaps the best way to address this gender split is through dismantling the negative stereotypes. This can be accomplished by directly hiring more women, making the field feel less isolated from other people, designing computing clubs that focus on women and their social desires, and encouraging those in the workforce to communicate the current industry to the schools."

Discussion Group B Facilitated by Dr. Roger Ferguson (GVSU) and Nana Owusu-Achau (student)—*The Call for Responsible Software Developers: Should Society Require Licensing for Software Engineers?*

"Unlike traditional engineers, software engineers are not required to be licensed. They are free to practice their profession without government supervision. Before the discussion took place, the students preparing for it felt that this arrangement was incredibly irresponsible and felt that the government has a responsibility to its citizens to protect against poor code.

"However, the discussion group concluded that now is not the time to force licensing on software engineers. It was noted that as any field matures, the licensing often starts when it is appropriate, and the requirements for the licensing become stricter as the field matures."

Group C Facilitated by T.R. Knight (Taylor University) and Ken Echtinaw (student)—*It's Still There: Do Chief Information Officers Have an Inherent Responsibility to Identify and Address the Digital Divide?*

"Group C was tasked with exploring the digital divide that many interpret as a growing problem in our technologically driven society. However, we did not focus on the digital divide in the traditional sense, where the problematic gap is perceived between those who have technology and those who do not; rather, we looked carefully at the growing gap as it is expressed in countries where the technology is readily available, but many of its users are ill-equipped to take full advantage of its resource.

"Although details were debated throughout our group's discussion, a shared conclusion was somewhat organically formed. The extent of a CIO's responsibility to both address the issue of a digital divide and be proactive in a solution remained inconclusive, but the group did agree that, particularly as a Christian CIO, there is an inherent responsibility to be proactive in the matter. "

The discussion papers for each group are available for reading at *http://cs.calvin.edu/p/DynamicLink

Dynamic Link 2011 Conference Discussion Groups

(S) = Student, (G) = Guest

Group A: Brent Sloterbeek (G), MariLou Richardson (G), Aravind Ranganathan (S), Aaron Koenes (G), Sarah Frisch (S), Melissa Bugai (Guest Co-Facilitator), Kent Voskuilen (S), Raylene Bradshaw (S),Cameron Boot (S-Master of Ceremonies), Erin Bushouse (S), Matthew Bushouse (Student Leader)

Group B: Nana Owusu-Achau (Student Leader), Brian Williams (G), Randy McCleary (G), Roger Ferguson, Ph.D. (Guest Co-Facilitator), William Noakes, JD (Keynote), Andrew Cooper (S), Robby Hoekstra (S), Brian Derks (S), Paul C. Jorgensen, Ph.D. (Keynote), Sim Vanderbaan (S), Chris Brown (S), Joel Adams, Ph.D. (Chair, Calvin Computer Science), Victor Norman, Ph.D. (Calvin Computer Science)

Group C: Carissa Barents (S), Joe Girolamo (S), Nicole Veenkamp (S), Denise Mokma (G), Rick Devries (G), Brian VanderZee (G), Ben Van Drunen (S), T.R. Knight (Guest Co-Facilitator), Kari Witte (S), William Vriesema (G), Barbara Egeler-Bailey (G), Ken Echtinaw (Student Leader), Priscilla "Yosh" VanOmen (G)

This journal is a publication of the Calvin College Department of Computer Science.
More information about the department is available at **http://www.cs.calvin.edu**.

Computing@calvin.edu

If you would like to propose an essay for the next release of *Dynamic Link*, be a participant in the next Dynamic Link Conference or offer a donation to support *Dynamic Link*, contact us at the email address above. Thank you!

The organizations below provided significant contributions to make this journal possible.

We are grateful for the individual contributions from the following:

Terry Woodnorth, Endicott, New York

Patrick and Barbara Bailey, Grand Rapids, Michigan