David Schirduan 9-14-11
CSCI362 Bowring

Response to Mythical Man Month

I enjoyed reading this article. I actually read a little bit further on, simply because it was interesting, well written, and seemed to address many of the problems and benefits of different software design strategies. It was written in 1995, or was reprinted then. A lot of the issues mentioned are still very relevant today, and some of his suggestions I had never heard of before. They aren't perfect, but at least they are steps in the right directions, and are promoting a change that is viable.

The "Tar Pits" explanation of the current difficulties in software hit most of the highlights. He mostly focuses on the difficulties involved with large software system. Many of the difficulties deal with communication, labor, etc. I wish that he had talked more about the difficulties in modeling what is basically an abstract idea into something that can be communicated among different groups. I see this as one of the main hurdles to overcome, and a large part of what makes Computer Science such a unique field to go into; as well as such a difficult one to work in teams with.

Another issue that I hadn't really thought about was scheduling. He did a great job explaining the difficulties in creating and maintaining deadlines, while still reporting back on the progress of the product. It is easy to forget that programming is a mystery to everyone else, and business partners, stockholders, and other stakeholders need some way to understand the real progress of the work. They especially need to understand that the lack of an easily visible prototype doesn't mean that there isn't development being done. In computer science, the planning phase is the most important, and the least evident.

I also liked the distinction that he made between man-power and man-hours. In some businesses it's perfectly acceptable to simply increase the workforce in order to hurry a product or finish a project. In Computer Science though, since people are basically working on a universal idea, then too many cooks spoil the broth. Adding new members to the team slows down the entire team. The experienced employees have to take time from their work to explain and train the new employees. This slows everyone down, since it might take a considerable amount of time to bring a new person up to speed on a difficult and complex project; like "adding gasoline to a flame" in order to solve the problem.

Finally, his proposed solution was interesting as well. The "surgical team" was unique, I had never heard of anything like it before. I could see how it would be beneficial to put people into smaller teams, with training and a sort of apprenticeship going on among the less experienced; with the more experienced taking the lead, and doing most of the coding. I don't know how well that would work on larger projects, since smaller teams means more teams, and more teams means more communication problems. Overall, this excerpt was very eye-opening. I enjoyed the way he writes, and the points that he makes are impeccable. I just hope that soon a "silver bullet" can be developed in order to simplifiy a lot of these difficulties.