Lessons from Assignment 1

Dr. James A. Bednar

jbednar@inf.ed.ac.uk

http://homepages.inf.ed.ac.uk/jbednar

Users do not write perfect "user stories" by default

- Drawing the use case diagrams requires understanding what the user wants to do.
- The original wish list was not a list of use cases! It has to be substantially reorganized, grouping similar items together and splitting others, to form a use case model.
- Many of the use cases that should be part of the model were not ever mentioned explicitly by the user, but can be inferred.
- Developing a use case model is not the same as just drawing pictures from the user's wish list.

User stories may be implicit

- Some of the wish list items are very clearly user stories.
- Many others consist of statements that e.g. the program should "have" something.
- Even though apparently expressed like a non-functional requirement, the only way the user would care that a program "has" such an item is if he or she can access it in some way.
- Thus such statements should be interpreted as sets of implied user stories (displaying, printing, etc.) that can be offered by a program having that item.
- Users need to be coaxed into providing user stories – it's an iterative process.

Danger: Customer techno-speak

- Be very afraid when a non-technical user uses words that have a specific technical meaning in CS or SE, e.g. "database" or "model".
- User very often is using the words non-technically, i.e., literally.
- E.g. the ParentMagic user thinks of a database only in user terms: a place where data is stored that she can access later. She has no idea about all the CS implications of that term, just how she can use e.g. imdb.com.

The customer is always right

- Even though the customer has no idea what is feasible, or how much work it would be to implement, there is always a reason for a request, even if not expressed well.
- E.g. never tell the customer that he or she was wrong to want a certain feature; just say why implementing it would be difficult, time-consuming, or would preclude other more important features.
- User can then decide what is really crucial to keep.

Dealing with impractical requests

- Users often want apparently unreasonable things, like having the program learn from them and “act reasonably.”
- Need to explain why that very reasonable goal is usually infeasible, because e.g. the current state of technology is not good enough to provide anything but annoyance.
- Features like unbidden popups may sound good, but would probably be very annoying – should gently explain this at first, and anticipate many iterations of tweaking later to make it useful (big argument in favor of XP approach!).

Data from 30 books

- Typing and scanning in the data from 30 books would take months or years, not 3 days as some people stated!
- Before even starting such a project, would need to obtain a license to use the data. That’s a huge barrier to shipping these features in 30 days.
- Even though finding the books and arranging the licenses is not programming, it’s still a huge component of the project.
- For this particular application, would probably need to hire at least one expert (doctor, etc.) to validate your information, and may even need a lawyer.

Should this code be written at all?

- Most of the things proposed in this project have been done dozens or thousands of times before, just not specifically for parents.
- Estimated time and effort depends crucially on how much reuse was proposed.
- Should acknowledge how ParentMagic could interface with external programs like contact managers, photo album software, DBMSs, etc.
- Actual functionality implemented specifically for ParentMagic should be small.

Taking the user's perspective

- The email to the customer was mostly an exercise in taking the user's perspective.
- Users, especially of consumer-oriented software such as this, require a vastly different approach language than do other developers.
- User wants to know what the software will do for them, not the "functionality" it will provide.
- The user wish list specifically labeled some features as important – it was absolutely crucial to address those, whether implementing them or not.

Most common email errors

- Omitting salutation, conclusion, paragraph breaks — emails are nearly unreadable without those
- Rudeness — why SW developers are normally kept away from the customers
- Too much info – customers get overwhelmed easily
- Jargon – must be in customer's language, not developer's

Bad example: Rude

Dear User,

We have reviewed your requirements list for Parent Magic. This is a huge project, and many of these requirements are not practical or or even possible for us to address. Here we will focus on three aspects of the software that can at least partially be achieved.

The first feature is a pop-up box that will appear from time to time on your screen and ask whether there's anything new to report about your baby. It will be very simple, since you did not specify what types of things you would want to record. . . .

We will be asking for your input at different stages of the project. If you have any questions or concerns, you are welcome to complain.

Sincerely, The Programmer

Bad example: Jargon

Dear User,

Having examined the user requirements document for Parent Magic, we are very excited to be working on this exciting new project, and feedback from end users will be invaluable during each iteration and release cycle! With that in mind, please provide feedback on the following summary of our design documents for the first end-user deliverable of Parent Magic.

Since our goal is to ship a beta release within one month, we have chosen to focus initially on three core functionalities. We're starting with features which seem highest priority, yet can rapidly be prototyped and deployed. Other features will be added, and existing features will be modified, according to user feedback, with each successive release. . . .

Summary

- It takes work to make user wishes coherent and understandable.
- Technical language from customers is likely to be used differently than you expect.
- Taking the user's perspective is difficult, but crucial for writing successful code.
- Treat your customers with respect; they know the domain better than you do.
- Always consider whether the code should be written at all, before you start!