David Owen

Article

DOI: https://doi.org/10.56315/PSCF3-26Owen

Tractability, Testability, and Joyful Work

David Owen

Fred Brooks is remembered for his Christian faith and for many contributions to computer hardware and software design, management, research, and education. Brooks is known particularly for his 1975 book, The Mythical Man-Month*, which includes reflections on unique joys, as well as management challenges, of software development. Computer programmers, according to Brooks, work in a tractable medium: What one imagines may be readily constructed, with few constraints imposed by the medium. And yet, in contrast to other kinds of creative work, the construct produced is testable: It is possible to determine experimentally, in a repeatable and unambiguous way, whether one has succeeded or failed. Here I consider whether these ideas of tractability and testability might contribute to our understanding of what makes (or could make) other kinds of work joyful. I also consider Brooks as a model for Christian vocation, as Brooks's writing provides the basis for a definition of joyful work grounded in a Christian understanding of the human person, made in the image of God.*

Keywords: Fred Brooks, Christian faith and vocation, software development, technology, automation, tractability, testability, joyful work

> The voice we should listen to most as we choose a vocation is the voice that we might think we should listen to least, and that is the voice of our own gladness … In a world where there is so much drudgery, so much grief, so much emptiness and fear and pain, our gladness in our work is as much needed as we ourselves need to be glad.
>
> –Frederick Buechner[1]

Frederick Buechner's words are profound and worthy of careful reflection. His ideas have been inspiring to many. But it is hard to read this and not react with questions: What about all of the unpleasant but much-needed work in this world? What about the needed work which might bring gladness but not a living wage? What if the voice of my gladness is shallow and self-centered, or uncertain, or inconsistent?[2] It may be true that I need to be glad and that the world needs my gladness. It may be true that I am called to find that place "where [my] deep gladness and the world's deep hunger meet," as Buechner famously said elsewhere.[3] But if I haven't yet found that place, how do I get from here to there? What practical steps can I take to move in that direction?

In this article, I reflect on ideas from Fred Brooks about gladness that he found in the work of software development. Rather than questioning or critiquing Buechner's advice, this article proposes Brooks as a model for Christian vocation consistent with Buechner: Brooks's writing about the joys of computer programming provides the basis for a definition of gladness in one's work grounded in a Christian understanding of the human person, created in the image of God. Moreover, Brooks's positive influence on so many people who knew him and worked with him throughout his

David Owen *teaches computer science at Messiah University in Grantham, Pennsylvania. In graduate school, he studied verification strategies used to assure that desired properties hold for a distributed software system. His more recent interests include computer architecture, graphics, and programming languages.*

life provides an illustration of Buechner's idea that gladness in one's work, shared with others, is something much needed in our world.

The Mythical Man-Month

Brooks is probably best known among software developers for his book, *The Mythical Man-Month,* first published in 1975.[4] Brooks writes about the joys of the craft of computer programming, including "the delight of working in such a *tractable* medium"—a medium "readily capable of realizing grand conceptual structures"—and yet creating a *testable* construct, "real in the sense that it moves and works, producing visible outputs separate from the construct itself."[5] Here, I consider, as Brooks does, tractability and testability as objective qualities, which may vary from one kind of work to another. Work in a tractable medium, to produce a testable construct, is *joyful* work—it produces various joys described by Brooks and summarized below. I also consider tractability and testability in the subjective experience of the worker, varying from one worker to another. The medium most tractable for you, through which you are best able to realize what you imagine, I may find awkward and frustrating to work with.[6] The construct most testable for you—you know how it ought to work, and it is obvious to you when you get it right—may be difficult for me to understand and evaluate.[7] Work in a medium that is tractable for you, to produce a construct testable for you, is likely, for you, to be joyful work—work in which you might hope to find deep and lasting gladness.

Fred Brooks wrote *The Mythical Man-Month* "to explain the quite different management experiences [he] encountered in System/360 hardware development and OS/360 software development" at IBM in the 1960s.[8] Software development was something relatively new at the time and, according to Brooks, surprisingly challenging to manage. The title of the book comes from Brooks's observation that the development of a complex software system requires much communication between developers. The bigger the team, the more communication is needed, so the idea that a development effort be measured in "man-months" is a "dangerous and deceptive myth."[9] Adding more workers to a task that requires training initially, and then ongoing communication between them, might not speed up the task; this might even slow it down, depending on just how much training and communication are needed. Because software development requires much training and communication, Brooks offers the following (greatly simplified, he admits) "law": "Adding manpower to a late software project makes it later."[10]

Referencing Andrey Ershov,[11] Brooks describes not only surprising management challenges but also particular joys of software development. Brooks mentions tractability and testability specifically in two places, but I suggest here that all of the joys he describes flow naturally from the tractability of the medium and the testability of the construct, if these are understood broadly as I have defined them above.

There are joys that flow from the tractability of the medium—joys of creativity and of excellence:

- The joy of creating something new, something of one's own design. "I think this delight," says Brooks, "must be an image of God's delight in making things, a delight shown in the distinctness and newness of each leaf and snowflake."[12]
- The joy of building a "complex, puzzle-like object of interlocking moving parts" that seems to take on a life of its own.[13] This is the joy of the craft, of excellence, of an elegant and not merely functional solution—a delight in the goodness or completeness of the thing one makes.

