SE 181
Software Engineering

The Mythical Man-Month

Ch 1: The Tar Pit

- Why have not all industrial programming teams been replaced by dedicated garage duos?

- Evolution of the programming systems product.
  - Program
    - Can be run by the author of the system
  - Programming product
    - Can be run in many operating environments
    - Can be repaired and extended by anybody
  - Programming system
    - A collection of interacting programs that act as a "system" for large tasks
    - Every input and output conforms in syntax and semantics with precisely defined interfaces.

Ch. 2: The Mythical Man-Month

- More software projects have gone awry for lack of calendar time than for all other causes combined.

- The man-month as a unit for measuring the size of a job is a dangerous and deceptive myth.
- The man-month assumes that we can partition a task among many workers with no communication among them.

- All programmers are optimists
- Assumption: "All will go well, i.e., each task will take only as long as it "ought" to take.

- Rule of thumb for scheduling:
  - 1/3 planning
  - 1/6 coding
  - ¼ component test and early system test
  - ¼ system test, all components in hand

- Brook's law: Adding manpower to a late software project makes it later

Ch. 3: The Surgical Team

- Mill's proposal:
  - The surgeon – Chief Programmer
  - The copilot
  - The administrator
  - The editor
  - Two secretaries
  - The program clerk
  - The toolsmith
  - The tester
  - The language lawyer

Ch. 4: Aristocracy, Democracy, and System Design

- Conceptual integrity is the most important consideration in system design.

- Most programming systems reflect conceptual disunity that arises from the separation of design into many tasks done by many men.

- Conceptual integrity dictates that the design must proceed from one mind, or from a very small number of agreeing resonant minds.

Ch. 6 : Passing the Word

- Future specifications will consist of both formal definition and prose definition.
- Every development organization needs an independent technical auditing group (testing) to keep it honest.

Ch. 7: Why did the tower of Babel fail?

- Lack of communication resulted in a host of other issues (e.g., disputes, bad feelings, etc.)

- The essentials which any organizational tree must have in order to be effective:
  - A mission
  - A producer
  - A technical director or architect
  - A schedule
  - A division of labor
  - Interface definitions among the parts.

Ch. 8: Calling the Shot

- How does one estimate?

- Extrapolation of times for the hundred-yard dash shows that a man can run a mile under three minutes.
- Estimates make an unrealistic assumption about the number of technical work hours per man-year.

Ch. 10: The Documentary Hypothesis

- Only when one writes do the gaps appear and the inconsistencies protrude.
- The documents will communicate the decisions to others.
- Only the written plan is precise and communicable.

Ch. 11: Plan to Throw One Away

- In most projects, the first system is barely usable: It may be too slow, too big, awkward to use or all three.

- The total cost of maintaining a widely used program is typically 40% or more of the cost of developing it.
- Surprisingly, the cost is strongly affected by the number of users. More users find more bugs.

- The fundamental problem with program maintenance is that fixing a defect has a substantial (20%-50%) chance of introducing another. So the whole process is two steps forward and one step back.
- Why aren't defects fixed more cleanly?
  - First, even a subtle defect shows itself as a local failure of some kind. In fact, it often has system-wide ramifications, usually non-obvious.
  - Second, the repairer is usually not the person who wrote the code, and often he is a junior programmer or trainee.

Ch. 12: Sharp Tools

- Specialized tools are important.
- Proposed one toolmaker per team.

- Target machine vs. vehicle machine.

- Control: The idea of program copies belonging to the managers who alone can authorize their change.
- Formal separation and progression from the playpen, to integration, to release.

Ch. 13: The Whole and the Parts

- How does one build a program to work?
- How does one test a program?
- How does one integrate a tested set of components into a tested and dependable system?

- The crucial task is to get the program defined. Many failures concern exactly those aspects that were never quite specified.

- The unexpectedly hard part of building a programming system is system test.

- Build plenty of scaffolding
  - All programs and data build for debugging purposes
- Control changes
  - Somebody must be in charge and authorize changes or substitution of one version for another.
  - Need to log all changes
- Add one component at a time
- Quantize updates
  - The replacement of a working component by a new version requires the same systematic testing procedure that adding a new component does

Ch. 14: Hatching a Catastrophe

- Day-to-day slippage is harder to recognize, harder to prevent, harder to make up.

- To control a big project on a tight schedule you need to have a schedule.

Ch. 15: The Other Face

- Importance of documentation
- Flowcharts
- Self-documenting programs