Spreadsheet: Model, View, Controller

- Model, View, Controller is MVC
  - Model stores and updates state of application
    - Example: calculator, what's the state of a GUI-calculator?
    - When model changes it notifies its views appropriately
      - Example: pressing a button on calculator, what happens?
  - The controller interprets commands, forwards them appropriately to model (usually not to view)
    - Example: code for calculator that reacts to button presses
    - Controller isn’t always a separate class, often part of GUI-based view in M/VC
  - MVC is a fundamental design pattern: solution to a problem at a general level, not specific code per se
    - GOF: Observer/Observable, related but no controller

How do Model/View communicate?

- Model has-a view (or more than one)
  - Can call view methods
  - Can pass itself or its fields/info to view
- View can call back on model passed (e.g., by model itself)
  - Model passes this, view accepts Model as parameter
  - Possible for controller/other class to pass model
- Controller contains both model and view (for example)
  - Constructs MV relationship
  - Possible for controller to be part of view (e.g., GUI)

Controller in MVC

- Loop until game over, where is code for board display?
  ```c++
  while (true) {
    getMove(m, player);
    if (ttt.makeMove(m)) {
      if (ttt.gameOver()) {
        break;
      }
      player = (player == 'X' ? '0' : 'X');
    } else {
      cout << "bad move " << m << endl;
    }
  }
  ```

GUI controller

- Typically no loop, GUI events drive the system
  - GUI events to event handlers (part of controller)
  - What about model/view game over coordination?
  ```c++
  connect(mouseClick, moveGenerator); // metacode
  void GUI::moveGenerator(MouseClick m)
  {
    controller->process(moveFromMouse(m));
  }
  void Controller::process(const TTTMove& m)
  {
    if (! myModel->makeMove(m)) {
      myView->showBadMove(m);
    }
  }
  ```
Designing classes in general

- **Highly cohesive**
  - Each class does one thing
  - Interface is minimally complete, avoid kitchen sink
    - What if client/user might want to hammer with an awl?

- **Loose coupling (and minimize coupling)**
  - Classes depend on each other minimally
  - Changes in one don’t engender changes in another
  - Subclasses are tightly coupled, aggregates are not
    - Prefer Has-a to Is-a

- **Test classes independently**
  - Unit testing means just that, and every class should have a unit test suite

Tell/ask and the Law of Demeter

- "Don’t talk to strangers"
  - Call methods in this class, parameters, fields, for created local variables, for values returned by class methods
  - No good, why? `fromPile.topCard().getSuit()`

> From David.E.Smyth@jpl.nasa.gov Mon May 26 17:33:30 1997
> From: "David E. Smyth" >To: lieber@ccs.neu.edu >Subject: Law of Demeter >
> I have been using LoD pervasively since about 1990, and it has taken firm hold in many areas of the Jet Propulsion Laboratory. Major systems which have used LoD extensively include the Telemetry Delivery System (a real-time database begun in 1960), the Flight System Testbed, and Mars Pathfinder flight software (both begun in 1993). We are going to use LoD as a foundational software engineering principle for the X2000 Europa orbiter mission. I also used it within a couple of commercial systems for Siemens in 91-93, including a Lotus Notes like system, and an email system.

More heuristics (some from Riel)

- Users depend on a class’s interface, but a class shouldn’t depend on its users

- Be suspicious of “God”-classes, e.g., Driver, Manager, System
  - Watch out for classes supporting method subsets

- Beware of classes with lots of get/set methods

- Support Model/View distinction
  - The model shouldn’t depend on the view, but should support multiple views

- If a class contains an object it should directly use the object by sending it messages

Working as part of a group

see McCarthy, *Dynamics of Software Development*

- establish a shared vision
  - what was/is Freecell? what can we add?
  - harmonious sense of purpose

- develop a creative environment
  - the more ideas the better, ideas are infectious
  - don’t flip the BOZO bit

- scout the future
  - what’s coming, what’s the next project
  - what new technologies will affect this project
Scheduling/Slipping

- McCarthy page 50, Group Psyche, TEAM=SOFTWARE
  - anything you need to know about a team can be discovered by examining the software and vice versa
  - leadership is interpersonal choreography
  - greatness results from ministrations to group psyche which is an “abstract average of individual psyches”
  - mediocrity results from neglect of group psyche
- Slipping a schedule has no moral dimension (pp 124-145)
  - no failure, no blame, inevitable consequence of complexity
  - don’t hide from problems
  - build from the slip, don’t destroy
  - hit the next milestone, even if redefined (“vegetate”)

Towards being a hacker

- See the hacker-faq (cps 108 web page)
  - Hackers solve problems and build things, and they believe in freedom and voluntary mutual help. To be accepted as a hacker, you have to behave as though you have this kind of attitude yourself. And to behave as though you have the attitude, you have to really believe the attitude.
- The world is full of fascinating problems
  - no one should have to solve the same problem twice
  - boredom and drudgery are evil
  - freedom is good
  - attitude is no substitute for competence

You may not work to get reputation, but the reputation is a real payment with consequences if you do the job well.

Aside: ethics of software

- What is intellectual property, why is it important?
  - what about FSF, GPL, copy-left, open source, ...
  - what about money
  - what about monopolies
- What does it mean to act ethically and responsibly?
  - What is the Unix philosophy? What about protection? What about copying? What about stealing? What about borrowing?
  - No harm, no foul? Is this a legitimate philosophy?
- The future belongs to software developers/entrepreneurs
  - what can we do to ensure the world’s a good place to be?

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?
- Implications in Freecell? What’s a Pile? A Deck?
Comfort: technology and mathematics

- "Show me all the first year students who live in Pegram and in Brown"
  - what does "and" mean here? Does the average user understand Boolean? Does the average programmer understand Boolean? Recursion? Threads? Queues?
- How you solve a problem in your program isn't necessarily how the user solves the problem, keep these distinctions clear
- "Saying that someone is 'computer literate' is really a euphemism meaning he has been indoctrinated and trained in the irrational and counter-intuitive way that file systems work, and once you have been properly subverted into thinking like a computer nerd, the obvious ridiculousness of the way the file system presents itself to the user doesn't seem so foolish."

From High to Low: genericity in C

- How do we sort by different criteria in C++ or in C
  - Templated function in C++, pass comparer
    - Can be a class or a function
    - Conventions required because of templates
    - Similar to inheritance, but differences?
  - Pass function pointer in C, what is prototype?
    - How do compare "generic" objects in C?
    - What is the type void * in C?
    - How do arrays work in C?
- Efficiency issues in code, see libsort.cpp

Pass by reference in C

```c
void fillArray(int * a, int size, int value)
{
    for(int k=0; k < size; k++) a[k] = value;
}
void setValue(int * a, int value)
{
    *a = value;
}
void createArray(int ** a, int size, int value)
{
    *a = new int[size];
    for(int k=0; k < size; k++) (*a)[k] = value;
}
```