Usability Engineering Process

Administration

- Questions about Assignment #0
- Questions about Assignment #1

- Well done to those who already submitted assignments. Don’t wait for the deadline!

Usability needs to be a process

- **"Usability is not a quality that can be spread out to cover a poor design like a thick layer of peanut butter." [Nielsen]**
- **Like Software Engineering, it is a *process* for developing software that helps insure high quality**
- **Must plan for and support usability considerations *throughout* development**

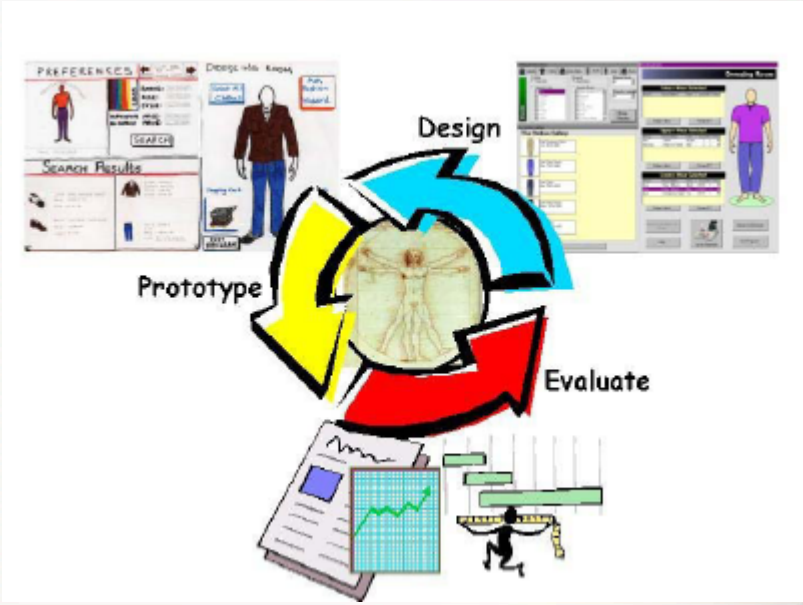
"Usability Engineering"

- Parallel to "software engineering"
- Make the application of usability methods
  more like engineering
  - Measurable
  - Process oriented
  - Not just "art"

Development process

- Software engineering has developed quite a bit of process for software development
- Bad news is that a lot of it does not work well for UI software
  - Traditional SE approaches are a flaming disaster
  - But need to understand the vocabulary and mindset

vs.



Traditional SE process

The "waterfall" model

- Not typically advocated anymore, but terminology and biases remain

Requirements specification

Design

Coding

Integration and testing

Operation and maintenance

Waterfall model

- Called "waterfall" model because when you finish one phase you are not supposed to go back "up stream"

Requirements specification

Design

Coding

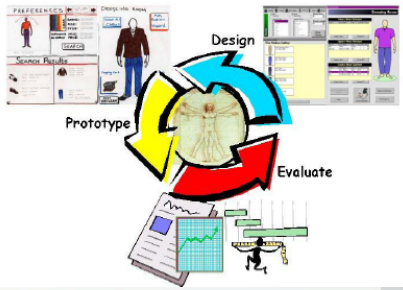
Integration and testing

Operation and maintenance

Waterfall model

- Implies that you design once (and get it right)
  - Not really possible for UI software
- Obsolete, but many of the parts remain in almost any SW process
  - Biases from this traditional approach remain
  - Also beware that terminology like "testing" doesn't necessarily match what we typically mean in HCI

Steps in an iterative process (not fully in sequence)



1. Study the user and their tasks
2. Study the competition
3. Set usability goals
4. Participatory design
5. Coordination of the total interface for consistency
   - Including documentation, help, etc.
6. Guidelines and heuristic evaluation
7. Make prototypes of the system early and quickly
   - Actually faster to prototype first
8. Empirical testing
9. Iterative design
10. Collect feedback from field use