And there are joys that flow from the testability of the construct—joys of purpose and of learning:

- The joy of making something useful to others, the joy of solving a problem that others have judged to be worth solving, the sense of purpose and fulfillment in service to others.[14]
- The joy of learning, of growing in skill and understanding as one's ideas are tested experimentally in a working implementation.[15]

But joyful work is not always delightful and fulfilling. Gladness is not merely the absence of pain or uncertainty. There are woes inescapably intertwined with the joys of software development. There are woes that flow from the tractability of the medium—woes of confusion and instability:

- The woe of "arbitrary complexity" (as Brooks describes it in a later essay):[16] Programmers invent wonderfully and woefully complex systems, puzzling to themselves and bewildering to others.
- The woe of imminent obsolescence: Because the medium imposes so few constraints and evolves so quickly, others move on to new and better ideas before there is time for one's current idea to be realized.[17]

And there are woes that flow from the testability of the construct—woes of frustration and dependence:

- The woe of debugging, the tedious and frustrating effort to correct one's ideas and implementation so that the program runs correctly on a strict and unforgiving machine.[18]
- The woe of dependence on others, including managers or users who decide what one's program should do; including other programmers whose arbitrarily complex work must be integrated with one's own.[19]

Because the joys and woes are intertwined, the woes must be managed in order to make joyful work a realistic possibility. Gladness in our work, to be sustainable, must be found within the "real world" of daily life among flawed human beings. There are management challenges associated with tractability—challenges of unrealistic optimism and the need for much communication:

- The challenge of unrealistic optimism: In contrast with other activities that create or build something, activities in which the physical limitations of the medium serve to constrain one's ambitions, the comparatively unconstrained nature of computer programming leads to unrealistic optimism about how much can be done, and how quickly.[20]
- The need for much communication: Because of the tractability of the medium, every project is potentially different and new; much training and ongoing communication is therefore needed if developers working together are to understand each other's ideas.[21]

These challenges are also associated with testability:

- Unrealistic optimism is a challenge precisely because developers' ideas will be thoroughly tested in the implementation. "Because our ideas are faulty, we have bugs; hence our optimism is unjustified."[22]
- And much communication is needed, because developers must indeed understand each other's ideas; misunderstandings will become obvious as they are manifested in a program that doesn't work correctly or doesn't even make sense to users.[23]

Much of *The Mythical Man-Month* is concerned with the elaboration of these challenges and with strategies for addressing them. My experience as a professional software developer was short and was nearly twenty years ago, before I began a career in computer science education; I have no experience managing a team of professional developers. So I am not in a position to evaluate the strategies Brooks suggests. I trust him, and I trust a generation of developers and managers influenced by the ideas in his book, that strategies like what he suggests are necessary to make software development work sustainable and productive, especially as teams grow larger and projects more ambitious. Still, I am interested in this question: If the joys, woes, and management challenges of software development flow from tractability and testability, and are thus intertwined, will attempts to address the management challenges—to mitigate the woes—risk diminishing the joys as well? To put the question more broadly, as I imagine Buechner might: Will attempts to make work sustainable and productive in the "real world" necessarily undermine workers' opportunities for deep gladness? In the next section, I will consider a few of the management strategies Brooks suggests with this question in mind.

The Challenge of Conceptual Integrity

The idea most often associated with *The Mythical Man-Month* is implied by the title: software development effort should not be measured in "man-months," because much communication is needed between developers working together. But this is not the most important idea in the book, according to Brooks; for Brooks, the book is about "conceptual integrity" and about strategies for managing a team of developers so as to produce it in a programming product:

> A clean, elegant programming product must present to each of its users a coherent mental model of the application, of strategies for doing the application, and of the user-interface tactics to be used in specifying actions and parameters. The conceptual integrity of the product, as perceived by the user, is the most important factor in ease of use … There are many examples of elegant software products designed by a single mind, or by a pair. Most purely intellectual works such as books or musical compositions are so produced. Product-development processes in many industries cannot, however, afford this straightforward approach to conceptual integrity … Any product that is sufficiently big or urgent to require the effort of many minds thus encounters a peculiar difficulty: The result must be conceptually coherent to the single mind of the user and at the same time designed by many minds. How does one organize design effort so as to achieve such conceptual integrity? … Deliberate, and even heroic management actions are necessary to achieve coherence.[24]

What sort of management actions are necessary? First, Brooks prescribes a strict distinction between architecture and implementation: a distinction between "what" the product is and does, from the user's point of view, and "how" it works internally.[25] The implementation—making the thing and making it work—may require the effort of many developers working together, but a single "architect" should be assigned to design the "what" of the product, acting as a representative for a single user's point of view.[26] Developers working with the architect to implement the product should be assigned distinct roles to form one or more "surgical teams," each responsible for a relatively independent aspect of the implementation.[27] Each team's "surgeon" is responsible for directly doing the work; others have distinct and clearly defined supporting roles. This structure, with a single architect and developers working in surgical teams, addresses the challenge of conceptual integrity, both at the architectural level and at the level of each surgical team. It also reduces the need for communication between developers: Every surgeon communicates with the architect, but little communication is needed between surgeons from different implementation teams; within a team, communication can be focused because members have distinct roles and responsibilities.

