YOU "NEED A FARMER, A BUILDER, AND A WEAVER …"

The Agile Enterprise and the Division of Labor

by Gene Callahan

Libertarian activist Leonard E. Read began his famous essay "I, Pencil" by noting:

> I, Pencil, simple though I appear to be, merit your wonder and awe, a claim I shall attempt to prove… Simple? Yet, not a single person on the face of this earth knows how to make me. This sounds fantastic, doesn't it? Especially when it is realized that there are about one and one-half billion of my kind produced in the USA each year.[1]

Read goes on to list just a few of the many, many people who contribute to the making of a "simple" pencil: loggers, miners, makers of chain saws, hemp growers, the manufacturers of railroads and railroad cars, millworkers, producers of precision assembly line machines, the harvesters of canola seed, farmers growing castor beans, and so on.

Enter the Agile of Specialists

What Read praises in his essay are the benefits of the division of labor, the economic process through which a human community, by dividing up tasks and "assigning" various members to specialize in each task, can greatly increase its output. (I put "assigning" in scare quotes because, in a market economy, for the most part people are not literally *assigned* to tasks, but instead choose their roles in the division of labor based upon their talents and the prevailing compensation for each possible role they could fill.) The benefits of the division of labor were, of course, recognized at least as far back as Plato and Xenophon. As Plato put it in *The Republic*, "Well then, how will our state supply these (physical) needs? It will need a farmer, a builder, and a weaver, and also, I think, a shoemaker and one or two others to provide for our bodily needs. So that the minimum state would consist of four or five men."[2] Adam Smith famously expounded upon those benefits in *The Wealth of Nations*, writing "The greatest improvement in the productive powers of labour, and the greater part of the skill, dexterity, and judgment with which it is anywhere directed, or applied, seem to have been the effects of the division of labour."[3]

Smith goes on to describe the production of pins, a task at which a single person, not specialized at the task, "could scarce, perhaps, with his utmost industry, make one pin in a day, and certainly could not make twenty." But when 10 workers took on specialized tasks, with the help of specialized machinery, Smith asserted that although "they were very poor, and therefore but indifferently accommodated with the necessary machinery, they could, when they exerted themselves, make among them about twelve pounds of pins in a day," with the result that each worker produced several thousand times the number of pins per day as would have been possible without the division of labor.

In the early 20th century, this method of increasing productivity was pushed to its limits. Tasks were broken down to the extent that workers with minimal skills could be assigned simple, highly repetitive actions and perform them with almost no knowledge of what anyone else on the assembly line was up to. Although this led to higher output of standardized products, the disadvantages of extending the division of labor to this extent were not overlooked. Karl Marx noted that the extensive division of labor alienated the worker from the product being produced: people who spend all day tightening a particular lug nut may be little able to associate what they do with "making a car." But even Smith, who, as we have seen, praised the effects of the division of labor, commented in *The Wealth of Nations*:

> In the progress of the division of labour, the employment of the far greater part of those who live by labour, that is, of the great body of people, comes to be confined to a few very simple operations, frequently to one or two. But the understandings of the greater part of men are necessarily formed by their ordinary employments. The man whose whole life is spent in performing a few simple operations, of which the effects are perhaps always the same, or very nearly the same, has no occasion to exert his understanding or to exercise his invention in finding out expedients for removing difficulties which never occur. He naturally loses, therefore, the habit of such exertion, and generally becomes as stupid and ignorant as it is possible for a human creature to become.

Smith is pointing out a *general* problem with the extensive division of labor, but there is a more *particular* problem, which only came to prominence in the recent days of increasing automation and increasing demand for innovative and customized products: the sort of mindless production line division of tasks common in mid-20th-century factories created a workforce downright discouraged from thinking about how their work fit into the production process as a whole, or how alterations in parts they did not make themselves might affect their own task. Such a holistic view was only supposed to be required of the engineers who designed new products or who designed the factory processes that would produce those new products. As in a planned socialist economy, all knowledge about the product and the production process would be concentrated at the top of a pyramid of work, and those below the peak were merely to follow the orders of those knowledge commissars.

No knowledge worker producing an even moderately complex product can do his work properly without an understanding of his part in the production process.

