

Overcoming Brooks' Law

Kealy Opelt, Software Developer at Menlo Innovations LLC
kopelt@menloinnovations.com

Abstract. Most programmers are familiar with the notion that adding new resources to a project will most likely slow down progress if not bring it to a complete stop while new team members are brought up to speed. Unfortunately, in business situations there are often important opportunities missed or dollars lost if a software development team cannot increase the rate at which they complete functionality. Surprisingly, many of the practices of Extreme Programming also help teams overcome Brook's Law and actually increase production by adding resources. If more software development teams successfully demonstrated this phenomenon, then many business sponsors would be interested in supporting the practices used by these teams.

Introduction

Brooks summarized his law, "The bearing of a child takes nine months, no matter how many women are assigned." It is possible that he never imagined software development in an open environment such as Extreme Programming (XP). Menlo Innovations, a custom software design and development shop, finds that the XP practices create on the job training and constant communication within the team, where new developers learn faster and contribute to the team productively within weeks rather than months. Teams can reap these benefits through an XP environment, overcoming Brooks' Law and directly benefiting each software project's production and its members.

Brooks' Law

Brooks' Law states "Adding manpower to a late software project makes it later" and that increasing the output of an expanded team is crippled by the "added burden of communication" where "communication is made up of two parts, training and intercommunication"[1]. Adding members to a team would increase the burden of communication for team members in some of the following ways:

- explaining code to another developer
- training on the specific technical tools of the project
- learning the architecture of the code
- learning how to detect failures when code is changed
- learning the business domain
- reading the design specifications
- being interrupted to answer questions from other team members

Clearly all of these things can have a negative impact on productivity, and if that negative impact is large enough, a team that is increased significantly would produce less output.

The Doubling Experience

While all of the above challenges are real, of Menlo's experience is different. An XP team with eight developers doubled to a team of sixteen developers in the period of less than three weeks. The choice to increase the team was based on the customer's desire to complete more functionality with an already established deadline. As each new developer joined the team they spent the first few weeks pairing with developers who were familiar with Menlo's process, the architecture, tools and project domain. As new team members paired in to the team their pair discovered they were less efficient, completing fewer stories. They spent more time explaining existing code, architecture, and guiding progress on new features. However experienced team members found that they did not stop producing output to bring new team members up to speed. Instead they learned what they needed by the way they worked. Reflecting about why this was possible the team concluded the XP practices helped them to overcome the challenges.

On the following page table 1 shows the challenges new programmers faced when joining the team and the XP practice the team discovered helped decrease the high cost of communication.

Table 1: Developer Challenges with Decreasing Practice

Challenges For New Developers	Practices That Reduce The Negative Impact
How do I know who owns the code?	Collective Code Ownership
How do I decide if my change broke something else?	Test Driven Development
How can I estimate my story cards if I don't know the code yet?	Estimation Every Iteration, Team Co-location
What should the architecture for new features look like?	Simple Design, System Metaphor, Design Improvement as Needed
How do I quickly communicate with my team members?	Team Co-location, Pair Programming
How do I gain the respect of my peers?	Pair Programming
Who can I turn to for help?	Team Co-location, Pair Programming
How do I add new features in code I don't know yet?	Simple Design, System Metaphor, Design Improvement as Needed, Team Co-location, Pair Programming
How do I merge my code with everyone else's?	Continuous Integration
What am I allowed to work on?	Planning Game, Collective Code Ownership
What story should I work on first?	Planning Game
How do I fix a bug with out being punished for failing?	Collective Code Ownership
How do I get my work done with out "burning out" from working too many hours?	Sustainable Pace

Even while the team was experiencing the practices as solutions, more than one programmer noticed their slow down and raised the question to the project managers; did it really make sense to add more resources, when it was

obviously less efficient? Some team members and some of the sponsoring managers gathered around a white board to explore the topic, eventually creating the following figure.

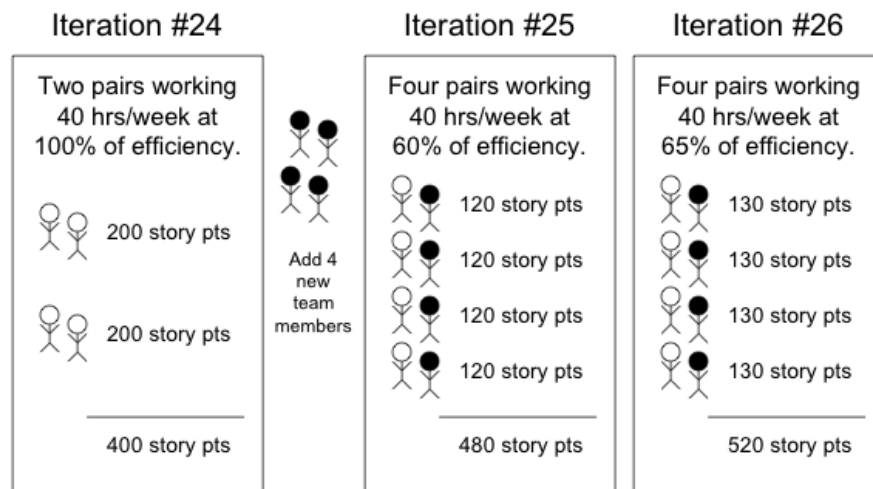


Fig. 1. Depiction of mature XP team the iteration before doubling the team and the effects of story points completed two iterations after.

Figure 1 of a team doubling separates the issue of increased output from decreased productivity. The team at two pairs is working at 100% productivity accomplishing 400 story points. When they are doubled to four pairs each pair's productivity decreases to 60% then only accomplishing 120 story points. However, overall story points increase to a total of 480 story points. This makes the choice to increase the team a business decision of whether cost or deadline is more important and not about what how many story points a pair can complete. Apparently many programmers focus on the cost and low personal efficiency, making them refer to Brooks' Law, when increasing the overall team productivity should be a choice made by the business.

Conclusion

Why should a business person care about the software development methodology you use? Perhaps they would if you could provide software with fewer bugs, software at a lower cost, or provide completed software sooner. When a business sponsor asks if software can be delivered sooner there are two obvious potential solutions; reduce the scope of the project or increase the size of the team. But when software developers are asked to increase the size of the team, most refer to Brooks' Law and declare that adding resources will simply slow the project down. Spending over a year on an XP team that rapidly changed size several times based on business deadlines creates the thought; can all mature XP teams easily overcome Brooks' Law.

References

1. See Brooks, Frederick P., Jr., *The Mythical Man-Month Essays on Software Engineering Anniversary Edition* (Addison Wesley Longman, Inc.: Boston, 1995), 18, 25.