Adding Manpower

Adrian Kosmaczewski

2008-08-08

Published in 1975, “The Mythical Man-Month” is considered an all-time classic in the software engineering field. The book author, Frederick P. Brooks Jr., used his experience as the project manager of the IBM System/360 and its software, the Operating System/360, to explain a common set of problem patterns, applicable to other software projects as well.

One of the most famous citations in the book is the one regarding the consequences of adding human resources to a late project; this article will provide a couple of thoughts about this assertion, and highlight some contrariwise opinions.

The Mythical Man-Month

The second chapter of Brooks’ masterpiece is named exactly as the book, “The Mythical Man-Month”; the core argument of this chapter is that the most frequent factor of project failure is schedule and time estimation. Brooks states that this is due to the fact that

> Men and months are interchangeable commodities only when a task can be partitioned among many workers with no communication among them. This is true of reaping wheat or picking cotton; it is not even approximately true of systems programming. When a task cannot be partitioned because of sequential constraints, the application of more effort has no effect on the schedule. The bearing of a child takes nine months, no matter how many women are assigned.

(Brooks, pages 16 & 17)

The final phrase of the above paragraph is often used as a graphical depiction of the nature and meaning of Brooks’ law. It implies the strong need for communication and integration existing in software projects; being social processes, software requires a strong network of communication between team members, allowing them to coordinate the inherent set of interdependencies that every project has.

After an interesting analysis of common time overrun situations, Brooks ends this chapter with the following conclusion, which contains the enunciation of the law itself:

> Oversimplifying otrageously, we state Brooks's Law: Adding manpower to a late software project makes it later. This is then the demythologizing of the man-month. The number of months of a project depend upon its sequential constraints. The maximum number of men depends upon the number of independent subtasks. From these two quantities one can derive schedules using fewer men and more months. (The only risk is product obsolescence.) One cannot, however, get workable schedules using more men and fewer months. More software projects have gone awry for lack of calendar time than for all other causes combined.

(Brooks, pages 25 & 26)

This "law" is known and cited throughout the industry as an example of a common pattern, observed once and again in different projects all over the world:

> Fact 3: Adding people to a later project makes it later(…) Intuition tells us that, if a project is behind schedule, staffing should be increased to play schedule catch-up. Intuition, this fact tells us, is wrong. The problem is, as people are added to a project, time must be spent on bringing them up to speed.(…) Furthermore, the more people there are on a project, the more the complexity of its communication rises.

(Glass, page 16)

As a personal experience, I must say that the lecture of this book opened my eyes more than many, many other books. It is a funny read, but also an enlightening one: many anecdotes told by Brooks strangely correspond to my own experience, and this one is no exception. I have seen projects gone unfortunately late because of the simple fact of adding more people; and in one particular case, the project was cancelled altogether. These projects had several factors in common, though:

- Bad documentation, or lack thereof; the only way for newcomers to the project to know what was going on was interrupting the other developers, disrupting the current operations on the project; I think that a good set of documents, describing both the high-level architecture and the low-level APIs are needed for new developers to jump in and catch up. It's maybe not enough, but a good leap forward anyway.
- Lack of architectural vision; projects that do not have an architect, providing vision and technical leadership to the team, are in my opinion exposed to problems when more developers join the project. The architect can act as a proxy person, guiding new developers while they familiarize themselves with the project, isolating other developers from this task.
- Bad project decomposition in components; if the system to be developed is sufficiently large, and the decomposition in components is not prop-

erly done, the overlap and extended communication paths among team members might affect the whole project negatively. A good decomposition breaks down the whole project in a set of smaller ones, with the corresponding set of interfaces, which brings the whole team to work separately on different subsystems. In these, the risk of getting later for adding manpower is reduced proportionally.
- Bad working conditions; I positively think that open spaces are a common disease in our industry. Teams working in open spaces suffer more of noise and visual distractions, and this is more evident when new team members join the project.

