

CS 428
THE MYTHICAL
MAN-MONTH
Chapters
1, 2, 4, 5

Winter 2018

Bruce F. Webster

Why *The Mythical Man-Month*?

- Originally published in 1975; updated in 1995
- Based on Fred Brook's experience overseeing the development of OS/360 for the IBM/360
- Remains a classic because it set forth most of the fundamental issues and causes of delays and failures in software projects
- Software failures still cost somewhere on the order of \$100 billion/year worldwide, and most of the root causes can be found in Brooks
- Me, before Congress, in 1998:

"Humanity has been developing information technology for half a century. That experience has taught us this unpleasant truth: virtually every information technology project above a certain size or complexity is significantly late and over budget or fails altogether; those that don't fail are often riddled with defects and difficult to enhance. Fred Brooks explored many of the root causes over twenty years ago in *The Mythical Man-Month*, a classic book that could be regarded as the Bible of information technology because it is universally known, often quoted, occasionally read, and rarely heeded."

Chapter 1: the tar pit

- ▶ Concept: levels of complexity in types of software
 - ▶ Individual program for personal use
 - ▶ Commercial product for distribution and sale (word processor, game, app)
 - ▶ “Programming system” (custom operating system, large-scale integrated system) for in-house use
 - ▶ Commercial “programming system” (OS, ERP, etc.) for distribution and sale
- ▶ What are some other types of added software complexity?
- ▶ What can make software difficult to maintain and update?

The Tar Pit (cont.)

- ▶ The Joys of the Craft of Programming
 - ▶ The sheer joy of making things
 - ▶ The pleasure of making things that are useful to other people
 - ▶ The fascination of building complex systems
 - ▶ The joy [heh] of always learning
 - ▶ The delight of working in such a tractable medium “only slightly removed from pure thought-stuff...yet...is real in the sense that it move and works, producing visible outputs separate from the construct itself”
- ▶ Why else do people enjoy software engineering (assuming they do)?

The Tar pit (cont.)

- ▶ The Woes of the Craft
 - ▶ You must perform perfectly
 - ▶ Other people set your objectives, provide your resources, and furnish your information
 - ▶ Usually your authority is not sufficient for your responsibility
 - ▶ You often depend upon other people's programs, which are less than perfect
 - ▶ The upper bound of quality of a complex system is determined by the lowest quality of any of its essential components
 - ▶ Designing grand concepts is fun; finding nitty little bugs is just work
 - ▶ Debugging has *at best* linear convergence
 - ▶ The product is often obsolete before it is completed
- ▶ What are other painful things you've discovered about software engineering?

Chapter 2: The Mythical Man-Month

- ▶ Root causes of software project delays and failure
 - ▶ Our techniques of estimation are [still] poorly developed
 - ▶ Our estimation techniques confuse effort with progress (people & months are interchangeable)
 - ▶ Because we are uncertain of our estimates, we often lack the courage to say we don't know when we'll be done
 - ▶ Schedule progress is poorly monitored and hard to measure
 - ▶ When the schedule slips, the impulse is to add staff, which is "like dousing a fire with gasoline"
- ▶ What have you observed?

The Mythical Man-Month (cont.)

- ▶ “All programmers are optimists”
 - ▶ Only optimists build complex systems. (Adele Goldberg)
 - ▶ We too often assume each task will take only as long as it “ought” to take
 - ▶ The probability that a given task will go well may be relatively high, but a meaningful software project comprises hundreds if not thousands of such tasks
 - ▶ Thus: It is very easy to lose a day; it is impossible to make it up.
 - ▶ Additional complication: we tend to focus on the easy tasks first and defer the difficult problems until late in the project – illusion of great progress
- ▶ What are some other ways in which we tend to be overly optimistic?

8 The Mythical Man-Month (cont.)

- ▶ The Man-Month
 - ▶ The “man-month” as a unit for measuring the size of a software engineering project is a dangerous and deceptive myth
 - ▶ Sequential constraints in development as well as communication requirements make the “man-month” concept unrealistic (and self-deluding)
 - ▶ Adding a person to a project not only increases the communication paths and requirements, it also costs time for bringing the new person up to speed
 - ▶ Thus, adding more people lengthens, not shortens, the schedule (Brooks Law)
- ▶ In light of the above, what do you think the impact of personnel turnover is?

The Mythical Man-Month (cont.)

- ▶ Component debugging and system testing forces sequential constraints
 - ▶ Testing is usually the most mis-scheduled (underestimated) part of programming
 - ▶ Brooks' rule of thumb: 1/3rd planning, 1/6th coding, 1/4th component test, 1/4th system test
 - ▶ "I found that few allowed one-half the project schedule for testing, but that most did indeed spend half of the actual schedule for that purpose."
 - ▶ The 90/90 rule: 90% of the work takes the first 90% of the schedule, and the remaining 10% of the work takes the other 90% of the schedule
 - ▶ Underestimation of system testing (integration, end-to-end, performance, stress) is particularly damaging since it shows up right when project completion is expected

The Mythical Man-month (cont.)

- ▶ Gutless estimating
 - ▶ Endemic in our industry
 - ▶ Completion date is picked because “we have to have it by then” or to meet a “market opportunity”, not based on any rational basis or realistic estimate
 - ▶ Upper management often does not want to hear a realistic estimate
- ▶ Regenerative schedule disaster
 - ▶ So, what happens when the project is late? “Add people to it. Work longer hours.” Both are counter-productive.
 - ▶ Only real solution: slip deadline and/or drop features.
- ▶ Observations?

Aristocracy, democracy, and system design

- ▶ Brooks: conceptual integrity is *the* most important consideration in system design (I agree)
- ▶ Simplicity, straightforwardness, unity of design are necessary
- ▶ The design must proceed from one mind or a very small number of agreeing resonant minds
- ▶ The conceptual integrity of a system determines its ease of use
- ▶ A consistent architecture enhances the creative style of implementers
- ▶ A well-thought-out architecture increases the robustness and adaptability of the resulting software system

The Second-System Effect

- ▶ Interactive discipline for the architect
 - ▶ The architecture is valuable input into estimating the implementation and testing
 - ▶ If the schedule is unacceptably long, the architect can look for ways to simplify
 - ▶ Big challenge: features that may seem simple to the customer may actually be very difficult to design and implement
- ▶ The second-system effect
 - ▶ Brooks notes later that true iterative development can diminish this problem, but...
 - ▶ The first shipping version usually has many deferred features; there is a strong temptation to throw in “everything but the kitchen sink” into version 1.1 or 2.0
- ▶ Real-world issue: incurring ‘technical debt’ and not handling it

Assignments for next class (1/28/19)

- ▶ NOTE: No class next week due to holiday
- ▶ By midnight on Saturday, 1/26
 - ▶ Finalize team membership and refine project definition and scope
 - ▶ Watch one podcast of your choosing (class website, under 'Readings and Podcasts')
 - ▶ Complete your team's project Wiki with a final list of team members
- ▶ Read *The Mythical Man-Month* chapters 7, 11, 14
- ▶ Read Webster #1 readings (online at class website under 'Readings and Podcasts')