Specifying Application System Requirements

Phase 3 in our sample SDLC
The critical point in the life cycle Why?

COMP 477/377
Spring 2016
Mr. Weisert

External System Design (ESD)

- Alternative names:
  - Functional specifications
  - Detailed system requirements

- Purpose:
  - To specify everything about what the proposed new or modified application will do.
  - Very little, if anything, about how (keeping our options open).

Author(s) of the ESD

- One or more systems analysts based on:
  - Interviews with the sponsor (client) management and with actual users
  - Documentation of current practice
  - Observation of current practice
  - Discussion of future desires

Why can't the users (client) write it themselves?

What would you do with this business rule?

- "Customers who place more than $10,000 business per year and have a good payment history or have been with us for more than 20 years are to receive priority treatment."

  - (from a popular systems analysis textbook)
Ambiguous scopes (which was meant?)

a. "Customers who (place more than $10,000 business per year and have a good payment history) or have been with us for more than 20 years are to receive priority treatment."

b. "Customers who place more than $10,000 business per year and (have a good payment history or have been with us for more than 20 years) are to receive priority treatment."

Ambiguous language
What does it mean to say:

- place more than $10,000 business per year?
- have been with us for more than 20 years?
- have a good payment history?

Audiences for the ESD

1. **User representatives**, in order to confirm the systems analysts' understanding of their needs
2. **Programmers** who will implement the system, in order to confirm:
   - that they understand it
   - that it's technically feasible
   - estimates of development cost and operational cost

The critical criterion

- It is essential that both audiences understand the ESD in exactly the same way in full detail
- The ESD is then like a **contract** between the developers and the users.
  - The developers will have fulfilled their commitment if the delivered system conforms to the specifications.

  *Does that mean the specifications are frozen?*
An idea

- **Problem:**
  - User representatives don't understand technical jargon, and programmers don't understand business concepts and terminology

- **Proposed solution:**
  - The systems analyst (who understands both) should prepare two versions of the ESD
    - One in business-oriented language for the sponsoring end users
    - The other in technical language for the developers

  Is that a good idea? Why or why not?

An idea

- **Problem:**
  - User representatives don't understand technical jargon, and programmers don't understand business concepts and terminology

- **Proposed solution:**
  - The systems analyst (who understands both) should prepare two versions of the ESD
    - One in business-oriented language for the sponsoring end users
    - The other in technical language for the developers

  When would be discover inconsistencies between the two versions?

No, no, no!

- That's a terrible idea! (although many organizations and some experts recommend it)

- Any discrepancy between the two versions will go undetected until the late stages of testing (phase 6 in our sample life cycle)

Conclusion

- We need a way of documenting detailed requirements that is **fully understandable** to both users and developers
  - Does such a thing exist?
  - Will both audiences really read it carefully?
Ways of documenting requirements

- Overview plus:
  - ~1957: Flowcharts and record layouts
  - ~1967: "Victorian novel"
  - ~1977: Structured analysis
  - ~1987: Object-oriented analysis
  - ~1997: UML with use-cases
  - ~2007: Discrete requirements list

- Some in "agile community" advise: *Don't bother!*
  - "Incremental approach"
  - "Emerging specification"

Which way of capturing detailed requirements is best and how do we do it?

- We won't answer that in this course
  - It would take too long
  - It provokes controversy

- Loyola offers courses (COMP 320, ISOM 496) in systems analysis

- But keep in mind the critical criterion!
  - *What was that?*

Conclusion (again)

- We need a way of documenting detailed requirements that is fully understandable to both users and developers

But we can dispose of one of them

- Overview plus:
  - ~1957: Flowcharts and record layouts
  - ~1967: "Victorian novel"
  - ~1977: Structured analysis
  - ~1987: Object-oriented analysis
  - ~1997: UML with use-cases
  - ~2007: Discrete requirements list

- Some in "agile community" advise: *Don't bother!*
  - "Incremental approach"
  - "Emerging specification"
### The "incremental approach"

(1959 and enjoying revival today)

1. A senior programmer confers with the user-sponsor to get a rough idea of the problem.
2. Programmer goes off and writes a little code.
3. Programmer shows output to the user.
4. User tells programmer what's not right and what else is needed.
5. Programmer modifies and augments code accordingly.
6. Back to step 3 until the user is satisfied!

*What's wrong with that?*

### Using I-O prototyping to help elicit requirements

- No matter which method we use to document requirements, it may still be hard to get users to state them completely and unambiguously.
- With some kinds of user it may help to show them and actual prototype of:
  - a report or other system output
  - a GUI form for entering a transaction or for navigating among various functions
- Prototype code just shows what something will *look like*. It needn't actually *do* anything.

*What's the danger there?*

### A warning about prototyping

- If an I-O prototype looks convincingly realistic, a naive user may infer that the system is nearly ready.
- Users may expect the final system to look exactly like the prototype that they approved.
- Make it clear that the prototype is (at least potentially) *throw-away* code.

### The make-or-buy decision

- In the 1960s nearly every need for application software was met by custom development
  - by in-house staff, or
  - by a contract developer
- Since then, software packages have been available for many applications that are common to many organizations, such as:
  - accounting systems
  - inventory control and forecasting
  - sales
  - manufacturing control
  - etc., etc.
So when would we now do custom development?

- Application is something really new.
- We can do it better than our competition and gain advantage.
- We're developing a software product to sell.
- We can't find a suitable package
  - A common policy in user organizations:
    - "We will develop a custom application only after it has been shown that no suitable program product exists."

When should we make the decision?

A common (but not recommended) approach

- Our company has chronic issues with manufacturing inventory control:
  - Out-of-stock parts hold up the assembly line.
  - Too much money is tied up in surplus parts.
- Make a list of software vendors who offer a manufacturing inventory control software package
  - Invite each of them to come in to make presentation
- Identify the 2 or 3 most impressive products and try to negotiate contracts with their vendors.
- Choose the winning product and charter a project to install it.

Problem with "install the product" projects

- When do we discover that the product lacks some capability that's absolutely essential to our manufacturing process or to our marketing strategy?
- How could we have found out sooner?

The rational make-or-buy decision

- Is based upon the ESD
- That is, we need to know our requirements before we can evaluate how well a product meets them.
- Note that the incremental approach rules out buying an application software product.