If the joys, woes, and management challenges of software development are intertwined, what impact might such management actions have on the experience of individual developers? Specifically, how might an individual developer's experience of the tractability of the medium and the testability of the construct be impacted? I will consider this question from the point of view of an architect, of a senior developer (at the level of Brooks's surgeon, but in a present-day development team), and of a member of the development team in a supporting role.[28]

The joy of creating something new, something of one's own design—this is the joy of the architect, together with a sense of purpose, the joy of making something useful to another, as the architect represents the user's point of view. The joy of the craft, of excellence in building something and seeing it come to life; the joy of learning to build with greater skill and understanding—these are the joys of the development team, particularly of senior developers. There is also the joy of making something useful, as the implementation is validated by the architect and contributes to the overall design. But what are the joys for those in narrower supporting roles? Their work is validated by senior developers and by the team's contribution to the overall design, assuming that they are included in the big-picture vision for the product and encouraged to identify with the user's point of view. There is a possibility, however, that as supporting roles become more specialized, their work may be valued primarily for its utility; others will call it "good" if it is good enough for their purposes, not recognizing or appreciating its excellence as someone experienced in the same kind of work would be able to do.[29] This is a danger for all roles as they become more specialized and distinct from each other, but especially for supporting roles as their work is further removed from the particulars of the product. There is a sense of purpose, a joy in making something useful to another, to be sure. But ideally the other joys—joys of creativity, excellence, and learning—would not be diminished for those in supporting roles.

How to Avoid Diminishing the Joys?

Is it possible to maintain conceptual integrity, assuming this requires a hierarchical structure of specialized roles, without diminishing the joys of creativity, excellence, purpose, and learning? Brooks must believe it is, as *The Mythical Man-Month* begins with an eloquent description of these joys before moving on to discuss management challenges and how to address them. One finds helpful ideas throughout the book. Here are some examples:

- The architect is responsible to make design decisions in a way that ensures conceptual integrity, but others should be encouraged to contribute design ideas.[30] Likewise, Brooks's surgeon (or perhaps a senior developer today) makes implementation decisions, but others should be encouraged to contribute implementation ideas. In particular, the architect should be prepared to suggest a possible implementation for any feature of the (architect's) design.[31]
- Assigning distinct roles might sometimes increase opportunities for creativity. If responsibilities are divided so that each person has authority to make decisions in their own area, they will be freed from time-consuming committee decision-making and enabled to focus on doing their own part well.[32] I would suggest that responsibilities be divided, not only so that each person has authority to make decisions in their own area, but so that each may also experience tractability in the medium of the work (choices through which they can express themselves creatively) and testability in the construct (time and resources to get it right, from their point of view, even if they have a higher standard than others).

- Brooks famously suggests that software developers "plan to throw one away."[33] The first version will be bad, but the team will learn from it, and the next will be much better. (This may be the second-most-well-known idea in the book.) In a later essay, he generalizes this idea and advocates for an incremental development process.[34] Either way, the idea is to treat learning as a normal and expected part of the design and implementation of a new software product. Consistent with this, Brooks suggests a non-threatening organizational structure with mechanisms for public documentation of tentative commitments.[35]
- Roles should be specialized and distinct, but each person should be trained for multiple roles and should be given opportunities to move between roles.[36] Also, a good manager will work to reduce role conflict[37] and to minimize sociological barriers (so that management roles are not given higher status, for example.)[38]

In the next section, I will consider whether these ideas from software development might offer insight into what makes other kinds of work joyful (or not). But first, imagine a team of developers working together on a project just for the joy of it, without the constraint that it must be profitable in order for them to remain employed. What sort of management structure could make such work sustainable and productive? Eric Raymond asks this question in *The Cathedral and the Bazaar*, a book about "hacker" culture and the open-source movement as it emerged along with Linux and the World Wide Web in the 1990s.[39] In a nutshell, the structure looks like this:

- A project typically has an "owner"; this person "has the exclusive right, recognized by the community at large, to distribute modified versions."[40]
- A project is initiated because it meets a need for the initial owner, who is the initial user, manager, architect, implementer, and tester. "Every good work of software starts by scratching a developer's personal itch," says Raymond.[41]
- The owner shares the project with (many) potential user-developer-testers. The owner exclusively retains the architect and manager roles while continuing to use, implement, and test the evolving code.
- Non-owner participants contribute feature suggestions, implementation changes, bug reports, and fixes publicly, but no changes are officially distributed to the group without the owner's approval.

The project itself is initially motivated by a developer's need. But what motivates the developer to share code with others? A variety of things, according to Raymond, including the need for help from other user-developer-testers, but most importantly "prestige," awarded and maintained in a "gift culture" based on shared appreciation for development ability and high-quality software.[42] In contrast to popular open-source licenses (the GNU General Public License, for example), which embody an "anyone-can-hack-anything consensus theory," says Raymond, "the open-source culture has an elaborate but largely unadmitted set of ownership customs."[43] In other words, ownership is highly respected. Any non-trivial change must be approved by the owner before it is distributed to the group, and transfer of ownership must happen in a deliberate, public way, even for apparently "orphaned" projects no longer maintained by the most recent owner.

