Technology Can Only Do So Much: Why the Human Factor Continues to Frustrate Software Developers

"Adding manpower to a late software project makes it later."

In 1975, Fred Brooks reminded us there are finite limits to our ability to compress the development process. Moreover, throwing people onto troubled software projects often backfires. These insights should not have surprised us; after all, time and effort are hardly fungible commodities. Even with the best tools and methods, nine women still can't deliver a baby in one month.

But if Brooks merely reminded people of what they already suspected, why do so many software projects still come in late and over budget? A recent study of the QSM database showed that large projects (defined as over 50,000 ESLOC) have only a 19% chance of meeting their planned schedules and a 30% probability of making their budgeted effort. After thirty years of technological change and process improvement effort it's discouraging to see organizations still struggling with the same old problems. Why does this still happen so frequently? More importantly, what can we do to overcome these problems?

Tools and Methods Improve, But People Remain All Too Human

Part of the problem is that while technology has changed rapidly, human nature remains constant. A critical ingredient in software development - *perhaps the critical ingredient* - is people. This is an insight technical managers sometimes forget to factor into their plans.

Tools and methods allow us to do things more efficiently, but software development remains a uniquely human endeavor. Consequently, successful project management requires a mastery of both people and technical skills. The first part of this paper deals with the human factors that trip up so many software projects. The latter part brings data to the problem solving table.

The people problems that plague software teams tend to involve over-optimism, fear of measurement, and using the wrong tools for the job. They fall into three broad categories:

The Triumph of Hope over Experience:

- **Competitive pressure**. Bid solicitation (especially in the outsourcing world) involves a great deal of internal pressure on participants to win business. This competitive 'tunnel vision' often leads to *overly optimistic assumptions that ignore an organization's proven ability to deliver software*.

- **Unfounded productivity assumptions**. If it has always taken 20 hours to produce a widget, assembling a crack team of developers will not cut that number to 10. *Productivity improvement is a long-term endeavor; not a short term fix.*

Fear of Measurement:

- **Not learning from history**. Companies which measure projects *well* develop organizational self-knowledge, identify capacities and patterns, and come to know their strengths and weaknesses*. In short, they learn from experience and develop an empirical basis for project planning.* Unfortunately, most organizations lack formal software measurement and evaluation capacity or measure and plan haphazardly. Lacking self-knowledge, these organizations continually put themselves at risk.

- **Not planning for growth**. The *planned* project generally differs from the *delivered* project in one key component*: it is smaller and delivers less functionality.* Good project management and effective change control help mitigate scope creep, but a recent QSM study showed median size growth of about 20%. *Projects locked into budgets and schedules based on one set of requirements will be sorely pressed to meet these commitments when the requirements increase.*

- **Not watching where we're going**. Most software teams work hard and want to succeed. There is an admirable human tendency to double one's efforts when problems arise. Such industry should be encouraged, but *Herculean effort makes a poor substitute for timely, gentle course corrections.* In fact, *it is usually too late to take effective countermeasures when problems finally manifest themselves*.

Applying the Wrong BandAid:

- **Ineffective or inappropriate countermeasures**. There are only three possible courses of action when a project threatens to exceed budget or schedule. Each works within a limited range of possibility and carries accompanying cost.

    - **Relaxing the schedule:** Results in a less expensive project with fewer defects. There are good and bad reasons why this option is not used more often. Legal or contractual requirements may mandate delivery by a certain date; late delivery may invoke penalties or loss of customer goodwill. Also, organizations may have committed project staff to other endeavors. The bad reasons center more on reluctance to change and unwillingness to "lose face".

    - **Reduce the scope of the delivery**. Deferring non-critical functionality until a later release (or eliminating it entirely) can keep a project within time and cost constraints. The cost is obvious: less is delivered than was promised or expected.

    - **Add staff**. Within a narrow range, adding staff can reduce schedule, albeit slightly and at considerable cost. As many managers have discovered, schedule/effort tradeoff is non-linear: a single unit of schedule reduction "costs" many units of effort and this ratio increases exponentially as the schedule is compressed.

