Some have criticized highly-structured processes (such as those based on a CMM framework) as unresponsive to change during a development cycle (e.g., change in requirements or change in technology).

- CMM-type processes are sometimes called “heavyweight” processes or “plan-driven” processes.

- So-called “agile methods” (or “lightweight” processes) have been developed to address this criticism and reduce the cost of change throughout a project.
Agile Development: Definition

- **Agility**
  - The ability to both create and respond to change in order to profit in a turbulent business environment
  - Companies need to determine the amount of agility they need to be competitive

- **Chaordic**
  - Exhibiting properties of both chaos and order
    - The blend of chaos and order inherent in the external environment and in people themselves, argues against the prevailing wisdom about predictability and planning
    - Things get done because people adapt, not because they slavishly follow processes
  - An agile view is a chaordic view
    - “Balanced between chaos and order, perched on the precipice at the edge of chaos.”
    - Some people are not comfortable in this environment; others thrive on it

Current Problem in SW development

31.1% of projects will be canceled before they ever get completed … 52.7% of projects will cost 189% of their original estimates.

– The Standish Group

Plus project complexity is increasing

- Demand for quicker delivery of useful systems
- Increasingly vague, volatile requirements
- Greater uncertainty/risk from limited knowledge of:
  - Underlying technologies
  - Off-the-shelf (OTS) components used

Are conventional development/management practices addressing the problem?
What is the Core Challenge?

What is the nature of system/software development?

- Well-understood, static problems, demanding definable, predictable process, like
  - Manufacturing a widget
  - Constructing a building

  OR

- Exploration of many evolving unknowns, demanding a fluid, social learning process, like
  - Mountain climbing
  - Tracking jet fighters with self-guiding missiles

Agile Manifesto

- Developed by 17 of the leaders in agile methodologies in Feb 2001.

  “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 on the items on the right, we value the items on the left more.”
Supporting Agile “Sentences”

1. Our highest priority is to satisfy the customer through early and frequent delivery of valuable software.
2. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter time scale.
3. Working software/product is the primary measure of progress.
4. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
5. Business people and developers work together daily throughout the project.
6. Build projects around motivated individuals. Give them the environment and support their needs, and trust them to get the job done.

Some Agile Processes

- Adaptive Software Development (ASD)
  - [http://www.agilealliance.com/resources/roadmap/asd/asd_index]

- Crystal Methods
  - [http://www.agilealliance.com/resources/roadmap/crystal/crystal_index]

- Extreme Programming (XP)
  - [http://www.agilealliance.com/resources/roadmap/xp/xp_index]

- Feature Driven Development (FDD)
  - [http://www.agilealliance.com/resources/roadmap/fdd/fdd_index]

- SCRUM
  - [http://www.agilealliance.com/resources/roadmap/scrum/scrum_index]
Extreme Programming

- XP is designed for work in small to medium teams (less than 10 members), building software with vague or rapidly changing requirements.

- The XP life cycle has four basic principles [Beck 2000] :
  - continual communication with the customer and within the team
  - simplicity, achieved by a constant focus on minimalist solutions
  - rapid feedback through unit and functional testing
  - emphasis on dealing with problems proactively

XP Practices (1)

1. **The Planning Game**: Quickly determine the scope of each release by combining business priorities and technical estimates. The customer decides scope, priority, and dates from a business perspective, whereas technical people estimate and track progress.

2. **Small Releases**: Puts simple system into production quickly. Release new version on a very short (2 week) cycle.

3. **Metaphor**: Guide all development with a simple shared story of how the overall system works.

4. **Simple Design**: Design as simply as possible at any given moment.

5. **Testing**: Developers continually write unit tests that must run flawlessly for development to continue; customers write tests to demonstrate that features are finished.

6. **Refactoring**: Programmers restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility.
XP Practices (2)

7. **Pair Programming**: All production code is written by two programmers at one machine.

8. **Continuous integration**: Integrate and build the system many times a day, every time a task is completed.

9. **Collective Ownership**: Anyone can change any code anywhere in the system at any time.

10. **40-hour weeks**: Work no more than 40 hours per week whenever possible; never work overtime two weeks in a row.

11. **On-site Customer**: Have a customer on the team full-time to answer questions.

12. **Coding Standards**: Programmers write code in accordance with rules that emphasize communication throughout the code.

Test Driven Development (TDD)

- An evolutionary approach to development where you must first write a test that fails before you write new functional code.

- Differs from traditional develop then test practices.
Development Cycle

1. Write a single test
2. Compile it. It shouldn’t compile because you haven’t written the implementation code
3. Implement just enough code to get the test to compile
4. Run the test and see it fail
5. Implement just enough code to get the test to pass
6. Run the test and see it pass
7. Re-factor for clarity
8. Repeat

Example

- We want to develop a method that given an integer (x), returns an Integer (y) that is the Fibonacci representation of x.
- Reminder: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …
- Known uses of Fibonacci numbers include
  - analysis of financial markets,
  - computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure
  - [http://www.youtube.com/watch?v=nt2KKUSSJsY](http://www.youtube.com/watch?v=nt2KKUSSJsY)
Benefits

- Reduce gap between decision and feedback
- Encourages developers to write code that is easily tested
- Creates a thorough test bed

Drawbacks

- Time must be taken away from core development
- Some code is difficult to test
Testing frameworks

- **JUnit**
  - A test structure definition
  - Tools to run tests

- *A Simple and Practical Approach to Unit Testing: The JML and JUnit Way* by Y. Cheon and G. T. Leavens

```java
@Test
public void firstFibNumberisZero() {
    assertEquals(0,
    new FibGenerator().getNumberAtPosition(0));
}

public int getNumberAtPosition(int position) {
    return 0;
}

@Test
public void secondFibNumberisOne() {
    assertEquals(1,
    new FibGenerator().getNumberAtPosition(1));
}

public int getNumberAtPosition(int position) {
    return position;
}

@Test
public void thirdFibNumberisOne() {
    assertEquals(1,
    new FibGenerator().getNumberAtPosition(2));
}

public int getNumberAtPosition(int position) {
    return 0;
}

@Test
public void fourthFibNumberisTwo() {
    assertEquals(2,
    new FibGenerator().getNumberAtPosition(3));
}

public int getNumberAtPosition(int position) {
    if (position < 2)
        return position;
    return (position - 1);
}

@Test
public void fifthFibNumberisThree() {
    assertEquals(3,
    new FibGenerator().getNumberAtPosition(4));
}

public int getNumberAtPosition(int position) {
    if (position < 2)
        return position;
    return (position - 1);
}

@Test
public void sixthFibNumberisFive() {
    assertEquals(5,
    new FibGenerator().getNumberAtPosition(5));
}

public int getNumberAtPosition(int position) {
    if (position < 2)
        return position;
    return (getNumberAtPosition(position - 1)) + (getNumberAtPosition(position - 2));
}
```