In some ways, the open-source structure, as described by Raymond, is surprisingly similar to the structure suggested by Brooks. There is an architect (the owner), a single mind responsible for the conceptual integrity of the product; there are distinct roles that function like a surgical team: a senior developer (the owner) who has the final say, and a team of supporting user-developer-testers who offer suggested changes.[44] Those in supporting roles clearly do find joy in their work, or they wouldn't keep doing it. So a multi-level structure, with distinct roles and responsibilities and a dynamic of unequal prestige—whether in Brooks's surgical team, Raymond's open-source gift culture, or a present-day development team with a mix of senior and supporting roles—is not necessarily opposed to joyful work; in fact, such a structure may be important for making joyful work sustainable and productive.

What About Other Kinds of Work?

Brooks, and likewise Raymond, make a distinction between software development and other kinds of work, such as "reaping wheat or picking cotton"[45] or "flipping burgers or digging ditches."[46] The implication is that, in software development, joyful work—creativity, excellence, purpose, and learning—is rewarded by productivity and is thus sustainable; in other kinds of work, such things may be irrelevant to productivity or even opposed to it.

In *Peopleware* (another well-known classic in the genre of software project management), Tom DeMarco and Timothy Lister devote the second chapter to a critique

of managers who treat developers as if they were making and selling cheeseburgers in a "production environment."[47] In such an environment, it makes sense to "optimize for steady state," to "do everything by the book," and to "eliminate experimentation."[48] To be fair, this advice is offered tongue-in-cheek, with the "production environment" put forward only as a foil for the management style DeMarco and Lister advocate for software development, which honors the unique value of each worker in various ways. Still, the suggestion is that software development is a special kind of work for which it is important to treat people well if you want to get a good product; there are, apparently, other kinds of work—production work—where treating people well is optional or even counterproductive.

This distinction between creative design and development work versus production work is treated by Brooks (and Raymond, and DeMarco and Lister) as an essential fact of software development, which managers would be foolish to ignore. Brooks writes about software development at scale as something new, as it was in the 1960s, for which he and others discovered that new management strategies would be necessary. In the years since, one finds a tension in the literature between this view, that unique management strategies will (always) be necessary, and an opposing view, that with adequate research, informing better processes and more effective tools, software development can be made more manageable—more like production work.

The first edition of *Peopleware* was published in the 1980s, 10 years after *The Mythical Man-Month* and 20 years after the work described in that book. The tone of *Peopleware* is humorous but scolding toward managers who foolishly try to turn software development into production work. Twenty years after *Peopleware,* Joel Spolsky preached the same message in his popular blog:

> Making software is not a manufacturing process … It didn't make any sense then [in the 1980s] and it doesn't make sense now. Shoving a lot of programmers into a room and lining them up in neat rows did not really help get the bug counts down.[49]

Moreover, one explanation sometimes offered for the open-source movement, as it emerged in the 1980s and 1990s, is that programmers who felt the joy of their paid work being managed away—by managers trying to turn it into production work—found that joy again as they began to use newly available personal computers and the Internet to work creatively with like-minded others in their free time.

Several years ago, I attended a lunchtime presentation for engineering students. The speaker talked about her work to automate part of a manufacturing process in which a complex wiring harness was assembled for use in an electric vehicle. The machine used for assembling the harness required a skilled operator to use it correctly. If the operator made a mistake in positioning the components or in the timing of joining them, this could produce an unusable wiring harness that would have to be thrown away. The speaker's work, to automate a part of the process, made it much easier for the operator to get it right, reducing waste and increasing productivity. Still, I wondered how these new features might change the experience of the operator of the machine. Would the operator's work—which was quite repetitive, maybe not very interesting to begin with—become a little more boring, a little less meaningful? Some of the skill that had been required before would now be irrelevant. There would be less reason to try to do the job well, less reason to learn how to do it better. Automation, despite its obvious benefits, would decrease the tractability of the medium. Automation would also decrease the testability of the construct, at least in the experience of the operator. Perhaps some expertise had previously been required to judge whether a wiring harness was usable; perhaps, previously, it would have made sense to say that a particular harness had turned out exceptionally well and to take pride in that. But the improved machine can now produce consistently good wiring harnesses, whether or not the operator is capable of judging their quality.

This is all speculation, of course. It could be that the operator would be thrilled to finally have a machine that does what it is supposed to do. My point in including the story is just to illustrate an idea: that qualities of testability and tractability, and thus the potential for joyful work as I have defined it—creativity, excellence, purpose, and learning—may be found even in a production environment. What makes production work different is not that these qualities are essentially absent but that they are more easily lost. They are more likely to be opposed to increased productivity and are therefore vulnerable, as productivity is what typically motivates innovation.

Now imagine how the story of the wiring harness might have been different: What if the machine for assembling wiring harnesses were designed with tractability in mind? It would be important, then, that the operator have choices about how best to use the machine; perhaps the operator would even have input into the design of the machine or the wiring harness. What if

the machine were designed with testability in mind? It would need to provide meaningful feedback, motivating the operator to do their best and enabling them to learn how to do better. Ideally, there would also be a way for operators of similar machines to interact in order to recognize and appreciate each other's skill.

