

# The Importance of Ignorance in Requirements Engineering: An Earlier Sighting and a Revisitation

Daniel M. Berry  
Computer Systems Group  
Computer Science Department  
University of Waterloo  
200 University Ave. West  
Waterloo, Ontario N2L 3G1  
Canada

Corresponding author. Tel: none; fax: +1-519-746-5422.

[dberry@uwaterloo.ca](mailto:dberry@uwaterloo.ca)

July, 2001

© Copyright 2001 by Daniel M. Berry

## Abstract

I describe an earlier publication of an idea that I published on the importance of ignorance in requirements engineering. Contained in that earlier publication are new ideas about Brooks's Law.

## Earlier Sighting of an Important Idea

In 1995, I published in these pages an article titled "The Importance of Ignorance in Requirements Engineering" (Berry, 1995). In it, I described experiences that let me to believe that it was essential to have on each requirements engineering (RE) team a smart ignoramus, a computer-system-savvy person who knew nothing about the domain of the computer-based system (CBS) being specified by the RE effort. This smart ignoramus has the role of asking the so-called ignorant, not stupid, questions that expose tacit assumptions that each domain-expert stakeholder assumes incorrectly that all other domain-expert stakeholders understand. If these conflicting tacit assumptions are not made explicit, the conflicts are not discovered until much later in the CBS's lifecycle, at a time in which it is one or two orders of magnitude more expensive to fix the errors in the CBS's code and documents that are borne of the conflicts (Boehm, 1981).

While I made this observation based entirely on my own experience, it has recently become apparent that I was not the first to make this observation. On the Software RE mailing list ([sre@ics.mq.edu.au](mailto:sre@ics.mq.edu.au)) maintained by Didar Zowghi, on Wednesday 9 May 2001 at 23:07:37 EDT, Haim Kilov pointed out that P. Burkinshaw, an attendee of the Second NATO Conference on Software Engineering in Rome in 1969 (Buxton and Randell, 1969), said:

Get some intelligent ignoramus to read through your documentation and try the system; he will find many "holes" where essential information has been omitted. Unfortunately intelligent people don't stay ignorant too long, so ignorance becomes a rather precious resource.

Interestingly, this quotation was Burkinshaw's *only* recorded entry in the proceedings.

I regret that I was not aware of this quotation at the time I wrote and published my paper. Had I known about Burkinshaw's quotation, I would certainly have cited it in my paper. Also, I would have written my paper differently, as providing more data points in support of a principle described earlier.

## Ignorance and Brooks's Law

There is more to Burkinshaw's quotation, one more sentence, in fact.

Suitable late entrants to the project are sometimes useful here.

This additional sentence immediately reminded me of Brooks's Law (Brooks, 1975; Brooks, 1995).

Adding manpower to a late software project makes it later.

Fred Brooks explains how the lines of communication within a team grows quadratically with team size, while the number of hours available for work grows linearly with team size. Consequently, for any project, at some number of team members, the number of hours needed for the additional communication required by an additional team member exceeds the number of hours that the additional team member adds to the hours available for work on the project. Brooks's law suggests that most large CBS development projects are already staffed beyond this number.

Tom DeMarco (1996) explains how, for a significant initial period, a new member on a CBS development team requires time from potentially all existing team members to learn the ropes, to be brought up to speed. Thus, not only does the new person not add time to the hours available for work, he or she takes hours away from the other team members. Until the new person gets up to speed, he or she is, at best, a non-contributor and, normally, a drag on the team.

Burkinshaw's quotation reminded me also of Steve McConnell's attempt to repeal Brooks's law. McConnell (1999) explains that in practice it is very hard to determine if a late CBS development project to which people were added mid stream was late *because* of the personnel addition. The typical software project starts with a ridiculous underestimate of the time required, and would be late according to this estimate anyway, even if no new personnel were added during development. Indeed, it is the pressure created by the underestimate that prompts the addition of new personnel. McConnell's second point is that if a project's plan calls for gradually increasing staff as it moves from analysis and specification through design into implementation, then the required learning and communication have been planned, and adding the people as planned probably does not delay completion of the project. McConnell summarizes:

Obviously, it's beneficial to add staff [early in the project, ] until some point in the project's schedule, after which adding staff becomes detrimental. Implicit in Brooks' law is that it applies only to the final phases of a project.

Thus, adding personnel slows a CBS development project down if the addition is done in desperation in the frantic period soon before an impossible deadline.

