Today’s Lecture

• Discuss additional issues from The Mythical Man-Month
  – Plan to Throw One Away
  – The Whole and the Parts
  – Hatching a Catastrophe
  – The Other Face

• Skipping
  – Chapters 9, 10, and 12

Plan to Throw One Away

• Brooks says
  – Plan to throw one (a software system) away; you will, anyhow.

• Why?
  – Consider our example of chemical engineers
    • Scaling a laboratory result up to actual (and practical) use requires a pilot step
    • desalting water 10,000 gallons/day to 2,000,000 gallons/day is impractical
  – Experience shows that you will discard a lot of the first implementation anyway! (Multics paper, 1972)
Why?, continued

• Brooks further argues
  – The management question
    • Plan to build a system to throwaway
      – or
    • Plan to build a throwaway that is delivered to the customer
  – Results
    • former: experience gained; feedback can be applied
    • latter: user is aggravated and demands support

Rapid Prototypes

• Brooks is essentially arguing for rapid prototypes
  • (although he doesn’t follow through)
    – They help gain early feedback
    – They are intended from the start to be thrown away
      • We have already discussed some of the problems associated with prototypes; these problems illustrate the need to educate all stakeholders in the purpose of prototypes
  • Instead he focuses on planning for change in a large software project

One cause of change

• A programmer delivers satisfaction of a user need rather than any tangible product
  – And both the actual need and the user’s perception of that need will change as programs are built, tested, and used.
    • Cosgrove, 1971
• Other factors
  – hardware, assumptions, and environment

Handling change in systems

• modularization and subroutines
• precise and complete interfaces
  – standard calling sequences
  – complete documentation
• table-driven techniques
• high-level languages
• configuration management
Organizational Issues

• Culture must be conducive to documenting decisions; otherwise nothing gets documented
• Brooks other points consider
  – job titles
  – keeping senior people trained
  – using the surgical team to combat the “too valuable” syndrome
• A lot of these, as discussed last time, are specific to IBM (back in the late 60s) and difficult to apply

Brooks on Maintenance

• Two Steps Forward and One Step Back
  – Campbell’s life cycle of bugs (Fig. 11-2)
  – Fixing a bug has a chance of adding another
    • Lots of regression testing needed
• One Step Forward and One Step Back
  – Maintenance is an entropy-increasing process
    • As maintenance proceeds, the system is less structured than before; conceptual integrity degrades

The Whole and the Parts

• How does one build a successful program?
  – Focus on the specifications and test them!
    • Testing should be performed by an external group
  – Top-down Design
    • Design as a set of refinement steps
    • Use of abstraction at each level
    • Modular decomposition

The Whole and the Parts, continued

• Other techniques
  – Structured Programming
  – Component Debugging
  – System Debugging
    • Use debugged components (reuse)
    • Build scaffolding (stubs, test data)
    • Control Changes
    • Add one component at a time, and quantize updates
Hatching a Catastrophe

• How does a project get to be a year late?
  – One day at a time!
• Major Calamities are “easy” to handle
  – The whole team pulls together and solves it
• It’s the day by day slippage that is harder to recognize
  – People are sick; machines go down, etc.

How to keep it on track?

• First, have a schedule!
• Second, have milestones
  – Not “coding complete”
  – But “specifications signed by architects”
  – Or “debugged component passes all tests”
    • government data
      – estimates made and revised two weeks early do not change as the
        start time draws near, no matter how wrong they end up being
      – overestimates come steadily down as the activity proceeds
    – underestimates do not change until scheduled time draws near

• Third, track the critical path
  – who is waiting on who to finish what
• Fourth, address the “status disclosure problem”
  – Managers must distinguish between action meetings and status meetings
    • If inappropriate action is taken in response to a status report, it
      discourages honest status reports
    – better to schedule an action meeting after the true status is known
  – Rule of thumb on schedules: have two dates “scheduled” and
    “estimated”
    • the former is owned by the top level product manager
    • the latter is owned by the manager directly involved with the artifact

The Other Face

• A program needs to be well-documented
  – Thomas J. Watson and the cash registers
• Document how to use the program
  – purpose, environment, I/O formats, options, etc.
• Document how to believe the program
  – Test cases
• Document how to modify the program
  – architecture diagrams, algorithm description, file
    hierarchy, data-flow, extensibility mechanisms, etc.