These suggestions might seem like a quixotic stretch, an attempt to turn a factory-floor production job into a very different kind of work. I had an experience like this, however, in my first job after college. I started in an hourly technician position, spending much of my time doing the tedious work of wiring hundreds of connections on the back of rack-mounted control stations. But since this was a small company, I worked with the engineers as well, helping to prepare the wiring instructions. The process used for preparing the instructions began with a spreadsheet automatically generated from technical (CAD) drawings; this was followed by cumbersome manual translation of the information into a new spreadsheet with a very different format, suitable for creating wiring instructions that a person in manufacturing would be able to follow without understanding the overall design.

Working with the engineers, I was able to suggest improvements to the process—so the work was tractable for me in a way that it was not for others working on the manufacturing side. Working on the manufacturing side, I was motivated to improve the process in a way that the engineers working only on the conceptual design were not—so the work was testable for me in a way that it was not for the engineers. I eventually suggested specific improvements to the process that were adopted by others in the company. Looking back, my suggestions seem simple—obvious things I'm surprised someone else hadn't already thought of (mostly implemented as a set of Excel macros). But I felt very good about what I had accomplished and how it was received by others. For me, at the time, it was joyful work.[50]

Moving beyond these specific examples, might these ideas of tractability and testability inform a more general way of thinking about what makes work interesting and meaningful, joyful for the worker? As mentioned above, Brooks sees the computer programmer's joy of making something new, of expressing oneself through creative choices, as "an image of God's delight in making things."[51] Elsewhere, Brooks references J.R.R. Tolkien and Dorothy Sayers, portraying the creative work of the computer scientist as an example of "the gift of work … the capability and the call to make things" given by God to all humanity in creation.[52] And Brooks, reflecting years later on the enduring popularity of *The Mythical Man-Month*, considers whether it might be more broadly applicable than he initially imagined: Perhaps the book "is only incidentally about software but primarily about how people in teams make things."[53] What if it were true that all of us—not just software developers but all of us, made in the image of God and living in community together—are called to creative work in cooperation with God and each other? Perhaps the only difference between a field such as software development, in which qualities of tractability and testability are more clearly evident, and supposedly less-meaningful production work is that these qualities, in the latter case, became irrelevant or opposed to productivity and thus were lost as an unintended consequence of innovation.

Albert Borgmann, in *Technology and the Character of Contemporary Life*, describes a technological transformation of work that has taken place in much of the world over the past 200 years.[54] This transformation of "pretechnological" work has made many kinds of work more productive, safer, and more humane. But it has also turned skilled crafts, which provided opportunities for creativity, excellence, purpose, and learning, into something like production work for many people. "The monotonous and endless repetition of one small task is typical of much modern labor and gives it its stupefying and draining character," says Borgmann.[55]

Borgmann may or may not be overstating the case. I have been blessed in my own work experiences, which have been rarely stupefying and often rewarding, especially in my relationships with coworkers. But surely many have not been privileged in this way. Consider "the monotonous and endless repetition of one small task" as opposed to the joys of work in a tractable and testable medium: There is no opportunity for self-expression through creative choices; excellence is irrelevant and would go unnoticed; the worker is disconnected from others who benefit from the work; and there is nothing to learn, only a simple task to be repeated. The joy is gone. Old woes may be gone, too, but there are new and deeper existential woes of boredom, loneliness, a lack of purpose, a lack of hope for the future. Why then do people continue to work? Not for the value of the work itself, but only for payment: Work has been reduced to a mere means.[56]

In response to those who might claim a unique status for software development or other creative design work, as opposed to production work, Borgmann would likely

point out that various kinds of work which we now think of as production work were much more creative and engaging in the past.[57] To summarize Borgmann's view, the long-term trajectory of technology is to transform more and more kinds of work into production work.

Attempts to transform software development into production work may have failed thus far, but they may be successful in the future. In fact, the arrival of generative AI tools appears very likely to be transformative for software development, in ways Brooks might never have imagined. Right now, the capability of such tools seems to be somewhere between that of an entry-level developer (with an undergraduate degree in computer science, for example) and someone with a few years of experience.[58]

For an experienced developer who is able to critically evaluate AI suggestions, there would be no loss in testability (although there might be an increased need for careful testing); and again, for an experienced developer who is able to interpret suggestions within a context of possible alternatives, there is a sense in which the medium becomes even more tractable.[59] However, someone less experienced may assume that the AI tool knows better than they do and may uncritically accept AI suggestions without considering alternatives; for them, tractability and testability (not to mention employability) would be decreased.

As the capability of AI tools continues to improve, the bar gets higher for what one needs to know to be in the "experienced developer" category—to be among those able to critically evaluate the output of the tools and thus work with them creatively and productively.[60] How much better will AI tools get? Assuming there will always need to be a "human in the loop," how many of these humans will continue to be needed? The long-term status of software development (and of other kinds of creative design and "thinking" work) as joyful work available to many people seems to be uncertain. And yet, we believe as Christians that the status of human beings, made in the image of God, is not uncertain. If it is true that we are made to experience the joys of creative work, even as these joys are bound up with the woes and management challenges we experience when we try to work creatively together, we should not accept as inevitable the loss of tractability and testability to innovation and automation.

