Session 14: Alternatives to the phased life cycle

Why change what works now?
But does it really work?
Do popular fad alternatives work any better?
If not, why?

Keeping an open mind
- I.T. is a young discipline. We're still learning:
  - What needs to be done in a software development project.
  - Effective ways of doing those things.
- We should always be willing to listen to and consider adopting new ideas, suggestions, recommendations.
- But we mustn't change what we're doing just for the sake of change or to feel up-to-date or "advanced".

Why change something that works?
- Suspected obsolescence factor
  - We've known about the phased life cycle for decades (since ca. 1976)
  - I.T. technology is rapidly evolving.
  - Therefore, the phased life cycle must be obsolete!
  - Opponents may call it the Waterfall Methodology (even, contemptuously, a Dinosaur Approach)!
  - Why? What do those terms imply?
- Burdensome discipline factor
  - Following a formal life cycle imposes constraints on what project team members do at any time.
  - Modern alternatives promise faster completion.

Methodology and project size
- Fact: Any approach works when the project is small enough!
- Beware of stunning success stories, esp. from enthusiastic sources who may gain from acceptance of their pet approach.
- The opposite of the phased life-cycle approach is the incremental and iterative approach.
  - What's that?
Let's examine UP

- UML promises that we can follow any life cycle we like, as long as that life cycle is:
  - Iterative and incremental (cf. "agile" development)
  - Use-case driven
  - Architecture centric

- But UP (orig. RUP) turns out to be just about the only such life cycle we know about!

Obstacles to following UP

- Where is the critical point in the life cycle?
  - The point where we know (almost) everything about the requirements (the what),
  - but haven't yet committed to any specific implementation (the how).

- In our sample life cycle, it occurs at the end of phase 3.
  - We haven't actually built anything yet.
  - Changes affect only documentation, not software.
  - Competent estimators can assess the rest of the project and make cost & time commitments to the sponsor.

When would this work well?
### A successful UP project
- We must be certain at the start that we're going to **build**, not buy, the software.
- Sponsoring users must be willing to accept broad range of uncertainty in time and cost estimates for the rest of the project.
  - Who knows how many iterations it will take?

**How many projects are like that?**

**How often are we certain at the start?**

**Do the sponsoring users care?**

---

## What if we want to consider buying a packaged software product with UP?
- **Conflict with iterative approach:** There's never a point when we:
  - know everything we need to know to evaluate and select a product, and
  - haven't built software that we'll never need.

**What do many organizations do?**
- Forget about defining rigorous requirements
- Just invite vendors to make presentations and choose the most impressive one

**Does that yield successful results?**

---

### Sponsoring user organizations rarely insist on a rigorous approach
- Because they may feel **intimidated**.
  - After all, the I.T. people are the experts!
  - They're unaware of alternatives.

- Because they're **impatient** to get their new system.
  - They're likely to favor almost any approach that promises at the start to yield tangible and usable results early.

---

### Buying packaged software with a traditional phased life cycle
- At the end of our phase 3, we know enough about our requirements to evaluate vendors' proposals.

- Of course, we can take some shortcuts if we know or strongly suspect at the start that we're likely to buy a packaged product.

**Such as?**