PROJECT MANAGEMENT
CONCEPTS & TEAMS

Overview

• Basic project management concepts: Four P’s
  • People – today’s lecture
  • Product – throughout the semester
  • Process – last week
  • Project – at the end of the semester

• Different team organizations
  • Democratic team approach
  • Classical chief programmer team approach
  • Beyond chief programmer and democratic teams
  • Synchronize-and-stabilize teams
  • Extreme programming teams

Project Management

• Project management involves the planning, monitoring, and control of the people, process, and events that occur as software evolves from a preliminary concept to an operational implementation
  • Software engineers manage day-to-day activities, planning, monitoring, and controlling technical tasks
  • Project managers plan, monitor, and control the work of a team of software engineers
  • Senior managers coordinate the interface between the business and software professionals
Four P's: People, Product, Process, Project

- People must be organized to perform software work effectively
  - PM-CMM – enhance the readiness of software organizations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability
  - Having the right people is not enough; teams must be organized

Four P's: People, Product, Process, Project

- Product
  - Objectives and scope established
  - Requirements understood
  - Alternative solutions considered
  - Technical and management constraints identified
  - Process must be selected that is appropriate for the people and the product
    - Process model chosen
    - Umbrella activities (software quality assurance, software configuration management, and measurement)

Four P's: People, Product, Process, Project

- Project must be planned by estimating effort and calendar time
  - Defining work products, establishing quality checkpoints, and establishing mechanisms to monitor and control work defined by the plan
  - Still a struggle: in 1998, 26% of software projects failed and 46% experienced cost and schedule overruns
Indications that a project is in jeopardy

- Software people do not understand customer’s needs
- Product scope is poorly defined
- Changes are managed poorly
- Chosen technology changes
- Business needs change or are ill-defined
- Deadlines are unrealistic
- Users are resistant
- Sponsorship is lost or was never properly obtained
- Project team lacks people with appropriate skills
- Managers and practitioners avoid best practices and lessons learned

How to avoid problems?

- Start on the right foot
  - Understand the problem well, set realistic objectives and expectations, build the right team, give the team autonomy, authority, and technology needed to do the job
- Maintain momentum
  - Project manager must provide incentives, the team should emphasize quality in every task
- Track progress
  - Work products are produced and approved as part of a quality assurance activity, software process and product metrics are collected and used to assess progress
- Make smart decisions
  - “Keep it simple”, use COTS, standard approaches, identify and avoid risks, allocate more time to complex or risky tasks
- Conduct a postmortem analysis
  - Lessons learned and best practices in written form

People

- “Companies that sensibly manage their investment in people will prosper in the long run” – Tom DeMarco & Tim Lister
- Stakeholders
  - Senior managers – define the business issues that often have significant influence on the project
  - Project (technical) managers – plan, motivate, organize, and control the software practitioners
  - Software practitioners – engineer a product or application
  - Customers – specify the requirements
  - End-users – interact with the software product once it is released
There are almost as many organizational structures for software development as there are organizations that develop software.

- Organizational structure cannot be easily modified.
- Formal team organization is more productive.
- The best team organization depends on the management style in the organization, number of people who will populate the team and their skill levels, and the overall problem difficulty.

Is 1 person x 1 year = 3 persons x 4 months?

- 3 persons may take nearly a year.
- The quality of the product may be lower.

Why?

- Task sharing
  - Perfectly partitionable tasks
    - If 1 person can pick a strawberry field in 9 days, 9 persons can pick the same strawberry field in 1 day.
  - Unpartitionable tasks
    - Bearing of a child takes 9 months, no matter how many women are assigned.

- Communication overhead
  - In tasks that can be partitioned but which require communication, the effort of communication must be added to the amount of work to be done.
Task Sharing & Communication Overhead

- Unlike bearing of a child, it is possible to share coding tasks between members of team
  - Implementation phase is a prime candidate for sharing tasks
- Unlike strawberry picking, team members must interact in meaningful and effective way
  - Team organization is a managerial issue

Example

3 developers are working on a project
- Problem - deadline is rapidly approaching but the product is not nearly complete
- “Obvious” solution - add a fourth programmer to the team
- Other three have to explain in detail
  - What has been accomplished
  - What is still incomplete

