Lecture Notes on

Microsoft Practice

Bill Scherlis

15-413 ***Software Engineering*** Fall 1999

4 November 1999

Focus

- Software process
  - **Milestones**
  - **Documents**
  - **Code**

- Other areas
  - **Measures**
  - **People**
  - **Standards and competition**
  - **Organizational improvement**

Microsoft Software

- *Example*: Windows 95 (ca. 1996)
  - **11 MLOC**
  - **200 programmers, testers**
  - **One of 250 products**

How Do They Do It?

- Early PC Culture
- Issues
  - **Scale-up product size, complexity, platforms**
  - **Increasing team size**
  - **Hastening time to market**
  - **Managing quality**
- Time to market
  - **Subscriptions**

The Five Principles (Cusumano and Selby)

- Large projects divided into buffered milestone cycles
  - **No separate “maintenance” or “post-release”**
- Vision statement and feature outline
  - **No formal specifications**
- Features selected and prioritized according to market
  - **Early and frequent user release, evaluation**
- Modular architecture
  - **Project structure mirrors product structure**
- Fix project resources; individuals commit to tasks
  - **Drive prioritization & schedule without top-down plans**

Synch-and-Stabilize Cycle, 1

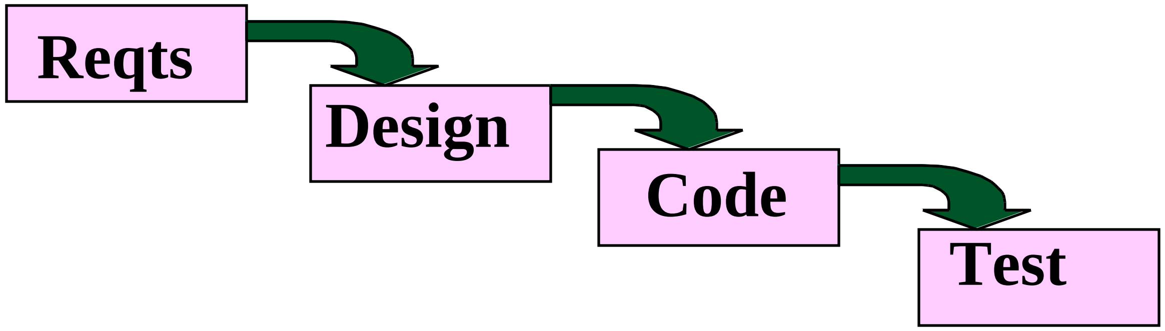
- **Planning phase** (3-12 months)
  - **Vision**
    - **Based on extensive customer input**
    - **Identify and prioritize features**
  - **Specificaiton**
    - **Feature defn. Architecture. Component interdependencies.**
  - **Schedule and feature teams**
    - **Team: 1 PM, 3-8 developers, 3-8 testers**
- Development phase
- Stabilization phase

Roles

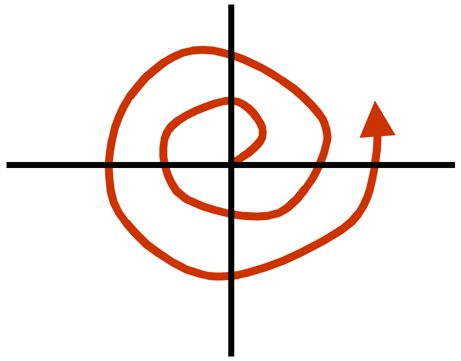
- Product mgmt
  - **Market research. Marketing plan. Beta sites. Launch.**
- Program mgmt
  - **Vision. Spec. Schedule. Comm. Sign-off.**
- Developers
  - **Design. Develop. Debug. Daily build.**
- Usability lab and testers
  - **Usability goals. Development/internal/field testing.**
- Visual interface design
  - **UI design. Icons and bitmaps. Review.**

Process Models

- Waterfall



- Iterative / spiral
  - **Iterate above**
  - **Prototyping cycles**



- Overlapped
  - **Customer in loop**

Synch-and-Stabilize Cycle, 2

- **Development phase** (6-12 months)
  - **3 - 4 sequential subprojects, each with milestone release**
    - **1. First 1/3 of features: Most critical features. Shared comps.**
    - **2. Second 1/3 of features.**
    - **3. Least critical 1/3 of features.**
  - **Continuous testing (tester ↔ developer)**
  - **Milestones (2-4 months each)**
    - **6-10 weeks: code, opts. Test/debug. Feature stabilize.**
    - **2-5 weeks: Integration. Testing.**
    - **2-5 weeks: (buffer time)**
  - **Visual freeze; feature complete; code complete.**
- Stabilization phase

Daily Build, 1

1. Check out
   - **Make changes, compile, test in private copies**
   - **Day or several days**
2. Implement feature
3. Private release
4. Test private release
   - **Test the new feature**
5. Synch code changes
   - **Compare (“synch”) changes with master source**
     - **A code diff**
   - **Insert diring “frozen” period -- e.g., after 2pm.**
6. Merge code changes
   - **Use tool: 5-20 minutes**

Daily Build, 2

7. Build private release
   - **Overnight, new private release with latest changes from others**
   - **Build for multiple platforms**
8. Test private release
   - **Test the new feature**
   - **Morning after 4, 5, 6.**
9. Execute quick test
   - **“Smoke test” of overall functionality. 30 min.**
10. Check in
    - **Synch and merge, again (to get latest changes).**
    - **Back out if conflict**
11. Generate daily build
    - **Build Master generates a build.**
    - **Stable snapshot.**
    - **Compile.**
    - **Automated test.**

Synch-and-Stabilize Cycle, 3

- Stabilization phase
  - Internal testing
    - Within company: “Dogfood”
  - External betas
  - “Zero-bug” release
  - Release preparation
    - “Going gold” -- master media
    - Documentation

Scaling Up

- Parallel teams
  - **Frequent synchronizations**
    - **Possibly daily**
    - **Debugging**
- Always have a product that you can ship
  - **Including all versions**
- Common language
- Continuously test
- Metric data drives milestone completion

Why (I Think) This Works

- Architecture
- Corporate memory
- Customer
- Code