Challenging the Conventional Wisdom

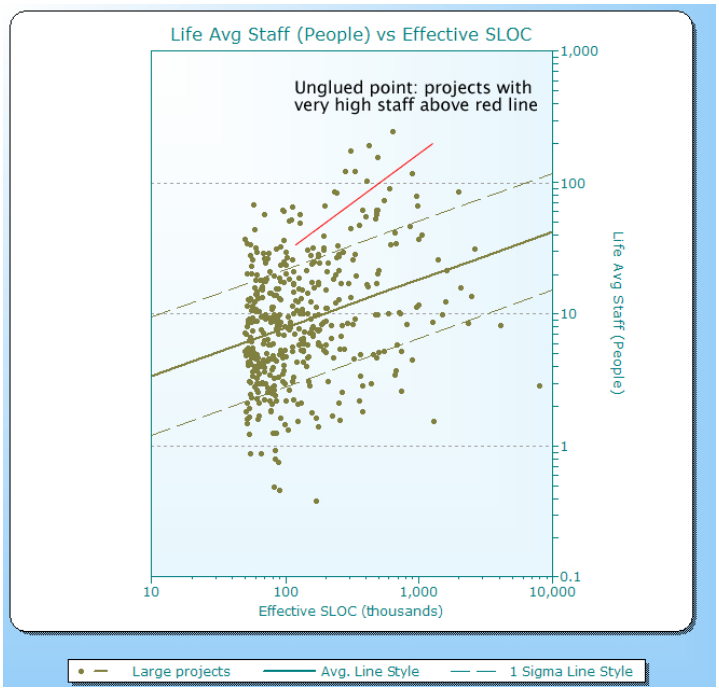
So, what are harried software managers to do when faced with non-linear relationships between time and effort, technology that changes constantly, and human behaviors that, despite experience, remain stubbornly entrenched? This is where measurement is invaluable. Having a good metrics program in place tells organizations several important things: what they have built in the past, what their historical capabilities are, and which patterns in the data may be helpful in the future. A good metrics program does one more thing: *armed with a good historical baseline, managers can monitor their progress and make timely course corrections as projects unfold.* For managers who need to assess the risks/benefits of using new technologies in real time, this kind of feedback is priceless.

As technology continues to shift the productivity curve outward, managers are tempted to challenge the conventional wisdom. The allure of Agile programming may make them wonder if it isn't possible, after all, to make that baby in *one* month instead of nine. This is not necessarily a bad thing. As new tools and methods appear it makes sense to reexamine old assumptions about the relationships between time, effort, and productivity. But that reexamination should be grounded in empirical methods and hard data, not pie in the sky optimism.

Take Fred Brooks' famous maxim, "*Adding manpower to a late software project makes it later".* QSM researchers have found a strong correlation between project size and most other metrics. In our experience, the non-linear relationships between size, time, effort, and defects often make simple rules of thumb less than universally applicable. In practice, these tried-and-truisms often hold true for many, if not most projects but since many software relationships 'go exponential' at certain points along the size spectrum, it's probably not a bad idea to test them against the data.

"Adding Manpower to a Late Project Makes It …"

We looked at large Information Technology software projects completed in the last decade to answer the question, "Just how does the 'mega staff' strategy affect large projects?" On a scatter plot of effective (new and modified) size vs. average staff we found an interesting separation in projects at the high end of the staffing curve. We call this gap the "Unglued Point": where staffing runs wild.



Below 100,000 lines of code, the projects are evenly distributed. But beginning at the 100 K ESLOC mark, a hole opens up, separating the bulk

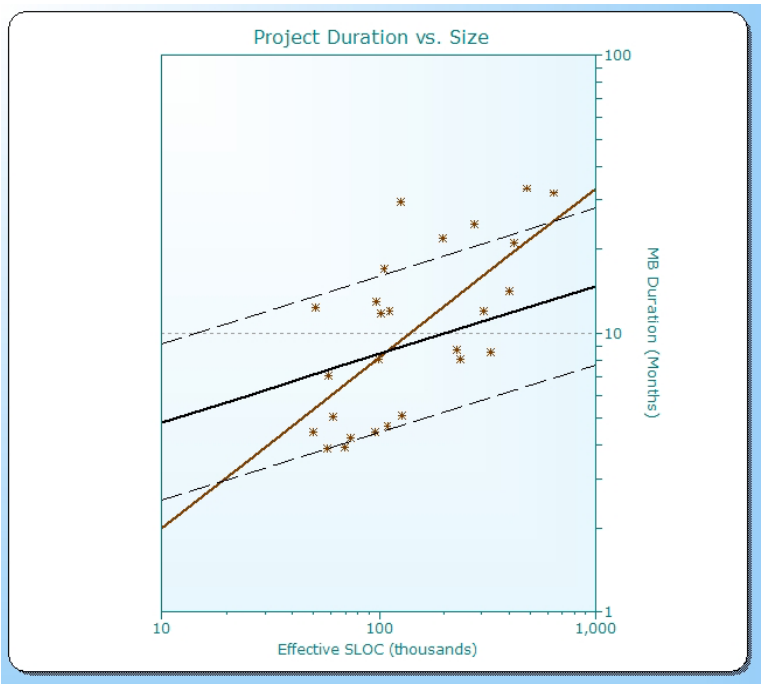
of these projects from those staffed at far higher levels.

The trend lines in the first chart are average, plus, and minus one standard deviation lines. At any point on the size spectrum, there is wide range of staffing strategies. Above the range of 'normal' variability is the unglued point, representing projects with exceptionally high staffing. The high staff projects position well above the +1 standard deviation line, placing them over the 68th percentile, closer to the 75th percentile or above.

What can these high staff projects tell us? How do their schedules compare with other, more reasonably staffed projects? How does the high staff strategy impact project quality? And of course, what are the cost implications of such a strategy?

