Most software products are too large for one person to finish in time allotted
Example: product must be completed within 3 months, but 1 person-year of programming is still needed
Simple solution:
- If one programmer can code the product in 1 year, four programmers can do it in 3 months
Not really a solution:
- It still will probably take nearly a year
- The quality of the product is usually lower

Two extremes:
- If one farm hand can pick a strawberry field in 10 days, ten farm hands can pick the same strawberry field in 1 day
- One elephant can produce a calf in 22 months, but 22 elephants cannot possibly produce that calf in 1 month

Producing software is in-between
- Unlike elephant production, it is possible to share coding tasks between members of a team
- Unlike strawberry picking, team members must interact in a meaningful and effective way

What can go wrong?
- Both Sheila and Harry may code m1, and ignore m2
- Sheila may code m1, Harry may code m2. When m1 calls m2, it passes 4 parameters; but m2 requires 5 parameters
- Or, the order of parameters in m1 and m2 may be different
- Or, the order may be same, but the data types may be slightly different
- This has nothing whatsoever to do with technical competency
- Team organization is a managerial issue

Example
- There are three channels of communication between the three programmers working on a project. The deadline is rapidly approaching but the code is not nearly complete
- “Obvious” solution:
  - Add a fourth programmer to the team
Communications Problems (2)

- But other three have to explain in detail
  - What has been accomplished
  - What is still incomplete
- Brooks's Law
  - Adding additional programming personnel to a team when a product is late has the effect of making the product even later

Team Organization

- Teams are used throughout the software production process
  - But especially during implementation
    - Here, the discussion is presented within the context of programming teams
- Two extreme approaches to team organization
  - Democratic teams (Weinberg, 1971)
  - Chief programmer teams (Brooks, 1971; Baker, 1972)

Democratic Team Approach

- Basic underlying concept — egoless programming
- Programmers can be highly attached to their code
  - They even name their modules after themselves
  - They see their modules as extension of themselves

Democratic Team Approach (2)

- If a programmer sees a module as an extension of his/her ego, he/she is not going to try to find all the errors in "his"/"her" code
  - If there is an error, it is termed a bug
  - The fault could have been prevented if the code had been better guarded against the "bug"
  - "Shoo-Bug" aerosol spray

Democratic Team Approach (3)

- Proposed solution: Egoless programming
  - Restructure the social environment
  - Restructure programmers’ values
  - Encourage team members to find faults in code
  - A fault must be considered a normal and accepted event

Democratic Team Approach (4)

- The team as whole will develop an ethos, a group identity
- Modules will "belong" to the team as whole
- A group of up to 10 egoless programmers constitutes a democratic team
Difficulties with Democratic Team Approach

- Management may have difficulties
  - Democratic teams are hard to introduce into an undemocratic environment
  - Confusion about who to reward
- Experienced programmers may not appreciate others critiques

Strengths of Democratic Team Approach

- Democratic teams are enormously productive
- They work best when the problem is difficult
- They function well in a research environment
- Problem:
  - Democratic teams have to spring up spontaneously

Classical Chief Programmer Team Approach

- Consider a 6-person team (Figure 4.2)
  - Fifteen 2-person communication channels
  - The total number of 2-, 3-, 4-, 5-, and 6-person groups is 57
  - This team cannot do 6 person-months of work in 1 month

Classical Chief Programmer Team

- Six programmers, but now only 5 lines of communication (Figure 4.3)

Classical Chief Programmer Team (2)

- The basic idea behind the concept
  - Analogy: chief surgeon directing an operation, assisted by
    - Other surgeons
    - Anesthesiologists
    - Nurses
    - Other experts, such as cardiologists, nephrologists
- Two key aspects
  - Specialization
  - Hierarchy

Classical Chief Programmer Team (3)

- Chief programmer
  - Successful manager and highly skilled programmer
  - Does the architectural design
  - Allocates coding among the team members
  - Writes the critical (or complex) sections of the code
  - Handles all the interfacing issues
  - Reviews the work of the other team members
  - Is personally responsible for every line of code
Classical Chief Programmer Team (4)
- Back-up programmer
  - Necessary only because the chief programmer is human
  - The back-up programmer must be in every way as competent as the chief programmer, and
  - Must know as much about the project as the chief programmer
  - The back-up programmer does black-box test case planning and other tasks that are independent of the design process