But Specialists Aren't Sufficient Anymore

A major problem with this approach is that as products become more complicated and the pace of innovation increases, no single mind, or even a small group of minds, is capable of grasping all the interconnections between the different parts of those complex products, and thus cannot foresee how an innovation supposedly concerning only one part will actually have ripple effects on many other, apparently separate, production tasks. This fact was realized quite early at Toyota and led to the invention of the Toyota Production System,[4] the forerunner of Lean software development. As Mary and Tom Poppendieck note in their book *Implementing Lean Software Development*:

> Toyota's real innovation is its ability to harness the intellect of "ordinary" employees. Successful lean initiatives must be based first and foremost on a deep respect for every person in the company, especially the "ordinary" people who make the product or pound out the code.[5]

As important as these ideas were in factory production, their importance is even greater in the world of software development, where production is *always* the production of a novel product; otherwise, one would simply buy or rent an existing software product, which is almost always a lower cost venture than "rolling your own."

In such an environment, it is simply not possible to assign the "workers" (programmers) a simple, repetitive task and expect them to achieve decent results without at least some understanding of the overall product design, as well as an understanding of how their particular "part" integrates with the other parts of the product as a whole. In such a situation, worker obedience no longer "works." A manager cannot tell a software engineer working on a product of even moderate complexity to just follow the manager's orders; the programmer can bring production to a halt simply by asking, "OK, what line of code should I write next?"

We Must Be Enterprise-Aware

But further: no knowledge worker producing an even moderately complex product can do his work properly without an understanding of his part in the production process via continuous interaction with the evolving understanding of all the other knowledge workers producing the product. One such worker gaining a better understanding of the nature of her component simply must convey that understanding to all other workers upon whom the changes in her component have an impact, and that set of workers typically encompasses almost everyone working on the product. As the Disciplined Agile framework states:

> Enterprise awareness is one of the key principles behind the Disciplined Agile (DA) framework. The observation is that DA teams work within your organization's enterprise ecosystem, as do all other teams. There are often existing systems currently in production and minimally your solution shouldn't impact them. Better yet your solution will hopefully leverage existing functionality and data available in production. You will often have other teams working in parallel to your team, and you may wish to take advantage of a portion of what they're doing and vice versa. Your organization may be working towards business or technical visions which your team should contribute to. A governance strategy exists which hopefully enhances what your team is doing.[6]

The various aspects of Agile/Lean/DevOps production follow from recognizing the real situation of knowledge workers cooperating to create innovative products. Programmers cannot do their jobs in isolation; thus, we need the practice of *continuous integration*, which quickly exposes mutual misunderstandings of how one person's work impacts that of others. Testers cannot test successfully without introducing large delays in deployment, unless they are part of the production process from day one; thus, we must employ *continuous testing*, guaranteeing that product flaws are exposed and fixed at the earliest moment possible. Operations cannot successfully deploy constantly evolving products unless deployment itself becomes a software product capable of evolving as fast as the products of the developers: thus, we must view *software as infrastructure*. The "business" stakeholders in the product cannot ensure the product is really meeting business needs unless they are continually engaged in the development process: thus, me must engage in *continual interaction* between the engineers and the "business people." How new versions of a piece of software impact the end users cannot be determined without continual feedback from those users: thus, we need *incremental development*, where developers work on small batches and can easily change course; *continuous deployment*, where end users can comment on the work done in those small batches; and *continuous monitoring*, so that any problems using the product become known almost as soon as they occur.

We Also Need a Wider Range of Skills

Given the above realities, a rigid division of labor hinders businesses from responding agilely to changing market conditions with new programs or new features added to existing programs. If workers are confined to narrow silos based on their job title, the interaction between the many components of a complex piece of software must be defined from the top down, and this restriction will result in a very limited capacity to deviate from an initially defined pattern of interaction. The DA framework notes that:

> IT departments are complex adaptive organizations. What we mean by that is that the actions of one team will affect the actions of another team, and so on and so on. For example, the way that your Agile delivery team works will have an effect on, and be affected by, any other team that you interact with. If you're working with your operations teams, perhaps as part of your overall DevOps strategy, then each of those teams will need to adapt the way they work to collaborate effectively with one another. Each team will hopefully learn from the other and improve the way that they work.[7]

The various aspects of Agile/Lean/DevOps production follow from recognizing the real situation of knowledge workers cooperating to create innovative products.

Let's consider a realistic change that might hit a project midstream, and just a few areas it might impact.

I was once developing an option trading package for a team of traders. At first, we were only getting quotes for options from a single exchange. The traders realized that they wanted instead to see the best bid and ask from every exchange, which meant we needed to receive quotes from four exchanges, not one. This might seem to be a specification change with a narrow scope: just add three more price feeds to the application. Who would this concern beyond the programmer who would be adding the feature?

Well, for one, it would concern the team supporting the price server: this was going to quadruple the load this application would place on it. It was also going to impact the order server: that server had to be prepared to send orders out to the proper exchanges. Oh, and the testing team had better be prepared to simulate quotes coming in from four sources, not one. Moreover, the monitoring team would have to detect if there was a lag on quotes arriving from any of those four sources.