Burkinshaw's observation suggests that there might be another reason that adding another person may prove beneficial. He or she might just ask the questions that expose errors whose cost to fix later would have been enough more than the cost to fix now that the savings offset the cost to bring him or her up to speed.

Indeed, in the project to build an ethernet switching hub that I cited (Berry 1995) as an example in which my ignorance of the domain paid off, I had been brought in to resolve the requirements conflicts a ridiculous six weeks before the running product was supposed to be exhibited at a trade show. After I helped work out the requirements, the implementors furiously worked to get the code of the project to match the now established requirements. Four weeks later, two weeks before the deadline, I, still very ignorant of the domain, ran an inspection of the implementing code. The programmers were dragged into the inspection kicking and screaming that their valuable time was better spent finishing the code. I had read the code to be able to do my job as the ignorant, fair facilitator. I found all sorts of weird things, including a semantically mismatched parameter. The data types of the actual and formal agreed, but the processing with the parameter being done on opposite sides of the call just did not jibe. As I was conducting the inspection, I stepped a bit out of my facilitator role and asked about that. I saw two programmers blanch and do double takes. They turned to each other and started talking some jargon that went way over my head. Two minutes later, they announced that I had caught a *major* interface flaw that would have brought the demo at the

trade show to an embarrassing screeching halt. The fix was actually quite simple. My ignorant question saved the company far more than my fee.

### **Other Work on Brooks's Law**

Several other people have looked more closely at Brooks's Law and have attempted to identify more precisely when it applies and when it does not. For example, Gordon and Lamb (1977) found that adding more people than necessary as early as possible does help recover from schedule slippage. Jerry Weinberg (1982) examined the causes of Brooks's Law and found the delay caused mainly by the additional coordination overhead and the additional training overhead required. Other work has focused on simulations based on models of system dynamics (Abdel-Hamid and Madnick, 1991; Hsia, Hsu, and Kung, 1999; Madachy and Tabet, 2000). This work has used system dynamics models that drive simulations in order to be able to pin down circumstances in which the law holds and it does not. To my knowledge, no one has done controlled experiments on real-live large projects to validate the law or the models. However, it would be very hard to set up experiments without major threats to validity and with sufficient data for significant conclusions.

### **Conclusion**

Adding domain-ignorant people to a CBS development project clearly affects the time required for completing the project. However, there are conflicting indications as to the direction of this effect.

### **Acknowledgements**

I thank each of Martin Glinz, Ric Holt, Michael Jackson, Haim Kilov, and Bashar Nuseibeh for showing me all or part of Burkinshaw's quotation. I thank Barry Boehm, Ray Madachy, and Juan Ramil for pointing me to some recent work on Brooks's law. I was supported in part by NSERC grant NSERC-RGPIN227055-00.

### **References**

- Abdel-Hamid, T.K. and Madnick, S.E., 1991. *Software Project Dynamics: an Integrated Approach*. Prentice-Hall, Englewood Cliffs, NJ.
- Berry, D.M., 1995. The Importance of Ignorance in Requirements Engineering. *Journal of Systems and Software* 28 (2), 179–184.
- Boehm, B.W., 1981. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ.
- Brooks, F.P. Jr., 1975. *The Mythical Man-Month: Essays on Software Engineering*. Addison Wesley, Reading, MA.
- Brooks, F.P. Jr., 1995. *The Mythical Man-Month: Essays on Software Engineering, Second Edition*. Addison Wesley, Reading, MA.
- Buxton, J.N. and Randell, B., 1969. *Software Engineering Techniques: Report on a Conference*. NATO Scientific Affairs Division, Rome, IT
- DeMarco, T., 14 April 1996. Human Capital, Unmasked. *New York Times*, New York, NY
- Gordon, R.L. and Lamb, J.C., June 1977 A Close Look at Brooks' Law. *Datamation*, 23(6), 81–83, 86.
- Hsia, P., Hsu, C.-T., and Kung, D.C., 1999. Brooks' Law Revisited: A System Dynamics Approach. In: *Proceedings of the Twenty-Third Annual Computer Software and Applications Conference (COMPSAC'99)*. IEEE Computer Society, Los Alamitos, CA, 370–375
- Madachy, R. and Tabet, D., 2000. Initial Experiences in Software Process Modeling. *ASQ Software Quality Professional* 2 (3), 15–27.
- McConnell, S., 1999. Brooks' Law Repealed. *IEEE Software* 16 (6), 6–8.
- Weinberg, G.M., 1992. *Quality Software Management: Volume 1, System Thinking*. Dorset House Publishing, New York, NY.