Classical Chief Programmer Team (5)
- Programming secretary
  - A highly skilled, well paid, central member of the chief programmer team
  - Responsible for maintaining the program production library (documentation of the project), including:
    - Source code listings
    - JCL
    - Test data

Classical Chief Programmer Team (6)
- Programmers hand their source code to the secretary who is responsible for
  - Conversion to machine-readable form
  - Compilation, linking, loading, execution, and running test cases (this was 1971, remember!)
- Programmers
  - Do nothing but program
  - All other aspects are handled by the programming secretary

The New York Times Project
- Chief programmer team concept
  - First used in 1971
  - By IBM
  - To automate the clippings data bank ("morgue") of the New York Times and other publications
- Chief programmer — F. Terry Baker

The New York Times Project (2)
- 83,000 source lines of code (LOC) were written in 22 calendar months, representing 11 person-years
- After the first year, only the file maintenance system had been written (12,000 LOC)
- Most code was written in the last 6 months
- Only 21 faults were detected in the first 5 weeks of acceptance testing

The New York Times Project (3)
- 25 further faults were detected in the first year of operation
- Principal programmers averaged one detected fault and 10,000 LOC per person-year
- The file maintenance system, delivered 1 week after coding was completed, operated 20 months before a single failure occurred
- Almost half the subprograms (usually 200 to 400 lines of PL/I) were correct at first compilation
The *New York Times* Project (4)

- But, after this fantastic success, no comparable claims for the chief programmer team concept have been made

Why Was the *NYT* Project Such a Success?

- Prestige project for IBM
  - First real trial for PL/I (developed by IBM)
  - IBM, with superb software experts, used its best people
- Extremely strong technical backup
  - PL/I compiler writers helped the programmers
  - JCL experts assisted with the job control language

Why Was the *NYT* Project Such a Success? (2)

- F. Terry Baker
  - Superprogrammer
  - Superb manager and leader
  - His skills, enthusiasm, and personality "carried" the project
- Strengths of the chief programmer team approach
  - It works when chief programmer is good
  - Numerous successful projects have used variants of CPT

Impracticality of Classical CPT

- The chief programmer must be a highly skilled programmer *and* a successful manager
  - There is a shortage of highly skilled programmers
  - There is a shortage of successful managers
  - The qualities needed to be a highly skilled programmer are unlikely to be found in a successful manager, and vice versa

Impracticality of Classical CPT (2)

- The *back-up programmer* must be as good as the chief programmer
  - But he/she must take a back seat (and a lower salary) waiting for something to happen to the chief programmer
  - Top programmers, top managers will not do that
- The *programming secretary* does nothing but paperwork all day

Beyond CP and Democratic Teams

- We need ways to organize teams that
  - Make use of the strengths of democratic teams and chief programmer teams, and
  - Can handle teams of 20 (or 120) programmers
- A strength of democratic teams
  - A positive attitude to finding faults
- Use CPT in conjunction with code walkthroughs or inspections
Beyond CP and Democratic Teams (2)

- Potential pitfall:
  - The chief programmer is personally responsible for every line of code
    - He/she must therefore be present at reviews
  - The chief programmer is also the team manager
    - He/she must therefore not be present at reviews!

Beyond CP and Democratic Teams (3)

- Solution (Figure 4.4)
  - Reduce the managerial role of the chief programmer

Beyond CP and Democratic Teams (4)

- It is easier to find a team leader than a chief programmer
- Each employee is responsible to exactly one manager — lines of responsibility are clearly delineated
- The team leader is responsible for only technical management

Beyond CP and Democratic Teams (5)

- Budgetary and legal issues, and performance appraisal are not handled by the team leader
- The team leader participates in reviews — the team manager is not permitted to do so
- The team manager participates in regular team meetings to appraise the technical skills of the team members

Larger Projects

- For even larger products, add additional layers (Figure 4.5)
  - The nontechnical side is similar

Larger Projects (2)

- Decentralize the decision-making process, where appropriate (Figure 4.6)
  - Useful where the democratic team is good
Problems 4.4, 4.5

- A student programming team is organized as a democratic team. What can be deduced about the students in the team?
- A student programming team is organized as a chief programmer team. What can be deduced about the students in the team?