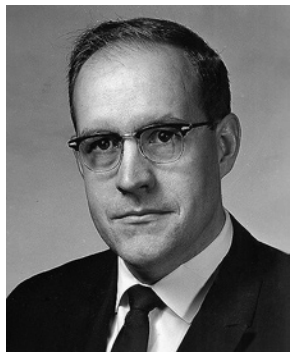
By Michael Schubach

Copyright 2010 Hospitality Upgrade
No reproduction or distribution without permission.

HAPPY ANNIVERSARY, MYTHICAL MAN



Dr. Frederick P. Brooks

I always hate it when I forget to send a card.

This year is the 35 anniversary of a landmark work in computer engineering, a small but mighty collection of essays on software manufacture entitled The Mythical Man-Month by Dr. Frederick P. Brooks, Jr. Dr. Brooks chronicles his experiences at IBM as he and his team sought to unravel (or, perhaps more accurately, ravel) the intricacies of the OS/360, IBM's operating system software for its newest generation machines. His results there were not stellar: the initial software deliveries were (and see if any of this sounds familiar) late, over budget and did not perform according to the specification. After his departure from IBM, Brooks reflected on his experiences there, looking for the management lesson to be learned. He focused on the fact that his team's results (system software) were not nearly as successful as the team that worked on the System 360 hardware. Brooks' contribution is a volume that contains many of the answers to a question that still haunts us today: why is software so freakin' hard to manage? Somewhere between an epic love story and a tear-jerker of a tragedy, his tale is of the innate contradiction that is the art of software construction.

And speaking of contradiction, let's start right out with the first one. Why would anyone in their right mind recommend a 35-year-old book about software engineering? Could the story of a process hewn in the misty vapors of antiquity when our simple ancestors, clad only in animal-skin loincloths (Author's note: Okay, 35 years ago puts us back to bell-bottom trousers and paisley neckties but remember, these guys worked for IBM) pounding primitive operating systems out of zeroes and ones possibly teach us smart folk anything today? Oddly enough, it turns out that it could.

Dr. Brook's first premise of complexity is that programs and programming products are two very different things. A program is a set of machine instructions that works to produce an end result of some sort – a calculation, a stored value or a repetitious and annoying e-mail campaign. A programming product is something written for sale or public use and is a far more complicated object than a simple program that could accomplish the same task. A programming product must be generalized for its community of users, allowing inputs with degrees of commonality but that can differ on a case-by-case basis like, oh let's say, maybe… room types.

Remember those nasty little inventory units that have never been the same any two of the world's million or so hotels? So programmers must generalize their code, test it, debug it and make it pretty, intuitive, error-free, idiot-proof and available at amazing low prices. Dr. Brooks estimates that the step from program to programming product takes three times as much effort (read: cost) as the program itself does.

Now consider a Programming Systems Product. A PSP is a series of interconnected programming products that interoperate by means of interface or integration. Again, Dr. Brooks estimates that the minimum requirement for a PSP (depending of the scope and depth of the offering) is three times that of a programming product. So what you buy from a software vendor takes a minimum of nine times the effort to create as the baseline program that solves the problem in the first place. It's not whining – it the nature of the beast… and the reason why the price is amazing but not because it's low.

Besides the complexity of the task, Dr. Brooks considers the process itself, hence the title of the book. Software projects are estimated in man-months, the units of programmer time required to create the finished product. (Another author's note: I hear those teeth grinding over the gender bias inherent in a term like "man-month." Remember that Dr. Brooks wrote his treatise in simpler times, back when we didn't have any issues with gender discrimination in technology companies because they only hired guys as programmers. I just know in my heart that Dr. Brooks means to be totally inclusive and we should just accept this archaic term without prejudice or reservation, just as we have accepted the removal of gender bias from such organizations as the Dallas Cow-persons.)

What else is wrong with man-months? They simply aren't interchangeable units–a month of one person's work is not the same as another person's. Further, a month of one person's work on one project or in one discipline is not the same as that same person's work in another area. For instance, there is me, writing an article (a man-month), me, filing paperwork (a man-year) and me, actually making home improvements with tools and other hardware-related things (a man-decade). Dr. Brooks reminds us that the imprecise tools with which we estimate software projects

contribute to imprecise estimates. And where the estimates are faulty, late and over budget are not far behind.

My personal favorite in the Brooksism is his assertion that the adding programmers to a project will actually cause it to slow down rather than speed up. Maybe that seems obvious, like too many cooks spoiling the broth. But we're living in world that's been unduly inspired by Henry Ford; we tend to think of manufacturing as a series of small, linear tasks that can be compartmentalized and rapidly executed. We think mathematically, and things that make sense to us should apply: if one programmer can write 100 lines of code in a day then two can write 200 and so on. Divide and conquer – you write this part and I'll write that part and we'll put it all together at the end of the day.

Ummm… no. Here software manufacturing takes a serious turn away from hardware manufacturing: matching up tabs and slots is nothing like matching up concepts, intuition and thought processes. It would be like a team of talented authors writing a novel from a detailed outline – it's not that it can't be done but each collaboration takes much longer than an artist working alone. We have to be a little careful here – Dr. Brooks is not saying that the most productive software manufacturing environment consists of only one programmer. But he is cautioning us: for our purposes 1 + 1 = 1.6. Eventually that 1.6 will grow to a 2, and in special circumstances sometimes you can get it up to 2.1 or 2.2. However, when you add the next resource, 2 + 1 is more like a 2.4. Many hands may make light work but many brains? Not so much.

And yet another complexity looms. The non-linear process of building software often leads us to mistaken conclusions about when a project will finally finish. That one allocates five of those error-prone man-months to a project doesn't mean that 20 percent of the code is completed as each man-month is rendered. In fact, it's common for projects to start slow and come together suddenly near conclusion and for the fertile imaginations of the software engineers who know this to believe the best about their challenged projects. Add those two factors together and what you get is an engineer who has completed four man-months and 30 percent of the code with an estimate for on-time completion. Bad news: sometimes they make it and other times they aren't correct to the nearest year. What we do know is that the earlier you estimate the project the more incorrect you are going to be. What this adds to the mix is that horrible understanding that you aren't much smarter as you near the end.

Once you start to wrap your head around the complexities of being complex, you begin to understand how valuable Dr. Brooks' insights really are. The Mythical Man-Month can be a challenging read, particularly when you pass by some of those archaic moments. (Author's note: Back off and give the guy a break – the book is 35 years old!) Nonetheless, serious management lessons are there for the learning.

I would say that a card is in order, wouldn't you?

MICHAEL SCHUBACH *has been in the industry at least as long as the mythical man, and a long time contributor to* Hospitality Upgrade. *He can be reached at michaelschubach@mac.com.*