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")

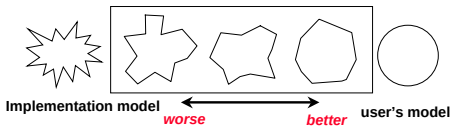
Aside: ethics of software

- 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."*

Applets and Applications

- Application run by user, double-clickable/command-line
  - No restrictions on access, reads files, URLS, ...
  - GUI applications typically include a JFrame
    - Has title, menus, closeable, resizeable
- Applet is downloaded via the web
  - Runs in browser, not trusted (but see policy later)
  - Can't read files on local machine (but see policy)
  - Can't be resized within browser
  - Uses jar file to get all classes at once
    - Alternative? Establish several connections to server

Developing Applets and Applications

- Create a JPanel with the guts of the GUI/logic
  - What will be in the content pane of both deployments
  - Makes GUI very simple, see code examples
  - Use JPanel in both Applet and Application
- Test with application first, easier to read files/resources
  - Migrate to Applet, test first with appletviewer
  - Migrate to web, may need to clear cache/reload

Packages, JAR files, deployment

http://java.sun.com/docs/books/tutorial/jar/basics/index.html

- Java packages correspond semantically to modules (related classes) and syntactically to a directory structure
  - Class names correspond to file names
  - Package names correspond to directories
  - Related classes belong together, easier to develop, easier to deploy
  - Leverage default/package access, use properties of protected which is subclass *and package* access

Packages, javac, java, javadoc

- In moderately big programs packages are essential
  - Can't easily live in a directory with 50 .java files
  - Can't easily co-exist in such a directory
  - Harder to use tools like Make and Ant
- Each of javac, java, javadoc is slightly different with packages, all must co-exist with CLASSPATH
  - File system vs. compiler vs. runtime
  - Source of confusion and problems
  - IDEs can manage Make/CLASSPATH issues

CLASSPATH and related concepts

- The default CLASSPATH is . current directory
  - Works fine with default/unnamed packages
  - Will not work with named packages
- Set CLASSPATH to directory in which packages live also include current dir
  - `setenv CLASSPATH "~ola:."`
  - `setenv CLASSPATH "`pwd`:."`
  - On windows machines change registry variable, separator is semi-colon rather than colon
- All problems are CLASSPATH problems

More package details

- To compile
  - Can cd into directory and type javac *.java
  - Can also type javac ooga/*.java from one level up
  - If CLASSPATH isn't set, the second won't work
- To run
  - `java ooga.TicTac` will work, you must specify the "real" name of the class being used.
  - Reading files requires full-paths or run from directory in which file lives
- To document
  - http://java.sun.com/j2se/javadoc/faq.html
  - Don't need to use –sourcepath, but can
  - `javadoc –d doc ooga ooga.timer ooga.game …`

javadoc for packages

- See the javadoc faq http://java.sun.com/j2se/javadoc/faq.html
  - For each package create a package.html file
    - Not in /** */ javadoc format, strictly html
    - First *sentence* after <body> is main description; a sentence ends with a period.
    - The package.html file should provide complete instructions on how to use the package. All programmer documentation should be accessible or part of this file, e.g., in the file or linked to the file
  - Use the `{@link foo.bar bar}` tag appropriately.
    - See the FAQ, or the source for the elan package online
- You may want to keep .java and .class files separate, see sourcepath and classpath as commandline args to java

From JITs to Deoptimization

- JITs compile bytecodes when first executed
  - If we can cache translated code we can avoid re-translating the same bytecode sequence
  - Spend time compiling things that aren't frequently executed (optimistic optimization?)
  - Errors indicate "compiled code" rather than line number
- Sun's HotSpot VM uses a different strategy for performance
  - Adaptive compilation: save time over JIT, compile "hotspots" rather than everything, uses less memory, starts program faster, http://java.sun.com/products/hotspot/
  - No method inlining, but uses *dynamic deoptimization*
    - Program loads new subclass, compiled code invalid, so ...?
- What does the class loader do?

Loading .class files

- The bytecode verifier "proves theorems" about the bytecodes being loaded into the JVM
  - These bytecodes may come from a non-Java source, e.g., compile Ada into bytecodes (why?)
- This verification is a *static* analysis of properties such as:
  - .class file format (including magic number 0xCAFEBABE)
  - Methods/instances used properly, parameters correct
  - Stack doesn't underflow/overflow
- Verification is done by the JVM, not changeable
  - Contrast ClassLoader, which is changeable, can modify classes before they're loaded into the JVM

`http://securingjava.com`
`http://java.sun.com/sfaq/verifier.html`

The ClassLoader

- The "boot strap" loader is built-in to the JVM
  - Sometimes called the "default" loader, but it's not extensible or customizable the way other loaders are
  - Loads classes from the platform on which the JVM runs (what are loader and JVM written in?)
- Applet class loader, RMI class loader, user loaders
  - Load .class files from URLs, from other areas of platform on which JVM runs
  - A class knows how it was loaded and new instances will use the same loader
- Why implement a custom loader?
  - Work at Duke with JOIE

Running an Applet

- An applet has an init() method
  - similar to constructor, called only once, when the applet is first loaded
- An applet has a start() method
  - called each time the applet becomes "active", run the first time, or revisited e.g., via the back button in a browser
- An applet has a stop() method
  - called when applet is invisible, e.g., user scrolls or goes to another web page
- other methods in an applet
  - destroy, getAppletInfo, getParameterInfo
- Applet subclasses Panel, so it is an Container/Component