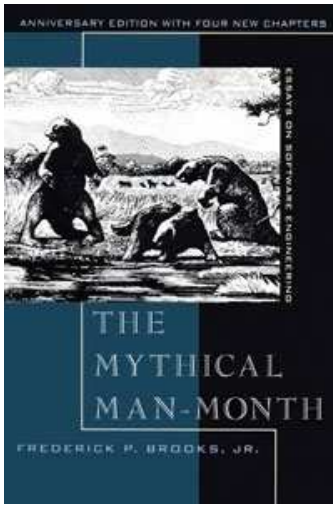
The Mythical Man-Month

From Wikipedia, the free encyclopedia

The Mythical Man-Month: Essays on Software Engineering is a book on software engineering and project management by Fred Brooks, whose central theme is that "Adding manpower to a late software project makes it later." This idea is known as Brooks' law, and is presented along with the second-system effect and advocacy of prototyping. The work was first published in 1975 (0-201-00650-2), reprinted with corrections in 1982, and republished as an *anniversary edition* with four extra chapters in 1995 (ISBN 0-201-83595-9) with the essay "No Silver Bullet" and commentary by the author.

Brooks's observations are based on his experiences at IBM while managing the development of OS/360. He had mistakenly added more workers to a project falling behind schedule. He also made the mistake of asserting that one project — writing an Algol compiler — would require six months, regardless of the number of workers involved. (It required longer.) The tendency for managers to repeat such errors in project development led Brooks to quip that his book is called "The Bible of Software Engineering" because "everybody reads it but nobody does anything about it!"

The Mythical Man-Month



Author	Fred Brooks
Subject(s)	Software project management
Publisher	Addison-Wesley
Publication date	1975, 1995
ISBN	0-201-00650-2 (1975 ed.), 0-201-83595-9 (1995 ed.)
Followed by	*No Silver Bullet*

Contents

Ideas presented

The Mythical Man-Month

Assigning more programmers to a project running behind schedule will make it even later, due to the time required for the new programmers to learn about the project, as well as the increased communication overhead. When N people have to communicate among themselves (without a hierarchy), as N increases, their output M decreases and can even become negative (i.e. the total work remaining at the end of a day is greater than the total work that had been remaining at the beginning of that day, such as when many

bugs are created).

- Group Intercommunication Formula: $n(n - 1) / 2$
- Example: 50 developers -> $50(50 - 1) / 2 = 1225$ channels of communication

The Second-system Effect

The second system an engineer designs is the most dangerous system he will ever design since he will tend to incorporate all of the additions he originated but did not add (due to the inherent time constraints) to the first system. Thus, when embarking upon a second system, an engineer should be mindful that he is susceptible to over-engineering it.

Progress Tracking

Question: How does a large software project get to be one year late? Answer: One day at a time! Incremental slippages on many fronts eventually accumulate to produce a large overall delay. Continued attention to meeting small individual milestones is required at each level of management.

Conceptual Integrity

To make a user-friendly system, the system must have conceptual integrity, which can only be achieved by separating architecture from implementation. A single chief architect (or a small number of architects), acting on the user's behalf, decides what goes in the system and what stays out. A "super cool" idea by someone may not be included if it does not fit seamlessly with the overall system design. In fact, to ensure a user-friendly system, a system may deliberately provide ***fewer*** features than it is capable of. The point is that if a system is too complicated to use, then many of its features will go unused because no one has the time to learn how to use them.

The Manual

What the chief architect produces are written specifications for the system in the form of the manual. It should describe the external specifications of the system in detail, i.e., everything that the user sees. The manual should be altered as feedback comes in from the implementation teams and the users.

The Pilot System

When designing a new kind of system, a team ***will*** design a throw-away system (whether it intends to or not). This system acts as a ***pilot plant*** that reveals techniques that will subsequently cause a complete redesign of the system. This second ***smarter*** system should be the one delivered to the customer, since delivery of the pilot system would cause nothing but agony to the customer, and possibly ruin the system's reputation and maybe even the company's.

Formal Documents

Every project manager should create a small core set of ***formal documents*** which acts as the roadmap as to what the project objectives are, how they are to be achieved, who is going to achieve them, when they are going to be achieved, and how much they are going to cost. These documents may also reveal inconsistencies that are otherwise hard to see.

Project Estimation

When estimating project times, it should be remembered that programming products (which can be sold to paying customers) and programming systems are both three times as hard to write as in-house programs[1]. It should be kept in mind how much of the work-week will actually be spent on technical issues, as opposed to administrative or other non-technical tasks, such as meetings.

Communication

To avoid disaster, all the teams working on a project should remain in contact with each other in as many ways as possible (e-mail, phone, meetings, memos etc.) Instead of assuming something, the implementer should instead ask the architects to clarify their intent

on a feature he is implementing, before proceeding with an assumption that might very well be completely incorrect.

The Surgical Team

Much as a surgical team during surgery is led by one surgeon performing the most critical work himself while directing his team to assist with or overtake less critical parts, it seems reasonable to have a "good" programmer develop critical system components while the rest of a team provides what is needed at the right time. Additionally, Brooks muses that "good" programmers are generally 5-10 times as productive as mediocre ones. See also Organization and Team Patterns (http://www.dfpug.de/loseblattsammlung/online/workshop/design_patterns/sonstiges.htm) .

Code Freeze and System Versioning

Software is invisible. Therefore, many things only become apparent once a certain amount of work has been done on a new system, allowing a user to experience it. This experience will yield insights, which will change a user's needs or the perception of the user's needs. The system should, therefore, be changed to fulfill the changed requirements of the user. This can only occur up to a certain point, otherwise the system may never be completed. At a certain date, no more changes would be allowed to the system and the code would be frozen. All requests for changes should be delayed until the ***next*** version of the system.

Specialized Tools

Instead of every programmer having his own special set of tools, each team should have a designated tool-maker who may create tools that are highly customized for the job that team is doing, e.g., a code generator tool that spits out code based on a specification. In addition, system-wide tools should be built by a common tools team, overseen by the project manager.

Lowering Software Development Costs

There are two techniques for lowering software development costs that Brooks writes about:

- Implementers may be hired only after the architecture of the system has been completed (a step that may take several months, during which time prematurely-hired implementers may have nothing to do).
- Another technique Brooks mentions is not to develop software at all, but to simply buy it "off the shelf" when possible.

References

1. **^** Mythical Man Month (http://www.amazon.com/gp/reader/0201835959/ref=sib_dp_pt#) Figure 1.1, Page 13

External links

- A review by Tal Cohen (http://tal.forum2.org/mythman)
- Frederick P. Brooks, Jr. Homepage (http://www.cs.unc.edu/~brooks/)
- Preface to 20th Anniversary Edition, as found on Safari.Informit.com (http://safari.informit.com/0201835959/pref03#X2ludGVybmFsX1RvYz94bWxpZD0wMjAxODM1OTU5L3ByZWYwMg==)
- Organization and Team Patterns (http://www.dfpug.de/loseblattsammlung/online/workshop/design_patterns/sonstiges.htm)