Or consider the patterns and tales from Michael Nygard's book *Release It!*[8] Continually, in Nygard's stories, solving a problem in a sophisticated Web operation involves a wide range of both technical and business knowledge. For instance, in terms of designing "circuit breakers" that limit the impact of the failure of one component, Nygard notes that deciding what to do when a circuit breaker trips is not merely a technical decision but rather involves a deep understanding of business processes: "Should a retail system accept an order if it can't confirm availability of the customer's items? What about if it can't verify the customer's credit card or shipping address?" Later in the book, Nygard discusses the example of a retail system that went down entirely on Black Friday, costing his client about a

million dollars an hour in sales. Fixing the problem involved understanding the functioning of the front end of the online store, the order management system, and the scheduling system, along with the interactions of all three.

The best bet to successfully respond to this changed business requirement is for the people working in each specialization to have a vision of the overall system, an understanding of how other specialized areas function, along with robust communication channels.

Enter the Age of Generalizing Specialists

A software engineer who thinks of his job narrowly, as just being responsible for writing the code to do the task he is told the code should do, is not going to be thinking of the multiple other areas any change in his task would affect. And a higher-level designer is unlikely to know enough of the details of all these areas to fully understand the impact of such a change. The best bet to successfully respond to this changed business requirement is for the people working in each specialization to have a vision of the overall system, an understanding of how other specialized areas function, along with robust communication channels open between the various specialties; in other words, to break down the silo walls produced by a rigid division of labor and embrace Agile development principles. Or, as Cutter Fellow and this issue's Guest Editor Scott Ambler — who calls people able to understand multiple aspects of the system being built "generalizing specialists" — puts it:

> A generalizing specialist is someone with a good grasp of how everything fits together. As a result they will typically have a greater understanding and appreciation of what their teammates are working on. They are willing to listen to and work with their teammates because they know that they'll likely learn something new. Specialists, on the other hand, often don't have the background to appreciate what other specialists are doing, often look down on that other work, and often aren't as willing to cooperate. Specialists, by their very nature, can become a barrier to communication within your team. Another challenge with specialists is that they have difficulty working together effectively with others because they don't have the background to understand the issues that the others are trying to deal with.[9]

An organization seeking to become Agile should therefore look to have a preponderance of generalizing specialists on their teams. There is, of course, room for some pure specialists, but too often businesses seek to hire a Python or Linux or Docker guru, when what they really need is someone "good enough" at one of those specialties but who also has broader technological and business understandings. Certainly, hiring a generalizing specialist may have some short-term downside in terms of cranking out the next couple of specialized projects, but most often that cost will be repaid several times over because it will create more cohesive and Agile teams in the long run.

Endnotes

[1]Read, Leonard E. "I, Pencil: My Family Tree as Told to Leonard E. Read." *The Freeman*, 1958 (http://www.econlib.org/library/Essays/rdPncl1.html).

[2]Plato. *The Republic.* The Internet Classic Archives (http://classics.mit.edu/Plato/republic.html).

[3]Smith, Adam. *An Inquiry into the Nature and Causes of the Wealth of Nations*. Compiled by Edwin Cannon. Methuen & Co., Ltd., 1904 (http://www.econlib.org/library/Smith/smWNCover.html).

[4]"Toyota Production System." Wikipedia (https://en.wikipedia.org/wiki/Toyota_Production_System).

[5]Poppendieck, Mary, and Tom Poppendieck. *Implementing Lean Software Development: From Concept to Cash.* Addison-Wesley Professional, 2006.

[6]"Principle: Enterprise Awareness." The Disciplined Agile (DA) Framework (http://www.disciplinedagiledelivery.com/enterpriseawareness/).

[7]"Principle: Enterprise Awareness." (see 6).

[8]Nygard, Michael T. *Release It! Design and Deploy Production-Ready Software.* 2nd edition. Pragmatic Bookshelf, 2018.

[9]Ambler, Scott W. "Generalizing Specialists: Improving Your IT Career Skills." Agile Modeling (http://www.agilemodeling.com/essays/generalizingSpecialists.html).

Gene Callahan is an Industry Associate Professor of Computer Science at NYU's Tandon School of Engineering, where he teaches one of the first DevOps courses in the US. Dr. Callahan worked for two decades as a software engineer and manager. He is the author of Economics for Real People *and* Oakeshott on Rome and America. *Dr. Callahan holds a master's degree from the London School of Economics and Political Science and a PhD from Cardiff University, Wales. He can be reached at ejc369@nyu.edu.*