Mythical Man-month (Ch 3,4)

Ch3. The Surgical Team

Problem

- Most experience with large programming systems shows that the brute-force approch is costly, slow, inefficient, and produces systems that are not conceptually integrated.
- It fails to approach the ideal of the *small* sharp team, which by common consensus shouldn't exceed 10 people.
- the original 200-man team was not large enough to build the really large systems by brute-force methods.
- small, sharp team의 문제: it is too slow for really big systems.

Mills's Proposal

- Mills proposes that each segment of a large job be tackled by a team, but that the team be organized like a **surgical team** rather than a hog-butchering team.
  - 각 멤버의 역할
    - the surgeon (a chief programmer)
    - the copilot
    - the administrator
    - the editor
    - two secretaries
    - the program clerk
    - the toolsmith
    - the tester
    - the language lawyer

How It Works

- uno animo (마음을 하나로 하여)
- The surgeon and copilot are each cognizant of all of the design and all of the code.
- There are no differences of interest, and differences of judgement are settled by the surgeon unilaterally(일방적으로)
- Yet the specilalization of function of the remainder of the team is the key to its efficiency, for it permits a radically simpler communication pattern among the members.

Scaling Up

- How to build things that today take 5000 man-years?
- The conceptual integrity of each piece has been radically improved.
- However, seperate techniques must be used.

Ch16. No Silver Bullet-Essence and Accident in Software Engineering

- Silver bullet, something to make software costs drop as rapidly as computer hardware costs do.
- But, as we look to the horizon of a decade hence, we see no silver bullet.

Does It Have to Be Hard?-Essential Difficulties

- We cannot expect ever to see two-fold gains every two years.
- The anomaly is not that software progress is so slow but that computer hardware progress is so fast.
- The essence of a software entity is a construct of interlocking concepts: data sets, relationships among data items, algorithms, and invocations of functions.
- The inherent properties of (this) irreducible essence of modern software systems: complexity, conformity, changeability, and invisibility.

Complexity

- Software entities are more complex for their size than perhaps any other human construct.

Conformity

- Much complexity comes from conformation to other interfaces; this cannot be simplified out by any redesign of the software alone.

Changeability

- The software entity is constantly subject to pressure for change.

Invisibility

- Software is invisible and unvisualizable.
- This lack not only impedes the process of design within one mind, it severely hinders communication among minds.

Past Breakthroughs Solved Accidental Difficulties

- High-level languages
- Time-sharing
- Unified programming environments

Hopes for the Silver

- Ada and other high-level language advances
- Object-oriented programming
- Artificial intelligence
- Expert systems
- “Automatic” programming
- Graphical programming
- Program verification
- Environments and tools
- Workstations

Promising Attacks on the Conceptual Essence

We must consider those attacks that address the essence of the software problem, the formulation of these complex conceptual sturcutres.

- Buy versus build

- Requirements refinement and rapid prototyping
- Incremental development-grow, not build, software
- Great designers