

# CPS 108

- **Object oriented programming and design, we'll use Java and C++**
  - *Language independent* concepts including design patterns, e.g., command, iterator, factory, strategy, ...
  - *Design independent* concepts, e.g., coupling, cohesion, testing, refactoring, profiling, ...
- **Non OO programming and design, we'll use C++ (and its C-subset)**
  - From Java/ ArrayList to C++/ vector to C/ int \*
  - From classes to functions, from references to pointers

# Goals for students in CompSci 108

- **Adept at solving problems requiring programming**
  - Design, test, implement, release, revise, maintain
- **Reasonably facile with basic Java idioms/libraries**
  - How to read the API, knowing what to ignore
  - Basic language features, basic libraries
- **Basic knowledge of C++ (and C) programming**
  - Beyond the old CompSci 100
  - Java-style use of STL

# More goals for 108 students

- **Know design principles**
  - Some characterized as hueristics
  - Some as patterns
- **Experience working in teams**
  - How to accommodate team needs, balance against individual needs (and goals)
- **Comfort in working alone, how to get and use help**
  - Peers, UTAs, TA, prof, Internet, ...

# Administrivia

- **check website and bulletin board regularly**
  - `http://www.cs.duke.edu/courses/cps108/current/`
  - See links to bulletin board and other stuff
- **Grading (see web pages)**
  - team projects
  - recitation programs (solo projects)
  - readings and summaries
  - tests (primarily take-home)

# Administrivia (continued)

- **Evaluating team projects, role of TA, UTA, consultants**
  - **face-to-face evaluation, early feedback**
- **Compiling, tools, environments, Linux, Windows, Mac**
  - **g++ 3.3, 3.4, 4.0?,**
  - **Java 5 (requires 10.4 on Mac)**
  - **Eclipse in all environments**
  - **Command-line tools**

# Classes: Review/Overview

- **A class encapsulates state and behavior**
  - Behavior first when designing a class
  - Information hiding: who knows state/behavior?
  
- **State is private; some behavior is public**
  - Private/protected helper functions
  - A class is called an *object factory*, creates lots of instances

# How do classes and objects work?

- **Classes communicate and collaborate**
  - Parameters: use-a (send and receive)
  - Containment: has-a (aggregate of parts, responsible for)
  - Inheritance: is-a (extends and specializes)
- **Understanding inheritance and interfaces**
  - What is polymorphism?
  - When is polymorphism not appropriate?
  - What is an interface in Java, what about C++?

# Design Criteria

*Good design comes from experience, experience comes from bad design*

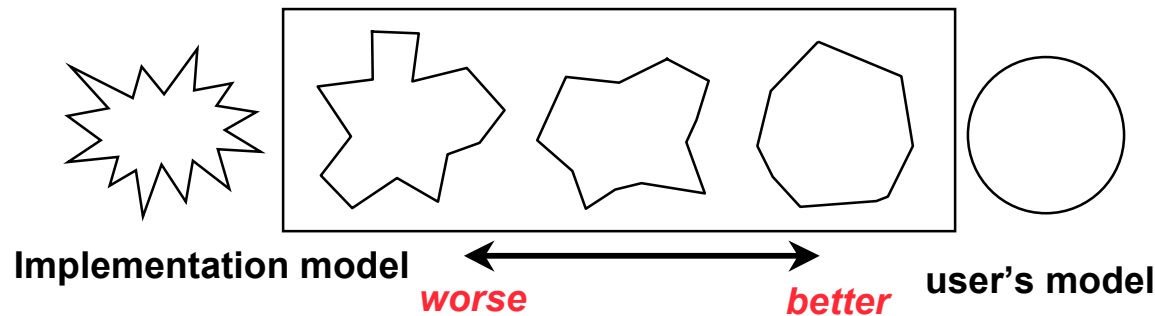
Fred Brooks

- **Design with goals:**
  - **ease of use**
  - **portability**
  - **ease of re-use**
  - **efficiency**
  - **first to market**
  - **?????**

# Software Design

See Alan Cooper, *The Essentials of User Interface Design*

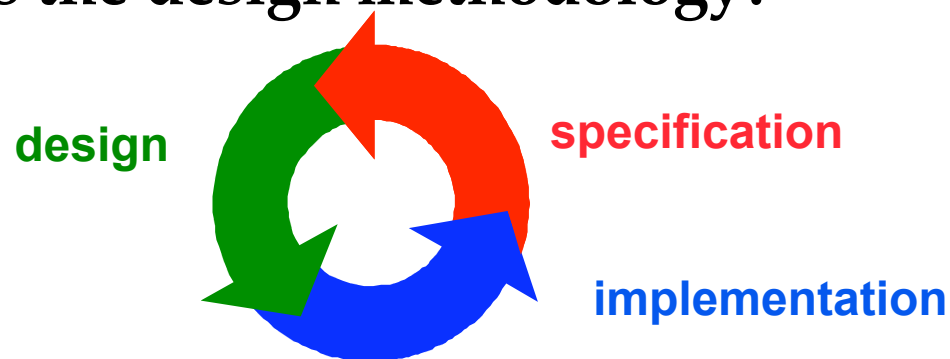
- who designs the software?



- Implementation is view of software developer, user's view is mental model, software *designer* has to bridge this gap
  - Example: copy/move files in a Windows/Mac environment, what's the difference in dragging a file/folder between two folders on the same device and dragging between devices, e.g., c: to a:? Is this a problem? To whom?

# How to code

- **Coding/Implementation goals:**
  - **Make it run**
  - **Make it right**
  - **Make it fast**
  - **Make it small**
- **spiral design (or RAD or !waterfall or ...)**
  - **what's the design methodology?**



# XP and Refactoring

*(See books by Kent Beck (XP) and Martin Fowler (refactoring))*

- **eXtreme Programming (XP) is an *agile* design process**
  - **Communication:** unit tests, pair programming, estimation
  - **Simplicity:** what is the simplest approach that works?
  - **Feedback:** system and clients; programs and stories
  - **Courage:** throw code away, dare to be great/different
- **Refactoring**
  - **Change internal structure without changing observable behavior**
  - **Don't worry (too much) about upfront design**
  - **Simplicity over flexibility (see XP)**

# Modules, design, coding, refactor, XP

- **Make it run, make it right, make it fast, make it small**
- **Do the simplest thing that can possibly work (XP)**
  - Design so that refactoring is possible
  - Don't lose sight of where you're going, keep change in mind, but not as the driving force [it will evolve]
- **Refactor: functionality doesn't change, code does**
  - Should mean that new tests aren't written, just re-run
  - Depends on modularity of code, testing in pieces
- **What's a module in Java?**
  - Could be a class, a file, a directory, a package, a jar file
  - We should, at least, use classes and packages