*11*
*Plan to Throw One Away*

## *Plan to Throw One Away*

- Pilot Plants and Scaling Up
- The Only Constancy Is Change Itself
- Plan the System for Change
- Plan the Organization for Change
- Two Steps Forward and One Step Back
- One Step Forward and One Step Back

## *Pilot plants and scaling up*

- Chemical engineers use a pilot plant to bridge the lab bench and the production system
  - I.e., deal with scale before commiting capital
- Programmers have not learned this lesson
  - Schedules do not account for incorrect design
  - First systems are slow, big and awkward
  - (Ever heard of, or seen, Windows 1.0?)
- Plan to throw one away; you will, anyhow

## *The only constancy is change itself*

- Accept change as a way of life
- Cosgrove: programmers deliver user satisfaction, i.e., the new program will affect
  - The user's needs, and
  - The user's perception of their needs
- Software's tractability and invisibility incite users to expect flexibility
- Designers also learn from initial designs

## *Plan the system for change*

- Modularization
- Subroutines
- Interface definitions
- Documentation of the same
- Standard calling sequences
- Table-driven design
- High-level languages; Self-documenting code
- Version control, with schedules and freeze dates

## *Plan the organization for change*

- The designer's reluctance to document reflects a genuine lack of confidence in the design itself
  - Written designs are exposed to criticism
  - A threatening organization structure discourages tentative documentation
- Structuring an organization for change is harder than designing a system for change
- Why?

## *Parallel career paths*

- Maximize flexibility through interchangeable parallel technical and managerial career paths
- Socialogical factors undermine this strategy
  - Seniority devalues technical skill
  - Management jobs carry higher prestige
  - Prestige comes from more than just salary
- Norm's observation: it doesn't work
  - I.e., power lies in budget and head count
  - Technologists control neither

## *Surgical team strategy*

- Reduces social obstacles to the joy of the craft
  - I.e., The surgeon does not demean himself when building programs
- Reduces the organizational interfaces
- Increases organizational flexibility by defining the structural unit in the surgical team
- “It is really the long-run answer to the problem of the flexible organization.”

## *Software maintenance is different*

- Hardware maintenance comprises three activities
  - Preventive, e.g., cleaning, lubricating
  - Supportive, i.e., replacing deteriorated parts
  - Corrective, i.e., adjusting defective designs
- Software maintenance has no physical aspect

## *Software reliability definitions*

- Failure
  - The software's deviation in behaviour from the specification
- Fault
  - The software's deviation from correct design
  - A.k.a “bug”
- Error
  - The human activity that created the fault

## *Five types of software maintenance*

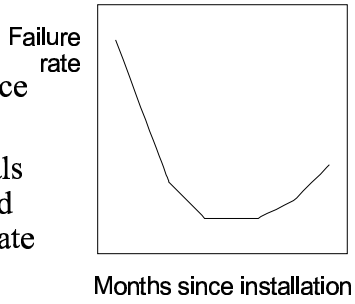
- Corrective: Remove faults
- Perfective: Improve capabilities
  - E.g., functionality, efficiency, maintainability
- Adaptive: Adapt to a different environment
  - E.g., hardware, operating system, middleware
- Emergency: Correct urgent faults
  - Increases risk due to reduced testing
- Preventive: Remove latent faults

## *Maintenance demands*

- User demands increase with usage
- Increased usage increases system longevity
- Increased longevity risks hardware obsolescence
- Hardware obsolescence requires adaptation
- The act of maintaining software introduces faults
- Software maintenance gets harder over time

## *Two steps forward – One step back*

- Betty Campbell of MIT conjectures that
  - Initial usage exposes many faults quickly
  - Corrective maintenance reduces fault density
  - Extended usage reveals new user demands and increases the failure rate




## *Positive feedback*

- Correcting a fault has a finite probability of introducing a new fault (e.g., Brooks 20~50%)
  - The correction itself has non-obvious effects in other parts of the system
  - The maintainer is often a different person from the design
  - The maintainer often has limited experience
- Maintenance requires regression testing

## *One step forward – One step back*

- Lehman and Belady; Large operating systems
  - Module count increases linearly with time
  - Modules affected increases exponentially
- Repairs tend to undermine conceptual integrity
  - Less effort goes to original faults
  - More effort goes to introduced faults
- Systems programming decreases entropy
- Software maintenance increases entropy