# User-Driven Design



## Including users in design

- Identifying *actual* requirements
- Evaluating design
- Including users in development process
- User interface design



## Requirements analysis

- When someone asks you to build a system, what do they want?
  - I.e., what are their **requirements**?
- Requirements analysis is hard
  - If you get it wrong, all the rest of the development work is misguided
  - Users' stated requirements aren't necessarily their actual requirements
    - *Why?*



## Some possible answers…

- Users may not know what they really want
  - Until they have something to play around with
  - Then they discover what they really need
- Users may not know what's possible and what's not
  - "Self-censoring"



## How to elicit requirements

- Work with users to…
  - Educate you about their domain & needs
  - Educate them about what you can & can't do
- Produce mock-ups for them to play with
  - E.g. drawings of user interfaces
  - E.g. story-boards of sequences of actions & responses



## Use cases

- **Use cases** are one way to write down requirements
- They are *scenarios* of possible use, from the user's point of view
  - Users may be able to think through possible scenarios of use of a system, and tell them to you accurately
  - Helping them think through possible outcomes may clarify the requirements for them
- Must be converted into more specific requirements later, by designers

## Examples

- Use cases for a classroom
- Use cases for a web form for on-line application for admissions to CS dept.
- Use cases for a debugger tool



## Spiral development model

- Want to get evaluation of a design as quickly as possible
  - So try to go from **requirements** to **design** to **implementation** to **testing** to **evaluation** as quickly as possible
- Spiral model: do a little of each, iteratively
  - Produce a quick prototype first, with just a little functionality (a few use cases)
  - Get users to evaluate it
  - Refine requirements & design
  - Enhance prototype a little, repeat



## Aids for spiral development

- Higher-level languages help prototypes get built quickly
  - Rich standard libraries, domain-specific libraries
  - System handles mundane, time-consuming details, e.g. memory management, safety checking
- Performance-critical parts can be tuned and reimplemented, possibly in faster but less productive languages, once the design stabilizes
  - Modular design critical to making this practical
  - "Plan to throw one away"



## Participatory design

- It's good to continue to include users in all phases of development
  - Early feedback is always cheaper
  - Greater confidence that the system being produced will be liked by its users
- Popularized in Scandinavia
  - Socialist societies, with strong labor union influence