Teams

Xiaojun Qi

Team Organization

- Suppose that a product can be accomplished by 1 person-year of programming. If this product must be completed within 3 months, how should we do?
  
  - Solution:
    - If one programmer can code the product in 1 year, four programmers can do it in 3 months
  
  - Reality:
    - 4 programmers will probably take nearly a year
    - The quality of the product is usually lower

Task Sharing

- If one farm hand can pick a strawberry field in 10 days, ten farm hands can pick the same strawberry field in 1 day
  - Unlike strawberry picking, team members must interact in a meaningful and effective way

- One elephant can produce a calf in 22 months, but 22 elephants cannot possibly produce that calf in 1 month
  - Unlike elephant production, it is possible to share coding tasks between members of a team

Programming Team Organization

- Team organization has nothing to do with the technical competency of the programmer.

- Team organization is a managerial issue

- Management must organize the programming teams so that each team is highly productive.

Communications Problems

- 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 (Cont.)

- Other three programmers have to explain to the fourth programmer 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
  - The following discussions are presented within the context of programming teams

- **Two extreme approaches** to programming-team organization
  - Democratic teams (Weinberg, 1971)
  - Chief programmer teams (Brooks, 1971; Baker, 1972)

Democratic Team Approach: Problems

- Programmers can be highly attached to their code
  - They name their modules after themselves
  - They see their modules as an extension of themselves

- 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

Democratic Team Approach: Solution

- **Egoless programming**
  - Restructure the social environment
  - Restructure the programmers’ values
  - Encourage team members to find faults in code
  - A fault must be considered a normal and accepted event
  - The team as a whole will develop an ethos, a group identity
  - Modules will “belong” to the team as a whole
  - A group of up to 10 egoless programmers constitutes a **democratic team**

Egoless Programming: Strengths and Problems

- **Strengths:**
  - Democratic teams are enormously productive
  - They work best when the problem is difficult
  - They function well in a research environment

- **Problems:**
  - Democratic teams have to spring up spontaneously
  - Management may have difficulty
  - Democratic teams are difficult to introduce into an undemocratic environment

Classical Chief Programmer Team Approach

- Six programmers with only 5 lines of communication

Classical Chief Programmer Team Approach: Key Aspects

- **Specialization:**
  - Each member of the team carries out only those tasks for which he or she has been trained.

- **Hierarchy:**
  - The chief programmer directs the actions of all the other members of the team and is responsible for every aspect of the operation.
Classical Chief Programmer Team Approach: 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 Approach: Back-up Programmer

- Necessary only because the chief programmer is human
- Must be in every way as competent as the chief programmer
- Must know as much about the project as the chief programmer
- Does black-box test case planning and other tasks that are independent of the design process so the chief programmer can concentrate on the architecture design

Classical Chief Programmer Team Approach: 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 (Job Control Language)
  - Test data
- Convert source code to machine-readable form, compile, link, load, execute and run test cases (1971, remember!)

Classical Chief Programmer Team Approach: Programmers

- Do nothing but program
- All other aspects are handled by the programming secretary

Successful Stories:

- The "New York Times" project by IBM.
- However, after this fantastic success, no comparable claims for the chief programmer team concept have been made

NYT Project: Why Successful?

- Prestige project for IBM
  - First real trial for PL/I (developed by IBM)
  - IBM, with superb software experts, used its best people
- Very strong technical backup
  - PL/I compiler writers helped the programmers
  - JCL experts assisted with the job control language
- F. Terry Baker
  - Superprogrammer
  - Superb manager and leader
  - His skills, enthusiasm, and personality "carried" the project

Impracticality of Classical CPT

- The chances of finding a chief programmer, a back-up programmer, and a programming secretary are small since:
  - The qualities needed to be a highly skilled programmer appear to be different from those needed to be a successful manager, and vice versa.
  - The back-up programmer 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.
- Classical CPT is impractical. Therefore, numerous successful projects have used variants of CPT
Beyond Chief Programmer (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 more than 10 programmers

- A strength of democratic teams
  - A positive attitude to finding faults

Beyond CP and Democratic Teams: Potential Pitfall

- The potential pitfall of using CPT in conjunction with code walkthroughs or inspections