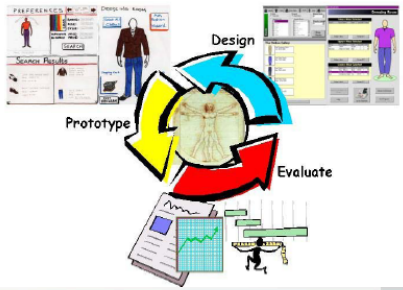
Requirement specification

- What will the system do?
  - What does it need to do meet the customer's (user's) needs?
  - What does the customer (user) want?
- Encode in a spec that drives the next phases

Requirement specification

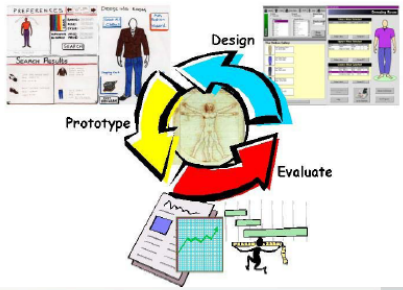
- Requirement specification quite hard in practice
  - Users can't tell you what they need, etc.
- Writing down a requirements spec is not very realistic for UI design
  - But, doesn't mean you shouldn't find out about user needs

1. Know the user



- **Study the intended user and the use of the product**
  - **Best if developers go and interview them personally (nothing like personal understanding → intuition)**
  - **But can be difficult because**
    - **May want to hide the developers**
    - **Developers may not have skills for dealing with users**
    - **Reluctance of sales people, etc.**
    - **Reluctance of users**
- **User Characteristics**
  - **Work experience, education level, age, previous computer experience, …**
  - **Time for learning, training**
  - **Available hardware (monitor size, acceptance of plug-ins, platforms**
  - **Social context of use**

Early focus on users & tasks

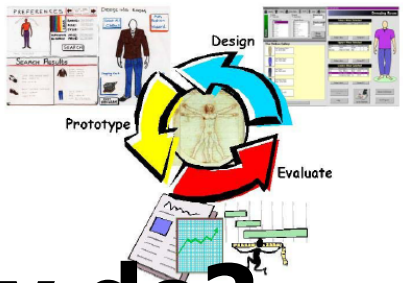


From Gould & Lewis article

- Note: this is quite old at this point, but still relevant

- Not just "identifying", "describing", "stereotyping" users
  - *Direct contact* through interviews & discussions
  - We teach contextual inquiry as method for this

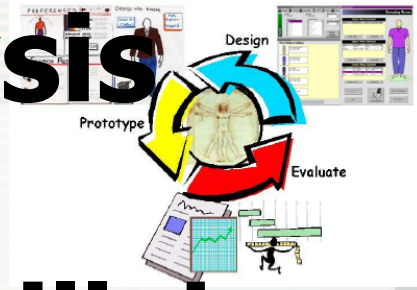
Task analysis



What tasks will the users really do?

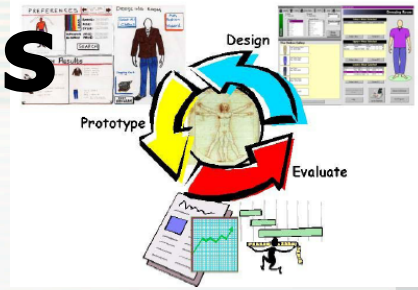
- Extremely important
- Involve the users in this
- Important to include exceptions and error conditions
- Many different kinds and variations on task analysis
  - Nielsen's
  - "Hierarchical task analysis"
  - Contextual inquiry
  - Choose method based on setting & goals

User-centered task analysis



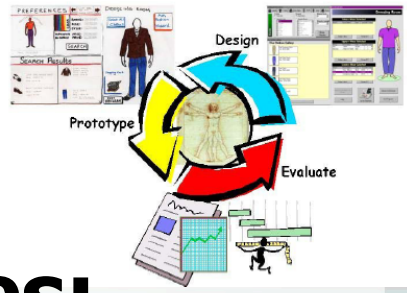
- **Based on what the *user* will do**
  - ***Not* what the system will do**
- **High-level**
- **What, not how**
  - **Nothing about how to accomplish at user level (no discussion of web pages, buttons, filling in fields,…)**