But what can we do? Where we still have power to influence the quality of our own and of others' work, how might we promote these joys that flow from tractability and testability? Here again is the list of joys from earlier in the article, this time framed as questions one might ask in order to find ways to promote them.

How might we promote the joys of tractability?

- Are there opportunities for creative choices? What is unique about the worker and how might that uniqueness be expressed in the work?
- How is excellence defined, recognized, and appreciated? Are adequate time and resources provided, not merely to do the work, but to do it well? How might the worker be more directly connected to others who understand the work well enough to delight in its excellence?

How might we promote the joys of testability?

- Will others benefit from the work? How might the worker be more directly connected to those who will benefit?
- Are there opportunities for professional and personal growth? What will motivate the worker to develop greater skill and understanding? What feedback will make this learning possible?

Concluding Reflection: Fred Brooks as a Model for Christian Vocation

Fred Brooks was a pioneer in computer hardware and software design, remembered for his lasting impact on managers, developers, and users.[61] He is remembered also for his wisdom, humility, and kindness,[62] and as an inspiration to Christian professionals and educators working in computing and related technologies.[63] Hank Tarlton, in a tribute to Brooks written shortly after his death in November 2022, shares about Brooks's Christian faith as it informed and enlivened his work:

> Fred was never one to separate his faith from his work. It was integral to all that he did. You see this in his monumental book, *The Mythical Man-Month*, with references to scripture and quotes from C.S. Lewis and Dorothy Sayers. And you would hear it in his public speeches, speaking with clarity about the role of Jesus Christ in his life as a computer scientist before all his colleagues … It was no surprise to those who knew him that Fred only fully retired in 2015 [at the age of 84]. He loved what he did and saw his work as a computer scientist as the outworking of God's call on his life. Fred used to say, if you enjoy what you do, why would you want to stop doing it,

especially if it was what God had created and called you to do?[64]

I wonder what Brooks might have thought of Buechner's advice about choosing a vocation, quoted at the beginning of this essay: "In a world where there is so much drudgery … our gladness in our work is as much needed as we ourselves need to be glad."[65] Reading Tarlton and others' tributes, it seems Brooks may have been just the sort of person Buechner had in mind. To find gladness in one's work, to share that gladness in a public way, as Brooks did, and to recognize joyful work as a gift from God—surely this meets a deep need in our world.

Notes

[1]Frederick Buechner, *Secrets in the Dark: A Life in Sermons* (HarperCollins, 2006), 39–40.

[2]Scott Waalkes points out that ideas attributed to Buechner are easily secularized: "just do what you love," "follow your passion," etc. One's "calling" can be used to rationalize selfish or unrealistic choices, to make an idol of the ideal career, or to denigrate others who settle for "just a job" and a paycheck. I agree, but I don't think this means we should reject Buechner's ideas. Instead, our definition of "gladness" (and "joyful work," in this article) should be grounded in a Christian understanding of the human person, created in the image of God. See Scott Waalkes, "Rethinking Work as Vocation: From Protestant Advice to Gospel Corrective," *Christian Scholar's Review* 44, no. 2 (2015): 135–53, https://christianscholars.com/rethinking-work-as-vocation-from-protestant-advice-to-gospel-corrective/.

[3]Frederick Buechner, *Wishful Thinking: A Theological ABC* (Harper & Row, 1973), 95.

[4]Katherine W. McCain and Laura J. Salvucci, "How Influential Is Brooks' Law? A Longitudinal Citation Context Analysis of Frederick Brooks' *The Mythical Man-Month*," *Journal of Information Science* 32, no. 3 (2006): 277–95, https://doi.org/10.1177/0165551506064397.

[5]Frederick P. Brooks, *The Mythical Man-Month: Essays on Software Engineering, [20th] Anniversary Edition* (Addison-Wesley, 1995), 7.

[6]The word *tractable* is used by computer scientists to describe problems for which it is possible to design an efficient solution algorithm. Tractability, in this sense, might be defined as "technical feasibility," as suggested by a reviewer of a draft of this article. But here I am using "tractability" as I understand Brooks to use it, to describe a medium through which one is enabled to express complex ideas, not limited by the medium but by one's imagination.

[7]A reviewer of a draft of this article said they were reminded "of a former colleague in mathematics, who said that with mathematical theorems, 'by God, you know when you've got it right!'"

[8]Brooks, *The Mythical Man-Month,* xi; and Tracy Kidder says of IBM's announcement that the 360 computers would provide a range of cost vs. performance options but all would run the same system software:

> In the commerce of computers, no single event has had wider significance, except for the invention of the transistor. Part of the 360's importance lay in the fact that all the machines in the family were software compatible. It cost IBM a true fortune and no end of trouble and anxiety to create system software for the 360 line. But all the machines in the family used that same [system] software … Any user program that worked on one machine in the family worked on all of them. (*The Soul of a New Machine* [Atlantic–Little, Brown, 1981], 42–43)

[9]Brooks, *The Mythical Man-Month,* 16–19.

[10]Brooks, *The Mythical Man-Month*, 25**.** "Brooks's Law" is apparently now so well known that no citation is necessary, as indicated by Bertrand Meyer's 2020 *Communications* article and Michael Ayres subsequent letter to the editor. See Bertrand Meyer, "In Search of the Shortest Possible Schedule," *Communications of the ACM* 63, no. 1 (2020): 8–9, https://cacm.acm.org/blogcacm/in-search-of-the-shortest-possible-schedule; and Michael Ayres, "Where Good Software Management Begins," *Communications of the ACM* 63, no. 3 (2020): 7, https://dl.acm.org/doi/pdf/10.1145/3385399.