- 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: Solution

- Solution
  - Reduce the managerial role of the chief programmer

Beyond CP and Democratic Teams: Team Leader

- It is easier to find a team leader than a chief programmer

- The team leader:
  - Is in charge of the technical aspect of the team’s activities. That is, budgetary and legal issues, and performance appraisal are not handled by the team leader.
  - Participates in reviews

Beyond CP and Democratic Teams: Team Manager

- It is easier to find a team manager than a chief programmer

- Team manager:
  - Is responsible for all nontechnical managerial decisions.
  - Participates in regular team meetings to appraise the technical skills of the team members

Larger Projects

- The nontechnical side is similar
  - For even larger products, add additional layers
Decentralize the Decision-Making Process

- Decentralize the decision-making process, where appropriate
  - Useful where the democratic team is good

Synchronize-and-Stabilize Teams

- Used by Microsoft
- Products consist of 3 or 4 sequential builds
- Small parallel teams
  - 3 to 8 developers
  - 3 to 8 testers (work one-to-one with developers)
  - The team is given the overall task specification
  - The individual team members may design the task as they wish

Synchronize-and-Stabilize Teams (Cont.)

- Why this does not degenerate into hacker-induced chaos?
  - Individual components always work together
    - Individual programmers are encouraged to be creative and innovative, a characteristic of a democratic team
  - Daily synchronization step
    - Rule: Programmers must adhere to the time for entering the code into the database for that day’s synchronization
- Alternate viewpoint
  - It is simply a way of allowing a group of experts to develop large products
  - Microsoft’s success is due to superb marketing rather than quality software

Extreme Programming Teams

- Feature of XP
  - “Pair programming”: All code is written by two programmers sharing a computer
- Strengths of pair programming:
  - Programmers should not test their own code
    - One programmer draws up the test cases, the other tests the code
  - If one programmer leaves, the other is sufficiently knowledgeable to continue working with another pair programmer
  - An inexperienced programmer can learn from his or her more experienced team member
  - Centralized computers promote egoless programming

People Capability Maturity Model (P-CMM)

- P-CMM describes best practices for managing and developing the workforce of an organization
- Each maturity level has its own KPAs (Key Process Areas)
  - **Level 2: The managed level.** The KPAs are staffing, communication and coordination, training and development, work environment, performance management, compensation
  - **Level 5: The optimizing level.** The KPAs are Continuous capability improvement, organizational performance alignment, continuous workforce innovation

P-CMM (Cont.)

- P–CMM is a framework for improving an organization’s processes for managing and developing its workforce
- No specific approach to team organization is put forward
Choosing an Appropriate Team Organization

- There is no one solution to the problem of team organization

- The "correct" way depends on
  - The product to be built
  - The culture of the organization (i.e., the outlook of the leaders of the organization)
  - Previous experience with various team structures

Choosing an Appropriate Team Organization (Cont.)

- Very little research has been done on software team organization
  - Instead, team organization has been based on research on group dynamics in general

  - Without relevant experimental results, it is hard to determine optimal team organization for a specific product

---

<table>
<thead>
<tr>
<th>Team Organization</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Democratic teams (Section 4.2)</td>
<td>High-quality code as a consequence of positive attitude to finding faults; particularly good with hard problems; Major success of New Ford two project</td>
<td>Excessive staff turnover; Ever-code being appraised by engineers; Cannot be externally imposed; Impractical</td>
</tr>
<tr>
<td>Classical chief programmer teams</td>
<td>Many successes</td>
<td></td>
</tr>
<tr>
<td>Modified chief programmer teams (Section 4.3.7)</td>
<td>Team managers own leader structure; stimates need for chief programmer; Support decentralized work needed;</td>
<td>His success comparable to the New Ford two project;</td>
</tr>
<tr>
<td>Modern hierarchical programming teams (Section 4.4)</td>
<td>Ensures creativity; Ensures that a large number of developers can work toward a common goal; Programs do not need their own code; Not that they have programmer teams; Inter-experience programmers can learn from others; Group ownership of code</td>
<td>His success so far that this method can be offered outside research; Still very little evidence regarding efficacy</td>
</tr>
<tr>
<td>Synchoronized cold-ribbon teams (Section 4.5)</td>
<td>Ensures creativity;</td>
<td>His success so far that this method can be offered outside research;</td>
</tr>
<tr>
<td>External programming teams</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.7