The UNIVERSITY of VERMONT

Benjamin Fenster
CS-205 (Damon)
20 December 2004

Requirements

I. Goals of Requirements
   a. Correctness – You're describing the right problem
   b. Completeness – Covers everything
   c. Clarity – May be open to misinterpretation
II. User Problem
   a. What is it? What can we do to solve it?
   b. Consider specific tasks the user needs to perform
   c. Don't think in terms of the *solution*, worry just about the tasks from the user's perspective
   d. Use Cases
      i. See [CS-205-2004-3-Lecture-03:07]
      ii. Describes a particular task the user will perform in detail (adds clarity)
      iii. Can be written like a script (i.e. for a play)
         1. User: Do this
         2. App: When …
      iv. Many formats for these
      v. Really they're there so both the customer and developers clearly understand it
      vi. Never discuss design or solution, just tasks
      vii. Can reference other tasks that are defined separately
   e. Where to get input?
      i. Other products, or the current manual solution
      ii. See how these do it. Read marketing literature, documentation, interview customers, try it yourself (if affordable)
      iii. Write use cases for how *they* do it
      iv. Requirements (money, et cetera). How well does it do? Be *honest!*
      v. Need product focus – one or two simple, good answers for why people should buy your product (need to know where you have the advantage)
      vi. Keep that focus!
      vii. Also know how your product is worse? For what users? By how much?
III. Functional Requirements – Features
IV. Non-Functional Requirements
   a. These are the "ilities." Portability, scalability, et cetera
   b. Platform – OS, memory requirement, speed, disk space, network bandwidth, special devices (required and/or supported)
   c. Data – What does it produce / consume? Format? Required accuracy / precision?
   d. Performance (of the product itself)
      i. Best Case
      ii. Worst Case – What we'd look at for real-time
      iii. Mean – What we'd use for batch processing
      iv. Median
      v. Best N% – Like "90% of the time, …"
      vi. Is it response time or throughput that matters?
   e. Security
      i. Keeping information private and / or correct, keeping system up.
      ii. Is this a high visibility or high value target for attackers?
      iii. There's real cost to supporting security
         1. Explicit (backups, for example)
         2. Implicit (slower throughput due to encryption, inconvenience for customers / employees)
   f. Scale – How much data? How many processors? How many users?
   g. Internationalization – Not just language, but cultural considerations (colors mean different things to different cultures), legal, et cetera
   h. Environmental – Home, office, on the subway, on a factory floor, outside?
   i. Reliability

i. What is acceptable downtime ("multiple 9s")
ii. Acceptable mean time between failures
iii. What constitutes a stop-ship bug? Many minor bugs?

j. Users
i. All similar (e.g. all doctors) or more like a mall kiosk crowd?
ii. How much raining will they get?
iii. How frequently will they use the software? If infrequently, will forget procedures

k. Likely Changes – Have some guesses about where we're going for future releases. Don't go massively out of your way to support future features.

l. Date – How much flexibility? If launching for Pluto, *must* launch within the window.

m. Must be Verifiable
i. Need a way to determine whether you've actually met the requirements
ii. Don't say "it's fast" – say, "It processes 1000 transactions per minute"
iii. Lets you constrain individual pieces of the design before you even start coding

V. Collecting Requirements
a. Competitive Analysis
b. Sometimes don't have a customer to ask: need surrogate
i. Focus groups
ii. Often someone in marketing
iii. Product reviews (playing the customer in the text of the review)

VI. Presenting Requirements
a. Share with members of the group, with customer, and communicate over time
b. Clear, correct, complete
c. Format
i. Narrative
1. Easy to read / write
2. Tends to be hard to validate
3. No guidance for whether it's complete or not
4. Tends to be ambiguous
a. "Shoes must be worn", "Dogs must be carried"
b. May seem completely obvious when you write it, but still leaves room for misinterpretation later (or presently by someone else)

ii. Structured Text
1. Has defined categories (maybe a list of non-functional categories, pre and post conditions, or some other categories)
2. Harder to be incomplete
3. Can get tedious to read since "fill in the blank" answers can be formulaic

iii. Visual
1. Great for certain situations
2. Particularly obvious choice for graphical applications
3. In other situation, graphs (et cetera) are great
4. In still others, you may be oversimplifying the thing in order to draw the picture. Make sure all the details are still available in the text.

iv. Semi-Formal
1. Adds some semantics to structured text
2. Tables are a common implementation of this
3. Decision table (states, actions, new states – much like Finite Automata. See [CS-243-2004-3-NOTES-01]
4. Means you can also draw a state machine (state chart) with pretty bubbles and arrows. A visual representation!
5. Doesn't describe *actions* (necessarily) very well. Just shows states
6. End up using natural language to provide the details

v. Formal Notation
1. Mathematically precise
2. Can describe the exact conditions that cause a transition and the exact actions when the transition occurs

3. Advantages
    a. Precise!
    b. May even check automatically
    c. Much easier to write code from the requirements
4. Disadvantages
    a. Harder to read and write
    b. Almost nobody can actually read it
5. Put in a lot of effort go get this precise. Make sure you're not creating false / artificial precision.

VII. When Are the Requirements Done?

a. Depends on corporate culture; nature of the project
b. When the customer asks a question about a whole category you haven't addressed, better put some more time into the requirements
c. Changes
    i. How much change will you allow after the initial version?
    ii. Could disallow changes completely
        1. Makes the customer put in effort up front
        2. You'll probably do everything "wrong"
    iii. Can allow unlimited changes, but you end up wasting a lot of effort
    iv. Need a balance
d. Tracking Requirements
    i. Establish baseline at the beginning
    ii. For every change, track
        1. What, When, Who Requested, Who Agreed
        2. Want engineering, marketing, customer (if customer would care)
    iii. Document rationale for change. Once you agree to a new requirement make sure you don't forget *why* you introduced it (and end up having the same argument again later)
e. Track Assumptions
    i. We want everything in Spanish *because* we *assume* we're selling to Mexico.
    ii. Tie requirements to relevant assumptions
    iii. That way if an assumption changes (as when the new Arianne rocket had a new maximum velocity) you'll know which requirements are affected, and thus what parts of the code need to change.