[11]Andrey P. Ershov, "Aesthetics and the Human Factor in Programming," *Communications of the ACM* 15, no. 7 (1972), https://softpanorama.org/Articles/Ershov/aesthetics_and_the_human_factor_in_programming_ershov1972.shtml.

[12]Brooks, *The Mythical Man-Month*, 7.

[13]Brooks, *The Mythical Man-Month,* 7.

[14]Brooks, *The Mythical Man-Month,* 7.

[15]Brooks, *The Mythical Man-Month,* 7.

[16]Frederick P. Brooks, "No Silver Bullet: Essence and Accident in Software Engineering," in *The Mythical Man-Month: Essays on Software Engineering, [20th] Anniversary Edition* (Addison-Wesley, 1995), 184.

[17]Brooks, *The Mythical Man-Month,* 9.

[18]Brooks, *The Mythical Man-Month,* 9.

[19]Brooks, *The Mythical Man-Month,* 8.

[20]Brooks, *The Mythical Man-Month,* 15.

[21]Brooks, *The Mythical Man-Month,* 18.

[22]Brooks, *The Mythical Man-Month,* 15.

[23]Brooks, *The Mythical Man-Month,* 74–75.

[24]Frederick P. Brooks, "The Mythical Man-Month After 20 Years," in *The Mythical Man-Month: Essays on Software Engineering, [20th] Anniversary Edition* (Addison-Wesley, 1995), 255–56.

[25]Brooks references Blaauw in support of this idea, which by now is, effectively, dogma among software developers, perhaps in part because of Brooks's persuasive presentation of it in *The Mythical Man-Month*. See Gerrit A. Blaauw, "Hardware Requirements for the Fourth Generation," in *Fourth Generation Computers* (Prentice-Hall, 1970).

[26]Brooks, *The Mythical Man-Month*, 44. The phrase "single user's point of view" should not be taken to refer to any particular user or type of user. The system may have been built by a group working together but needs to be designed so that it is possible for an individual to make sense of it. If it is designed well, diverse individuals will be able to make sense of it, but each will need to make sense of it as an individual.

[27]Brooks, *The Mythical Man-Month,* 32. Brooks credits this idea to Harlan Mills, "Chief Programmer Teams, Principles, and Procedures," IBM Corporation, Report No. FSC 71–5108 (IBM Federal Systems Division, 1971).

[28]The "surgical team" is one of Brooks's ideas sometimes considered outdated today, because specific roles he suggests have greatly changed or are no longer needed. But developers today still make a clear distinction between architecture and implementation, and teams are still divided into distinct roles. An experienced senior developer would be like Brooks's "surgeon"—although the number of senior developers on a team would not be strictly limited to one, as in

Brooks's "surgical" model. Others in supporting roles would be responsible for quality assurance, user experience, deployment and IT infrastructure, etc.

[29]Matthew Crawford makes a distinction between the validation of the marketplace, which assigns monetary value based on utility, and the validation of "peculiar excellence" in one's work, which can only be recognized and appreciated by a peer, someone who understands the craft. See Matthew B. Crawford, *The World Beyond Your Head: On Becoming an Individual in an Age of Distraction* (Farrar, Straus and Giroux, 2015), 159.

[30]Brooks, *The Mythical Man-Month,* 46.

[31]Brooks, *The Mythical Man-Month,* 55.

[32]Brooks, *The Mythical Man-Month,* 47.

[33]Brooks, *The Mythical Man-Month,* 115–23.

[34]Brooks, "The Mythical Man-Month After 20 Years," 267.

[35]Brooks, *The Mythical Man-Month,* 118.

[36]Brooks, *The Mythical Man-Month,* 119–20.

[37]Brooks, *The Mythical Man-Month,* 157.

[38]Brooks, *The Mythical Man-Month,* 119.

[39]Eric S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary* (O'Reilly, 2001). This book is about open-source software as a part of "hacker" culture in the 1990s. Raymond's observations do not apply to all developers of open-source software as it exists today, within various management structures and business models.

[40]Raymond, *The Cathedral and the Bazaar*, 73.

[41]Raymond, *The Cathedral and the Bazaar*, 23.

[42]Raymond uses these terms as a sociologist would. "Prestige" is not meant in a dismissive or derogatory way; it simply refers to status or recognition within the "gift culture" community—a "good reputation among one's peers." If you share your work with the community, and others show their appreciation by using it or contributing to it so that your recognition within the community increases, that is your reward.

[43]Raymond, *The Cathedral and the Bazaar*, 72.

[44]Raymond makes a point of distinguishing the emergent open-source structure from Brooks's "surgical team" structure as described in *The Mythical Man-Month*. But Raymond's distinction is based primarily on specific roles in Brooks's description, which I have not considered in this article. See Raymond, *The Cathedral and the Bazaar*, 221.

[45]Brooks, *The Mythical Man-Month,* 16.

[46]Raymond, *The Cathedral and the Bazaar,* 108.