Brooks’s law & other quotes

- Brooks’s law: Adding additional personnel to a late software project makes it even later
- Other quotes from Brooks’s “The Mythical Man-Month”
  - Cost does indeed vary as the product of the number of men and the number of months. Progress does not.
  - More programming projects have gone awry for lack of calendar time than for all other causes combined.
  - Good cooking takes time; some tasks cannot be hurried without spoiling the result.
  - Partitioning a task among multiple people occasions extra communication effort – training and intercommunication.
Overview

- Basic project management concepts: Four P's
  - People – today's lecture
  - Product – throughout the semester
  - Process – last week
  - Project – at the end of the semester

- Different team organizations
  - Democratic team approach
  - Classical chief programmer team approach
  - Beyond chief programmer and democratic teams
  - Synchronize-and-stabilize teams
  - Extreme programming teams

Democratic Team

- Programmers can be highly attached to their code
  - Name their modules after themselves
  - See their modules as extension of themselves
  - Don't try to find all the faults (defects) in their code

- Solution - egoless programming
  - Restructure the social environment and programmers' values
  - Encourage team members to find faults in code
  - Faults are considered normal and accepted events
  - Team as whole will develop group identity
  - Modules belong to the team as whole

Democratic Team (contd)

- Democratic team - group of up to 10 egoless programmers
  - No single leader, no one is trying to get promoted
  - Communication among team members is horizontal
  - Important – team identity and mutual respect

- Advantages
  - Enormously productive
  - Work best when the problem is difficult
  - Function well in a research environment
  - Positive attitude toward finding faults

- Problems
  - Difficult to introduce into an undemocratic environment
  - Democratic teams have to spring up spontaneously
  - Management may have difficulty
Consider a 6-person team
- 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

Six programmers, but now only 5 lines of communication
- Analogy - chief surgeon directing operation, assisted by other surgeons, anesthesiologist, nurses, and other experts (e.g. cardiologists, nephrologists)
- Two key aspects
  - Specialization
  - Hierarchy – communication with the leader (Chief programmer) is vertical

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 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 (contd)

- Back-up programmer - necessary only because the chief programmer is human
  - Must be 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

Classical Chief Programmer Team (contd)

- Programming secretary - highly skilled, well paid, central member of the team
  - Maintained the program production library (documentation of project), including source code listings and test data
  - Programmers handed their source code to the secretary who is responsible for conversion to machine-readable form, compilation, linking, loading, execution, and running test cases (in 1971 keypunches were still widely used)
  - Programmers did nothing but program; All other aspects were handled by the programming secretary

Chief Programmer Team - 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

- Results were impressive
The New York Times Project (contd)

- 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
- Principal programmers averaged 10,000 LOC per person-year and one detected fault
- Almost half the subprograms (usually 200 to 400 lines of PL/I) were correct at first compilation

The New York Times Project (contd)

- Only 21 faults were detected in the first 5 weeks of acceptance testing
- Only 25 further faults were detected in the first year of operation
- File maintenance system operated 20 months before a single failure occurred

The New York Times Project (contd)

- Many successful projects have been carried out using chief programmer team approach
- Although satisfactory, reported figures are not as impressive as those obtained for the New York Times project – Why?
Why The New York Times Project worked?

- 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 support
  - PL/I compiler writers helped the programmers
  - JCL experts assisted with the job control language
- Chief programmer - F. Terry Baker
  - Superprogrammer
  - Superb manager and leader
  - His skills, enthusiasm, and personality carried the project

Why is Classical CPT impractical?

- Chief programmer must be a highly skilled designer, programmer, and manager
  - Shortage of highly skilled designers and programmers
  - Shortage of successful managers
- Back-up programmer must be as good as the chief programmer
  - But must take a back seat (and a lower salary)
  - Top programmers and managers will not accept such a role
- Programming secretary does only paperwork all day
  - Software professionals hate paperwork

Beyond Classical Chief Programmer Team

- Problem:
  - Chief programmer is personally responsible for every line of code - Must be present at reviews
  - Chief programmer is also team manager - Must not be present at reviews
- Solution - Two individuals
  - Team leader – responsible for technical aspects
  - Team manager – responsible for non-technical managerial aspects (budgetary, legal issues, and performance appraisals)
Beyond Classical Chief Programmer Team (contd)

- Team leader participates in reviews; team manager is not permitted to do so
- Team manager participates at regular team meetings to appraise technical skills of the team members