Components of task analysis



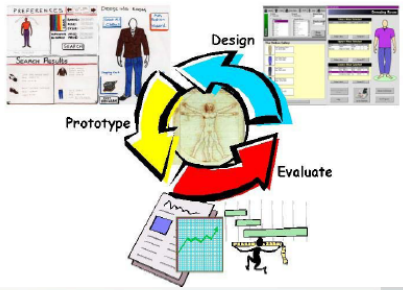
- Goals
  - What is this task supposed to accomplish (again: what, not how)
- Information needs
  - What does the user need to know or view to do this task?
  - Including what needs to be visible to the user (on the screen)
    - What the system needs to show, and
    - What the user needs to know

Task analysis: scenarios



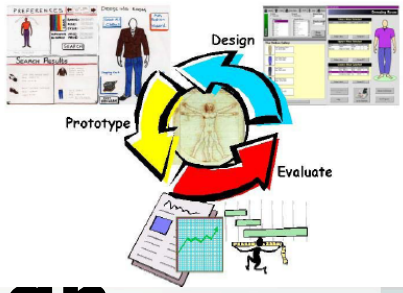
- Scenarios (stories) of typical uses:
  - Related to software engineering "use cases"
  - Specific example of how user might use system
  - One scenario for each major class of user doing each kind of important task
    - Tasks you want to make efficient & easy
    - What is important to optimize?
  - Will significantly affect design
  - Try to include lots of exceptional cases
  - Shows how interface will be used

Uses for task analysis



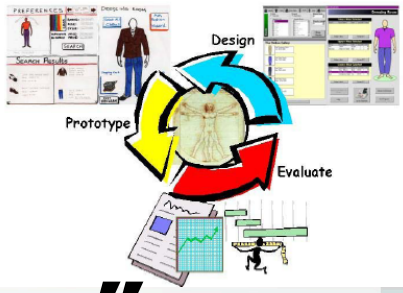
- Drive refinement of interface
- Communication of concepts
  - To management, marketing, rest of dev. team, and customers
- Can potentially replace much textual specification

Results of task analysis



- Scenarios to be used during design
- List of thing users want to accomplish (goals)
- Information they will need to accomplish those goals
- Communication needs of users with other people
- Steps to be performed and interdependencies
- Criteria to determine quality of results

2. Competitive analysis

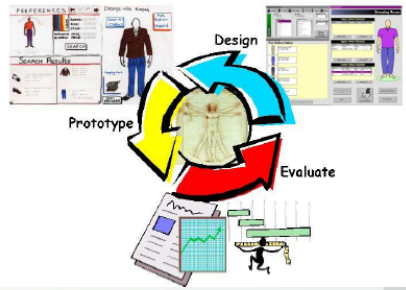


Goal: "Know the competition"

- **For usability and function**
- **Read trade-press reviews**
- **Visit competitor's web sites**

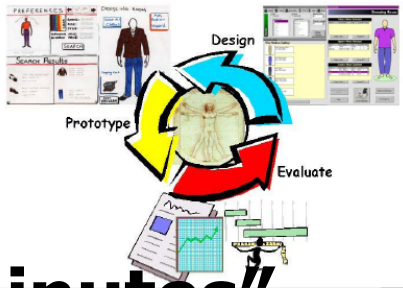
➢ **Determine importance of various features and issues**

3. Setting Usability Goals



- What does it mean to be "easy to use"?
- Some possible definitions:
  - "I like it"
  - "I always do it that way"
  - "That is the way system X does it"
  - "It's easy to implement"

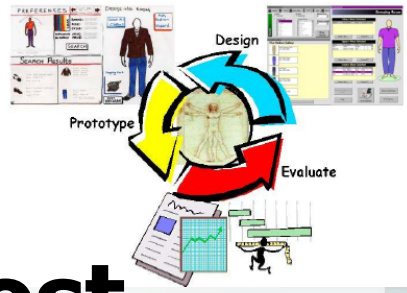
Much better goals:



- "Can be learned in less than two minutes"
- "User will perform N error-free tasks per session"
- "Error rate will be lower than 2 per 10 operations"
- "Tasks will be performed in 30% of the time it takes using competitor's system"
- "Users will have a high satisfaction with system (as measured by a survey)"

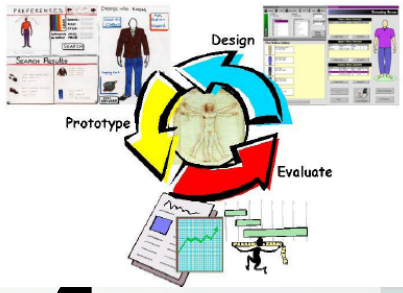
- ***Explicit, specific, measurable metrics***
- **Allows *objective* decision making**

Goals (cont.)



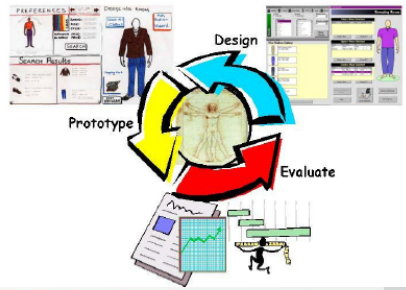
- Tradeoffs, so have to pick most relevant metrics
- Some measures:
  - Learnability: time to learn how to do specific tasks (at specific proficiency)
  - Efficiency: (for expert) time to execute benchmark tasks.
  - Errors: rate per task, time spent on errors, error severity
  - Subjective satisfaction: typically via questionnaire

Goal levels



- **Multiple levels to consider for your system**
  - **Minimum acceptable level**
  - **Desired (planned) level**
  - **Theoretical best level**
  - **Also take note of current level and/or competitor's level**

Financial impact analysis



- Prove it
- Demonstrate the importance of usability

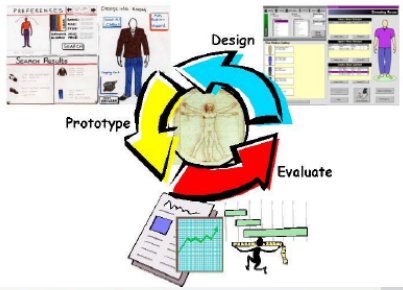
- # users * salary per hour * # hours on system = cost of system
- Use to estimate savings (also reduced training, error time, need for support staff, etc.)

➢ Tells how much time to spend on usability

Note: whole book on this subject:

- Randolph Bias and Deborah Mayhew, "Cost-Justifying Usability", Academic Press, Boston, 1994.

4. Participatory design



- **Users involved *during* the design process through regular meetings**
  - **Not just at the beginning e.g., during contextual inquiry**
  - **Users are good at reacting to concrete designs and prototypes**
  - **But users are *not* necessarily good designers**

Design (from traditional SE process)

- Several types
  - Architectural design
    - High level decomposition
    - What are the big pieces, how do they fit together to make system
  - Detailed design
    - The littler boxes that go in the big boxes

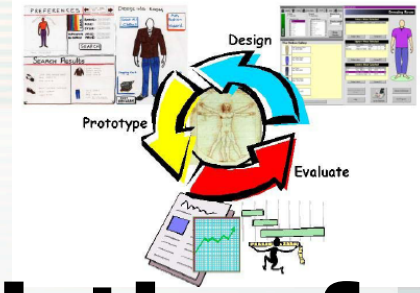
Design (from traditional SE process)

- UI design would be detailed design + requirements (but iterated)
- But UI design doesn't fit well
  - Traditional SE design is mostly about the system structure
  - UI design is mostly about what the user sees
    - *Often without regard to system structure that makes it happen*

Coding and unit testing

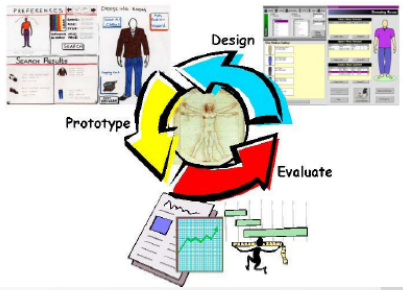
- Actually write the code
  - Typically the only part you get graded on in university
  - Only part you can't skimp on
- Test small scale parts (units) to ensure they function right
  - Extremely important (and under appreciated) in practice

5. Consistency



- Most important characteristic of UI
- Requires oversight
  - Not each dept./developer creating own section
- May require overall design document, vocabulary guide, style guide, templates, etc.