[47]Tom DeMarco and Timothy Lister, *Peopleware: Productive Projects and Teams* (Addison-Wesley, 2013). Brooks's endorsement is on the back cover.

[48]DeMarco and Lister, *Peopleware,* 7.

[49]Joel Spolsky, "Craftsmanship," in *Joel on Software … and on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity* (Apress, 2004), 119.

[50]I was eventually rewarded with a promotion to a full-time position in engineering. Initially, I was very happy with the change. The pay was higher, and this is what my electrical engineering degree was supposed to prepare me for. But I found the new job much more stressful and quit within a year, eventually going back to school for a degree in computer science.

[51]Brooks, *The Mythical Man-Month*, 7.

[52]Frederick P. Brooks, "The Computer Scientist as Toolsmith II," *Communications of the ACM* 39, no. 3 (1996): 63, https://www.cs.unc.edu/~brooks/Toolsmith-CACM.pdf.

[53]Brooks, "The Mythical Man-Month After 20 Years," 254. See also his later book, Frederick P. Brooks, *The Design of Design: Essays from a Computer Scientist* (Addison-Wesley, 2010), which considers the challenge of conceptual integrity more broadly and in much greater detail.

[54]For Borgmann, "technology" does not refer merely to tools of human making or to scientific knowledge applied to human ends, but to a cultural pattern: the ideal technological device is conveniently available and easy to use; it may be very complex internally and may depend on a complex and interconnected technological machinery working behind the scenes, but all of this is hidden from the user. We are surrounded by and dependent on a multitude of such devices. A smartphone is an obvious example, but other things we might not immediately think of as technological fit the pattern: homes with electricity and central heating, processed foods and their packaging, payment and banking systems, etc. See Albert Borgmann, *Technology and the Character of Contemporary Life: A Philosophical Inquiry* (University of Chicago Press, 1984).

[55]Borgmann, *Technology and the Character of Contemporary Life*, 115.

[56]Borgmann, *Technology and the Character of Contemporary Life*, 114.

[57]Borgmann is ambivalent about the focal (i.e., pretechnologically engaging) significance of working with computers. My own experience, however, confirms Brooks's claim that this work is joyful in its own unique way. See Borgmann, *Technology and the Character of Contemporary Life*, 217.

[58]Matthew Kam et al., "What Do Professional Software Developers Need to Know to Succeed in an Age of Artificial Intelligence?" paper presented at the 33rd ACM International Conference on the Foundations of Software Engineering, June 23–28, 2025, Trondheim, Norway, published in *FSE Companion '25: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering* (Association for Computing Machinery 2025), 947–58, https://doi.org/10.1145/3696630.3727251.

[59]According to a 2023 study, developers were happier and found their work more satisfying and meaningful when they used AI tools—perhaps this could be interpreted as the result of an increase in tractability? See Begum Karaci Deniz et al., "Unleashing Developer Productivity with Generative AI," McKinsey Digital, June 27, 2023, https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/unleashing-developer-productivity-with-generative-ai.

[60]The ever-increasing capability of AI tools to complete programming assignments presents a major challenge to computer science educators: How do we motivate students to learn to do things they know AI can do for them, so that they can eventually learn enough to do things AI can't do … and become employable?

[61]Brooks says, for example, "Of all my technical accomplishments, making the 8-bit byte decision is far and away the most important. And the reason was, it opened up the lowercase alphabet …" It would be hard to exaggerate the significance of this decision! See Fred Brooks (Frederick P. Brooks, Jr.), "Oral History of Fred Brooks," interview by Grady Booch, ed. Dag Spicer, recorded on September 16, 2007, Computer History Museum, Cambridge, UK, https://www.computerhistory.org/collections/catalog/102658255.

[62]Simson Garfinkel and Eugene Spafford quote Mary Whitton, "He was an incisive scholar, but also humble and kind"; Pat Hanrahan, "Fred was always highly positive and very supportive … he was deeply insightful … 'wise' in a broad sense

of the word"; and Ivan Sutherland, "Fred was a statesman in the best sense of the word, as well as a great leader, a very kind man, and a fine engineer." See Simson Garfinkel and Eugene H. Spafford, "In Memoriam: Frederick P. Brooks, Jr. 1931–2022," *Communications of the ACM* 66, no. 1 (January 1, 2023): 21–22, https://cacm.acm.org/news/in-memoriam-frederick-p-brooks-jr-1931-2022.

[63]Russ Tuck, for example, in "Computer Science: Creating in a Fallen World," references Brooks's "The Computer Scientist as Toolsmith II," cited earlier in this article. Many Christians in computer science point to *Toolsmith* as an inspiration. I first read it as a graduate student 25 years ago; I have recommended it to others, including my own students, many times since. See Russell Tuck, "Computer Science: Creating in a Fallen World," *ACMS Conference Proceedings 2019*, 159–71, https://pillars.taylor.edu/acms-2019/21.

[64]Hank Tarlton, *A Tribute to Fred Brooks, Computer Scientist and Christian*, https://blog.emergingscholars.org/2022/12/a-tribute-to-fred-brooks-computer-scientist-and-christian.

[65]Frederick Buechner, *Secrets in the Dark: A Life in Sermons* (HarperCollins, 2006), 39–40.