This approach is scalable for larger projects
- Additional layers can be added

Beyond Chief Programmer and Democratic Teams

- We need ways to organize teams that
  - Make use of the strengths of both democratic teams and chief programmer teams
  - Can handle teams of 20 (or 120) programmers
Beyond CP and Democratic Teams (contd)

- Project leader who coordinates specific tasks and team leaders that are responsible for subtasks
- Decentralize the decision-making process where appropriate
- Useful where the democratic team approach is good
- Communication among subgroups and individuals is horizontal
- Arrows from level to level still point downward (vertical communication)

---

Synchronize-and-Stabilize Teams

- Used by Microsoft
  - Windows 2000 – 30 millions LOC, build by over 3000 programmers and testers
- Products consist of 3 or 4 sequential builds
- Each build is constructed by small parallel teams
  - 3 to 8 developers
  - 3 to 8 testers (work one-to-one with developers)
  - Team is given the specifications of their overall task
  - Each team member has a freedom to design and implement their portion as they wish

---

Synchronize-and-Stabilize Teams (contd)

- Daily synchronization step – partially completed components are tested and debugged; Individual components always work together
- Few rules
  - Must adhere to the time to enter the code into the database for that day’s synchronization
  - Analogy - letting children do what they like all day … but have to be in bed by 9 pm
  - If developer’s code prevents the build from being compiled for that day synchronization, the problem must be fixed immediately
Synchronize-and-Stabilize Teams (contd)

- Individual programmers are encouraged to be creative and innovative – characteristic of democratic team
- Daily synchronization step - hundreds of developers work toward common goal without communication and coordination characteristics of chief programmer team
- Will this work in every company?
  - Microsoft consists of a highly talented set of managers and software developers
  - Again, more research is needed

Extreme Programming Teams

- XP is designed for use with small teams who need to develop software quickly in an environment of rapidly-changing requirements
- Two programmers work in a collaborative fashion on the same design, algorithm, code or test sharing a single computer - pair programming
  - First, one programmer draws up test cases for the task, then jointly with the other programmer implement the code of the task
- **Strengths**
  - If one member of the team leaves, the other has sufficient knowledge to continue working
  - Less experienced professional can learn from more experienced team member
  - Group ownership of the code – feature of egoless team

Comparison of various team organizations

<table>
<thead>
<tr>
<th>Team organization</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Democratic team</td>
<td>• Positive attitude to finding faults</td>
<td>• Cannot be externally imposed</td>
</tr>
<tr>
<td></td>
<td>• Good for hard problems</td>
<td>• High volume of communication</td>
</tr>
<tr>
<td></td>
<td>• Teams that are together for long time</td>
<td>• Require more time</td>
</tr>
<tr>
<td>Classical chief programming team</td>
<td>• Many successes</td>
<td>• It is hard to find a chief programmer and back-up programmer</td>
</tr>
<tr>
<td>Modern hierarchical programming team</td>
<td>• Separate team manager/team leader</td>
<td>• Problems arise if responsibilities of team manager and team leader are not clearly delineated</td>
</tr>
<tr>
<td></td>
<td>• Scales up</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Works well for high modularity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Supports decentralization</td>
<td></td>
</tr>
<tr>
<td>Synchronize-and-stabilize team</td>
<td>• Encourages creativity</td>
<td>• Not utilized outside Microsoft</td>
</tr>
<tr>
<td></td>
<td>• Ensures that hundreds developers work toward common goal</td>
<td></td>
</tr>
<tr>
<td>Extreme programming team</td>
<td>• Knowledge is not lost if one programmer leaves</td>
<td>• Only small teams</td>
</tr>
<tr>
<td></td>
<td>• Less experience programmer can learn</td>
<td>• Controversial, especially for larger systems</td>
</tr>
<tr>
<td></td>
<td>• Group ownership of code</td>
<td>• Little evidence regarding its efficacy</td>
</tr>
</tbody>
</table>
Final Remarks

- No single solution for the team organization
- The “best” team organization depends on
  - Product
    o Difficulty of the problem to be solved
    o Degree to which the problem can be modularized
    o Size of program(s) in lines of code or function points
    o Required quality and reliability of the system to be build
  - Process & Organization
    o Management style in the organization
    o Number of people who will populate the team and their skill levels
    o Previous experience with various team structures
  - Project
    o Rigidity of the delivery date
    o Degree of communication required for the project
    o Cost

---

Final Remarks

- Very little research has been done on software team organization
- Without relevant experimental results, it is hard to determine optimal team organization for a specific product