6. Use of guidelines and Heuristic Analysis

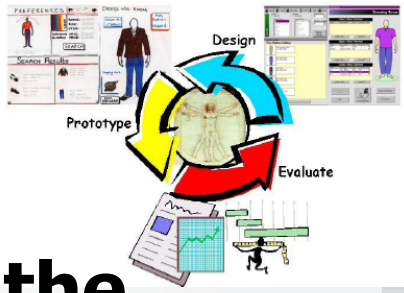


Designers evaluating the UI

- **Based on their experience**

(Future homework on this topic)

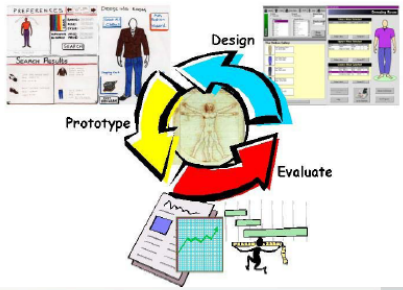
7. Build prototypes



- Simulation of (important aspects of) the interface
- Quick and cheap to create (e.g., no "back end")
- Can be "low fidelity"
  - E.g., Paper prototypes
    - Can be roughly drawn
    - Actually better if not refined
    - Focus on addressing important questions about tasks (not surface issues, e.g., colors, exact layout, icon design, etc.)
  - Can use in studies
    - Experimenter can "play the computer"
    - Useful and revealing

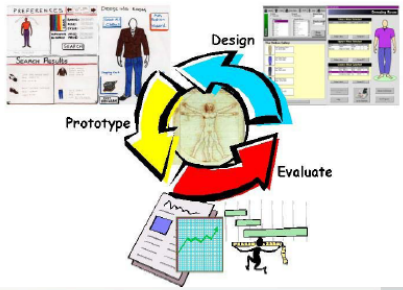
Next lecture considers this topic in detail

8. Empirical testing



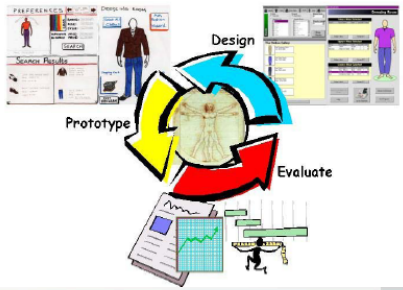
- Critical to usable products
- Designers must watch users
  - Not just e.g., web logs
- Not necessarily difficult or expensive
  - Often a few user tests get you most information
    (get the "high order bits" quickly)
  - Don't necessarily need a fancy lab

9. Iterative design



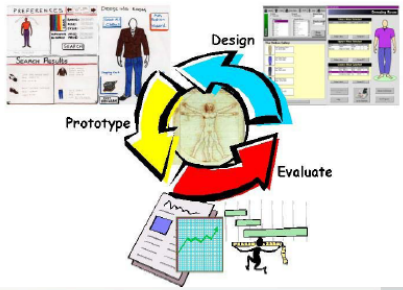
- Redesign based on evaluation
- Note: New design may be worse or break something
  - be prepared for that, often need to "back out" of recent change
- Keep track of reasons for design decisions
  - "Design rationale"
  - So you don't need to keep revisiting the same decisions
  - When future conditions suggest changing a decision, use this to remember why you made it that way and what implications of change are

9. Iterative design



- **Instead of arguing about a design feature, figure out what information would tell you which way to go**

Iterative design



From Gould & Lewis article

- Empirical testing with intention to fix the problems
- Not just *goals* ("be easy to use") but a process to achieve the goals

Integration and testing (in traditional SE process)

Typically don't build things in university big enough to hit this

- Testing that when you put the pieces together, they work
  - Even if "units" work perfectly, whole may not