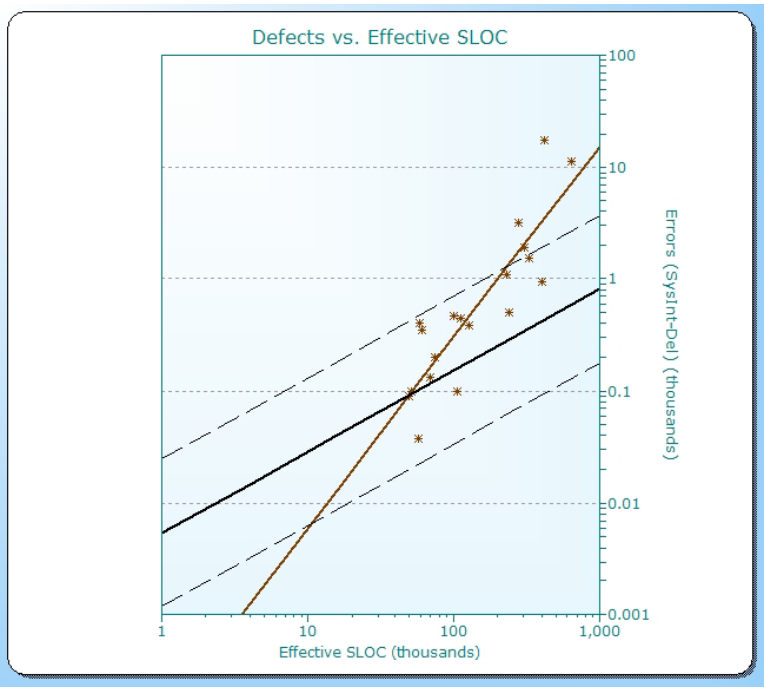
Let's find out.

The second graph displays only projects above the unglued point for staffing. The parallel lines show average, plus and minus 1 σ trend lines for "reasonably staffed" projects. Crossing these diagonally is the trend from the high staffed projects shown. For projects up to 100,000 lines of code, using large teams seems to deliver projects at or below the QSM average for schedule**.**



However, matters deteriorate rapidly as projects increase in size. At best, aggressive staffing may keep a project's schedule within the normal range of variability but this strategy becomes increasingly ineffective as project size increases.

What about quality? Again, only high staff projects are shown. The steeply sloped line crossing the QSM defect trend lines is the average of the mega-staffed projects. **Their quality is consistently worse than average (higher defect density) and increases precipitously as the projects increase in size.** The impact of high staffing on project quality is clearly negative.



Finally, what are the cost implications of the large team strategy? First let's review what is purchased in terms of schedule reduction: at best high staffing moves a project into the range of normal schedule variation, though this strategy becomes increasingly ineffective as projects increase in size. Overall project quality, which is its legacy to its users, is worse than normal. Now the cost: as the following table illustrates, high staffed projects are several times more expensive.

Average Project Cost at $10,000/Staff Month		
Project Size	Average Peak Staff	Unglued Staff
50k	$1,976,200	$5,551,100
100k	$2,974,600	$10,470,300
200k	$4,449,400	$20,200,000
500k	$7,799,200	$47,300,000
1M	$11,652,100	$87,900,000

Conclusion

So, how did Brooks' famous maxim hold up against the evidence? Does adding staff to a late project only make it later? It's hard to tell. **Large team projects, on the whole, did not take notably longer than average.** For small projects the strategy had some benefit, keeping deliveries at or below the industry average, but this advantage disappeared at the 100,000 line of code mark**.** At best, aggressive staffing may keep a project's schedule within the normal range of variability.

Contrary to Brooks' law, for large projects the more dramatic impacts of bulking up on staff showed up in quality and cost. Software systems developed using large teams had more defects than average, which would adversely affect customer satisfaction and, perhaps repeat business. **The cost was anywhere from 3 times greater than average for a 50,000 line of code system up to almost 8 times as large for a 1 million line of code system.** Overall, mega-staffing a project is a strategy with few tangible benefits that should be avoided unless you have a gun pointed at your head. One suspects some of these projects found themselves in that situation: between a rock and a hard place.

How do managers avoid these types of scenarios? Software development remains a tricky blend of people and technical skills, but having solid data at your fingertips and challenging the conventional wisdom *wisely* can help you avoid costly mistakes. Measurement allows you to manage both the technical and people challenges of software development with confidence whether you are negotiating achievable schedules based on your proven ability to deliver software, finding the optimal team size for that new project, planning for requirements growth, tracking your progress, or making timely mid-course corrections.

Kate Armel is a technical manager with Quantitative Software Management, Inc. She has 8 years of experience in technical writing, metrics research and analysis, and assisting Fortune 1000 firms estimate, track, and benchmark software projects. Ms. Armel was the chief editor and co-author of the QSM Software Almanac.