**Mythical man month**

**Chapter 1 : The tar pit**

**The programming systems product**

Having a program as programming product, can be run tested repaired and extended by anybody, generalized costs 3 times than just a product

Having a program in a programming system (collection of interacting programs). Every input/output conforms in syntax and semantics with precisely defined interfaces, uses prescribed budget of resources, tested with other system components in all combinations. It is subtle due to component interaction bugs. It costs 3 times more than anormal product

A programming systems product differs from a normal product in all the ways listed above, truly useful and costs 9 times more than a normal program

**The joys of the craft**

Programming is fun because there is the creation of something useful that you can see it and it feels it was built from scratch

**The woes of the craft**

It needs to be perfect to work (against human nature)

One rarely controls the circumstances of his work, or their goal (other people set them)

Debugging has a linear or even quadratic convergence

Better programs can come after and take place (ours becomes obsolete)

**Scale of programmers** from -10 to +10

If you want to go higher in the scale, you should be humble and have good communication skills. If you are intelligent but arrogant you may go quickly to the negative (damaging the project)

**Chapter 2 : The mythical man-month**

**Optimism**

Programmers (young mostly) are too time optimistic.

The inconsistencies and incompleteness of our ideas become clear only during implementation.

In a single task, things might go well without delays. But in chained tasks, the probability of each working becomes vanishingly small

**The man-month**

When a task cannot be partitioned because of sequential constraints, adding more programmers won’t affect the schedule

Tasks that can be partitioned but require communication take less time when there are more people but not inversely proportional due to the effort put into communication

Intercommunication increases effort by n²/2, so if you exceed the number of programmers, things get counterproductive

**Systems test**

Rule of thumb for scheduling a software task: 1/3 planning, 1/6 coding, ¼ component test and early system test, ¼ system test with all components in hands

**Gutless estimating**

Failure to allow enough time for system testing is disastrous, causes delay without warning, unsettling to customers and managers

**Regenerative schedule disaster**

Suppose a task is estimated at 12 man-months, assigned to 3 men (estimate time 4 months). First task takes 2 months instead of 1. Still 9 m-m effort. Solutions

1. Assume task must be done on time, only first part was misestimated. Add 2 men to the 3 assigned
2. Assume task must be done on time, whole estimate was uniformly low. Add 6 men to the 3 assigned
3. Reschedule, allow enough time to ensure that the work can be carefully and thoroughly done, rescheduling won’t have to be done again
4. Trim the task, formally and carefully by hasty design and incomplete testing

First 2 sol°, estimating that the task must be done on time is disastrous. Consider regenerative effects like the time of recruitment and training, + different partitioning. Adding manpower toa late software project makes it later.

This demythologizes the man-month. Nb of months depends upon its sequential constraints. The max nb of men depends upon the nb of independent subtasks