A startling discovery

- "Everything we’ve studied so far in COMP 320 is worthless!"
  
  Good heavens! Who says that?

- "It’s all about BUFP (Big Upfront Planning). We don’t do that any more."
  
  What can we do instead?

- "Emergent specifications"!
  
  How does that work?

"Emergent specifications"

1. Interview user representatives for 30-90 minutes.
2. Write down a few stories representing (among other things) features of a proposed application.
3. Confer with the users to
   - verify that the stories are valid
   - rank the stories by priority.
4. Write program code to implement the highest priority features.
5. Run it and show the output to the sponsoring users.
6. Modify stories and code to conform to users' direction.
7. Repeat the above steps until . . . what?
   
   (At each iteration the collection of stories comprises the emergent specifications for the proposed new application.)

What’s wrong with that?

Note that a pile of user stories lacks structure

- Data definitions, process specifications, business rules, data flows, etc. are scattered throughout the pile.

- How do we know that they're:
  - complete?
  - consistent?

- How can we assess the impact of a new one on work that's already been done?
Agile methods

Emergent specification are one component of the so-called "agile approach" that originated in the 1990s.

Other agile components include:
- extreme programming
- test-driven development
- YAGNI
- sprint-driven delivery

Most of them are characterized by minimal documentation!

The "agile manifesto" favors:
- 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."

Who are "we"?

Original agile inner circle

Kent Beck  Mike Beedle  Arie van Benekum  Alistair Cockburn  Ward Cunningham  Martin Fowler  James Grenning*  Jim Highsmith
Andrew Hunt  Ron Jeffries  Jon Kern  Brian Marick  Robert Martin*  Steve Mellor  Ken Schwaber  Jeff Sutherland  Dave Thomas

An interesting observation

Kent Beck  Mike Beedle  Arie van Benekum  Alistair Cockburn  Ward Cunningham  Martin Fowler  James Grenning*  Jim Highsmith
Andrew Hunt  Ron Jeffries  Jon Kern  Brian Marick  Robert Martin*  Steve Mellor  Ken Schwaber  Jeff Sutherland  Dave Thomas

* based in the Chicago area

Who are "we"?

Based in the Chicago area

What is (or should be) unusual or surprising about that roster?

They met at a ski resort in 2001 to try to codify the principles of their various versions of "agile" methods. Main result was the agile manifesto.
How well does the agile approach work?

- It depends on whom you ask:
  - Some report stunning successes
  - Others report costly fiascos
- Use your common sense.
- Remember that any methodology works well if the problem is small enough!

Major categories of software development project

A. A complete **new application** for one or more sponsoring users.
B. A new **software product** to be sold in the competitive marketplace.
C. A reusable **library component** (or family of related components)

Which of them would best fit the agile paradigm?

Which of them fit the YAGNI (You Aren’t Going to Need It) principle?

A. A complete **new application** for one or more sponsoring users.
B. A new **software product** to be sold in the competitive marketplace.
C. A reusable **library component** (or family of related components)

Serious conflict! Does that rule out YAGNI? Does it rule out agile?

So that leaves one kind of project for which we can consider the agile approach

- A **complete new application** for one or more sponsoring users.
- But that raises other serious issues.
  What are some of them?
Some serious doubts about the agile approach

1. Is it possible to solve the users' problem by purchasing one or more application software products?
2. When can the project team provide reliable time and cost estimates?
3. What use are partial (incremental) solutions in replacing a major system that must continue to run until the replacement system is finished?

Final word on suitable agile projects

Agile development may work well for:

a. A complete new application for one of more sponsoring users, provided that:
   - Everyone is certain at the beginning that no packaged product solution is suitable.
   - User management is willing to fund the project without a firm idea how much it will cost or how long it will take.
   - There is no current system being replaced.

b. A very small application system

The refactoring trap

- Suppose a developer trying to implement a user story discovers that the design structure (database, module hierarchy, O.O classes, user interface, etc.) that emerged in earlier iterations is in conflict with the new feature.
- What now? **Oops! Refactor!**
  - That means: Change the software structure to be consistent with what the developers would have done in the first place, if they'd only known about the new feature.
  - That can be **extremely** costly in the late project stages.

  *Can't that also happen with a non-agile approach?*

Another agile idea: **Test cases as specifications!**

- In theory we could do away with specifications (ESD) altogether and just code test cases to exercise the (still unwritten) code!
- Some extreme agilists claim that a collection of good test cases (driver code) *is* the ESD!

  *Issues:*
  - Who writes those test cases and when?
  - Who can read and understand them?
  - How can the developers exercise their creativity in the user-interface design?