Criticism

However famous, Brooks' law has had a good deal of criticism as well, regarding the specific characteristics of projects that might be affected in case that new people is assigned to them. The OS/360 project, which served as the basis for Brooks' work, might not be similar to other projects, and as such, the law would not necessarily apply to them:

> For Brooks' Law to be true, the amount of training effort required from existing staff must be significant. The amount of effort lost to training must exceed the productivity contributed by new staff when they eventually become productive. (…) "Late" chaotic projects are likely to be much later than the project manager thinks–project completion isn't three weeks away, it's six months away. Go ahead and add staff. You'll have time for them to become productive. Your project will still be later than your plan, but that's not a result of Brooks' Law. It's a result of underestimating the project in the first place.(…) Controlled projects are less susceptible to Brooks' Law than chaotic projects. Their better tracking allows them to know when they can safely add staff and when they can't. Their better documentation and better designs make tasks more partitionable and training less labor intensive. They can add staff later in the project with less risk to the project.

(McConnell, 1999)

Scott Berkun gives a more concrete analysis on why the law could be wrong:

> - It depends who the manpower is. The law assumes that all added manpower is equal, which is not true.
> - Some teams can absorb more change than others. Some teams are more resiliant to change.
> - There are worse things than being later. (…) That can be ok if you also get higher quality
> - There are different ways to add manpower. (…) The more experience everyone has with mid-stream personnel changes, the better.

- It depends on why the project was late to begin with. (…) no amount of programming staff modifications will resolve the psychiatric needs of team leaders or the dysfunctions of executives.
- Adding people can be combined with other management action. (…) if you're removing your worst, and most disruptive, programmer and adding one of your best, it can be a reasonable choice.

(Berkun, 2006)

And what about open source projects? Many of these (Linux, Apache, MySQL) are potentially among the biggest software projects ever undertaken, and they don't appear to suffer o fthe problems pictured by Brooks' law:

> But proponents of open source and free software development, including Linux developers, are not completely satisfied with the Law. Most famously (among geeks at any rate), Eric Raymond in his "The Cathedral and the Bazaar," declared Brooks' Law obsolete, if not simply limited, saying "if Brooks' Law were the whole picture, Linux would be impossible." Although Raymond now says that he has somewhat modified his views or was misunderstood, some still would say he is given to oversimplifying and outrageousness himself. "I don't consider Brooks' Law 'obsolete' any more than Newtonian physics is obsolete; it's just incomplete. Just as you get non-Newtonian effects at high energies and velocities, you get non-Brooksian effects when transaction costs go low enough. Under sufficiently extreme conditions, these secondary effects dominate the system – you get nuclear explosions, or Linux."

(Jones, 2000)

Conclusion

So far, the discussion seems to be open. There might be a scale factor for projects, which in turn might expose them to be affected by Brooks' law. I think that research is needed to arrive to a conclusion, even if it will be a statistical one.

Other important facts highlighted in the book are the "second system phenomenon", the productivity advantage of using high-level languages, and the importance of building a prototype - "one to throw away". I can only recommend this book to everyone interested in the field of software engineering (which I did in my own review of classic books in this blog: http://kosmaczewski.net/2005/11/20/my-bookshelf-part-iii/)

References

Berkun, S.; “Exceptions to Brooks’ Law”, January 11th, 2006, [Internet] http://www.scottberkun.com/blog/2006/exceptions-to-brooks-law/ (Accessed June 8th, 2007)

Brooks Jr., F. P.; “The Mythical Man-Month - Essays on Software Engineering, Anniversary Edition”, 1995, Addison Wesley, ISBN 0-201-83595-9

Glass, R. L.; “Facts and Fallacies of Software Engineering”, Addison-Wesley, 2003, ISBN 0321117425

Jones, P.; “Brooks’ Law and open source: The more the merrier?”, IBM, May 1st, 2000, [Internet] http://www.ibm.com/developerworks/linux/library/os-merrier.html (Accessed June 8th, 2007)

McConnell, S.; “Brooks’ Law Repealed?”, IEEE Software, November/December 1999 [Internet] http://stevemcconnell.com/ieeesoftware/eic08.htm (Accessed June 8th, 2007)