Session 9: Alternative Project Methodologies

What's wrong with mainstream project planning & control?
Recent fads
What does a Project Manager need to know about them?

Our changing profession

- Every three or four years, some new approach or methodology is publicized.
- Some new methodologies are evolutionary.
  - We can integrate the new techniques into what we already know and practice
- Others are revolutionary
  - We have to stop doing what we do now in order to start doing something very different.
- Why is that distinction important? Examples?

Evolving traditions

- Supporters often cite project failures using the "traditional approach".
  - What are some of them?
    - Which ones affect project management?
- Yesterday's major breakthrough becomes tomorrow's tradition!
  - So the "traditional approach" is not always bad.
  - It needn't always be avoided.

Some alternative project management methodologies

- RAD (Rapid Application Development)
- XP (Extreme programming)
- Other "Agile" techniques
  - Incremental development with "emerging specifications"
  - "Sprints" and "stories"
  - Status reporting through daily (!) "stand-up" meetings
  - etc.
What's wrong with mainstream project planning & control

- Nothing!

- But:
  1. In the young computer field, practitioners (managers, too) often believe that new must be better.
  2. Many projects fail that appear to be practicing mainstream project management. Why?

Why do mainstream projects fail?

- We've already seen these reasons:
  1. Insufficient detail in the project plan
  2. Inadequate ESD (user requirements)
  3. Careless test plan

- Here's one more:
  4. Lack of management discipline

    - We know what we should do, and we know that compromise will lead to failure, but we compromise, anyway! Why?

Lack of management discipline

- Arises most often in negotiations over target dates
  - We prepare a well-reasoned project plan and present it to the client.
  - Client is disappointed in the schedule
    - states a deadline that absolutely must be met.
    - doesn't grasp the magnitude and complexity of the job; it sounds simple and straightforward.

- We'd like to please the client, so what should we do?
  a. Propose a reduced-scope application system?
  b. Cave in and make a rash commitment?

A common faulty assumption

- "You can commit to just about anything in 18 months!" Why?
  - That's a long way off.

    - Something good may happen before then Such as?
    - We'll work extra hard.
    - ("I may leave the company by that time." Someone else's problem then!)

- It keeps the customer happy (for now) and takes the immediate pressure off.
One negotiating step
- When confronted by an impossible user-mandated deadline, prepare a detailed project plan (at least for the next phase)
  - Show it to the client. Ask
    - "What tasks on this plan would you like us not to do?"
  - The clients may then realize that the job is larger than they originally expected, and may agree to the rational schedule and budget.
  - Or we may lose the client to a competitor willing to underbid.

Rapid Application Development (RAD) overview
- Follows a conventional life cycle
  - The analysis phases are accelerated
  - Programming relies on 4GL wherever possible

RAD Accelerated Analysis Phase
1. Round up every participant
   - Stakeholders representing all affected user departments at all levels
   - Systems analysts
   - Data administrator
   - Project manager (if "hands on")
   - et al.
2. Put them in a conference room
   - Don't let them out until requirements (ESD) are done!
   - Sessions are led not by a chairman but by a facilitator.

The RAD facilitator role
- Not the most senior or highest-ranking person in the meeting.
  - so that participants speak freely and won't feel intimidated by the chair.
- Leads the session.
  - Makes sure that
    - All points of view are heard and understood
- Strives for consensus
  - Mediates disagreements
  - Refrains from expressing his / her own opinion.
- Makes sure results are captured.

Is that worse than underbidding ourselves?
Promoted by IBM and others ~1985

What's that?
How well does that work?
For what kinds of project?
**RAD advantage**
- Avoids delays waiting for interview appointments and returned phone calls.
- Identifies conflicting requirements within the user organization sooner.
- Assures conflicts are resolved before requirements are documented.

**RAD disadvantage**
- Fatigue may lead to
  - problem oversights
  - solution oversights
  - poorly thought-out ideas or conflict resolution
  - careless errors

---

**"Agile" methodologies**
- Began with "Extreme Programming" ~1999
  - Chrysler project (Kent Beck et al)
- Various other versions since then
  - Some common characteristics compared with mainstream approaches:
    - Less planning
    - Less documentation
    - More very short meetings
    - Earlier code writing
    - YAGNI
    - Frequent "Refactoring"

---

**YAGNI (You aren't going to need it!) principle**
- A reaction against overdoing fancy, often costly to implement, features. Focus on what's truly essential to the organization.

- But we need to distinguish between:
  - sophisticated features that are really advanced and not really essential, and
  - sensible features that may not have been the first thing we thought of.

*Remember the Y2K crisis?*

---

**The "Agile Manifesto"**
- We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan

- That is, while there is value in the items on the right, we value the items on the left more.
  - from a 2001 meeting of 17 "agile" innovators

*What does that mean?*
*What are its likely consequences?*
"Agile" impact on project planning

- Don't do it! "That's BUFP and it doesn't work!"
- Plan one "sprint" at a time:
  - Calendar-driven rather than phase-driven
  - We deliver *something* every sprint
  - Then we decide what to do in the next sprint
  - The backlog consists of a pile of "stories"
  - Deliver working code every sprint

User stories

- An unstructured pile of brief documents, which may be:
  - discrete requirements ("The system shall . . .")
  - data definitions
  - business rules, formulas
  - project task specifications
- Not necessarily rigorous
  - Serve as "reminders" of issues to be discussed
- They are commonly written on cards.
- Users and developers cooperate to *prioritize* them for each sprint.

An embarrassing question

- Under an *incremental approach*, how do we determine whether a project is justified?
- How do we know what it's going to cost?
- How do we know how long it's going to take?

"Agile" impact on task status reporting

- While short, daily stand-up meetings are a far more effective way of communicating status than written reports, many traditional managers still insist on documentation. These reports take valuable time—time that could be better spent developing working software."
  - Scott Ambler (IBM)
- *How much time does each actually take?*
Task status reporting

- We already discussed a preference for strict binary status reports. A task is either
  - complete, with deliverables available
  - incomplete, with estimated completion date and remaining cost

- Percentage complete (or "almost" or "just about" or "virtually" done except for a couple of "loose ends") is rarely meaningful and encourages wishful thinking.

Time spent in status reporting

- How often should team members report the status of their assigned tasks?
  - Weekly is often enough to ensure early warning of schedule slippage or cost overrun
  - but not so often as to be burdensome.

- The team member fills out a simple form provided by the project management system (manual or automated)

Individual's status report

- Each team member gets a form (paper or computer file) containing list of incomplete tasks assigned to him or her
  - along with last week’s estimates

- Checks off any tasks which were completed in the past week.

- Revises any estimates changed since last week, appending an explanation for any that are worse.

- Does nothing for unchanged estimates.

How long will that take?

Three very different kinds of software-development project

1. An application system for a specific user (client, customer) organization
   - An entirely new application
   - Replacement for an existing system
2. A program product to be marketed
3. One or more software components for a library of potentially reusable modules.
   - Which of them is non-mainstream (agile, incremental, RAD, etc.) project methodology well suited to? Why?