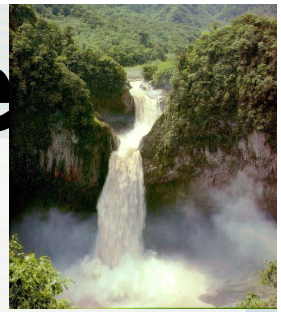
Types of testing

- System testing
  - Do you think it works
- Verification
  - Does it match the spec
- Validation & acceptance testing
  - Does it work to the customer
  - Does it meet the contract / spec

Testing

- Notice that all that testing is about testing the system
  - "User tests" are not really there
  - This testing typically aimed at uncovering mistakes in implementation
  - When you user test you find out the requirements and/or design were wrong

Operation and maintenance

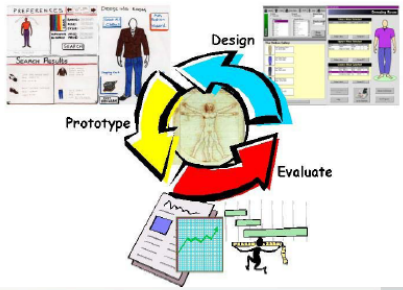


- What happens after it's delivered
  - The next release
  - Bug fixes
  - New features

Waterfall model does not work for UI software

- UI requirements and design are very hard
  - Too hard to get right on the first try
  - Human beings are just too complex
    - Just don't know enough to do it from first principles
  - Hidden aspects contributing to mental models
- Must iterate the design

10. Measure real use



- Follow-up after release
  - For the next version
- From bug reports, trainers, initial experiences, …
- From web logs, reports, customer support, …

User-centered iterative approach has been around for a long time

- Catching on, but practices still don't get followed as much as they should
  - Increasing, but not there yet
- Why?

Obstacles to user-centered iterative design

- Big reason: Impractical
  - Iteration is expensive
  - Can barely afford to build it once
    - Even with high levels of resources
  - Dealing with this is one of the things this class is about
  - ➢ Good prototyping practice helps a lot

Obstacles to user-centered iterative design

- Competing approaches
  - The power of reason and "getting it right the first time"
  - CS typically teaches that you can (and should!) get your design right the first time

Obstacles to user-centered iterative design

- Value of interaction with users is misestimated
  - User diversity is underestimated
    - "I understand the users"
  - User diversity is overestimated
    - "I'll never understand them all"
    - "Can't get statistically sound info"
  - Believe that users don't know what they want (true, but…)

Obstacles to user-centered iterative design

- Difficult to manage, measure, and set goals
  - When will the UI software be done?
    - Very hard to estimate software development times anyway
    - Open-ended iteration makes it even harder

Chicken and egg problem

- Can't afford to build it more than once
- Can't get it right the first time
  - Must test and redesign, but can't do that without building
- How do we get past this?

Chicken and egg problem

- How do we get past this?
- Build something less than the full system and iterate on that

→ Prototyping… next lecture

Warnings about iterative design

- Big picture first
  - It's easy to get bogged down in details and miss the forest for the trees
    - E.g., layout, color, etc.
  - Get the "high order bits" first
    - "Is this the right functionality to support the user's tasks?"
    - "Is this conceptual model going to work for the user?"

Warnings about iterative design

- Beware of delivering (what was supposed to be) the prototype
  - A lot of pressure to deliver the first thing that looks like it works
    - Can get you in big trouble later
    - Need to make sure everyone knows this is a prototype
  - May look a lot closer to done than it is
    - Often want to make things look "sketchy" early on to avoid this

Warnings about iterative design

Design inertia

- First designs have a huge impact
  - Evolutionary process & like biological evolution can be hard to back out of decisions
- Need to be willing to make radical changes when maturity of design is low
  - Needs to be low cost early to allow this
  - Explicitly consider several designs

Warnings about iterative design

- Need to understand the reasons behind usability problems
  - When "feature X" causes usability problems the simple thing is to eliminate X
    - But if we don't understand it, we may make same mistake again and/or make things worse

Questions?