Benjamin Fenster
CS-205 (Damon)
21 December 2004

Productization

I. Documentation Group
   a. Not usually considered part of the development group
   b. Technical people with a user perspective
   c. Very involved in meetings, development process
   d. Internal
      i. Starts with comments
      ii. Usually written by the developer. Sometimes when releasing source (or object code), someone will rewrite for clarity
      iii. Want something for every function, something interesting about structure of data
      iv. Includes other documentation: technical documents
      v. System / code designs
      vi. Data file documentation (format)
      vii. Important: Update these when you're finished to reflect the reality. Makes a nice ramp down phase at the end of the project
      viii. Include rationale for "paths not taken" (or you'll just have the same arguments all over again)
   e. Customer Documentation
      i. Audience, Audience, Audience!
      ii. Don't write things they already know (their business model, for example)
      iii. Do write lots of stuff about what they don't know (including "obvious" product features – they're not obvious to the customer yet!)
   f. Deliverables
      i. "Read me First" – No more than one page! People won't read anything that's too long, so find the critical stuff the user needs to know.
      ii. Installation Guide. How to upgrade, how to configure the many, many options
      iii. User Manual, Operator Manual
      iv. System Guide
      v. Troubleshooting Guide (appendix, often)
      vi. Tutorial (online)
      vii. Programmer's Guide
   g. Other Responsibilities
      i. Online help, error messages (make them clear)
      ii. May provide training for customers

II. Release Engineering
   a. Write "code" (make, shell scripts)
   b. Responsible for creating the Golden Master (building it and deciding when it's done)
   c. Versions
      i. Be able to tell what version the customer is running ("About" or command line)
      ii. Customer Support needs to know to answer questions
      iii. Need to know not only base release, but what patches are installed too
      iv. For any given version, Release Engineering can produce the entire source code
   d. Writes Installers
      i. Usually have tool to *generate* the installer that's dominant for each platform
      ii. May be really simple; typically not
      iii. Are prerequisites in place? (e.g. JVM)
      iv. Copy needed files based on the prerequisites found
      v. Update database (registry, "desktop database", et cetera)
   e. Path Releases
      i. Rolling Patch: Assume they've got everything up to a point, then add one more. This limits the number of possible configurations to O(N)
      ii. Selective Patches: Install only the patches you want. Gives $O(2^N)$ combinations
      iii. Customers prefer Selective Patches since they know some patches may actually make things worse and some just aren't applicable

        iv. Testing
            1. For cumulative: Just test each version
            2. For selective, test all required patches; will miss *many* combinations

III. Customer Support
   a. Can charge for customer support, so can milk for revenue years after all development has stopped.
   b. How-To Questions
      i. Range from very simple to very complex
      ii. Simple ones are intolerably dull
      iii. Complex ones border on consulting
   c. Bug-Handling Questions
   d. Levels
      i. Front Line. Run away! Not knowledge based. Plug search terms into the computer database and read the answer back to the customer
      ii. Mid Level. Some technical knowledge required (not already in the computer, but still pretty basic). Figure it out; add to the computer database
      iii. Back Line. Former or would-be developers. Paid about the same as developers. Solve problems based on code – really complex problems
   e. Communication
      i. Phone, e-mail, web (online chat growing)
      ii. Lots of good search technologies come from Customer Support

IV. Maintenance
   a. Once you're done with a release, keep working
   b. Add features you wanted. Fix bugs.
   c. Also rewriting code if it's become fragile
   d. Forward Development: New features, recoding, major bug fixes
   e. Maintenance Group: Bug fixes (simple)
   f. Expensive!
      i. Code gets very fragile (you designed for change but you're never dead on)
      ii. Understanding of the code declines
   g. Need to improve documentation.
   h. Need to retain developers.
   i. Need good communication between maintenance and forward developers
   j. Next Iteration: Support feeds back into the requirements for the next generation