Week 12
Software Engineering Tools

Paul Chew
CS 212 – Spring 2004

Announcements

- Part 4
  - Will be due on last day of classes (Friday, May 7)
- No Sections today (or next week)
- Paul Chew's Office Hour for today (W 4:30 – 5:30) is cancelled due to special computer graphics talk
  - Today's talk
    - 4:30pm, Call Auditorium
    - Marc Levoy (Stanford)
    - *The Digital Michelangelo Project*
  - Tomorrow's talk
    - 4:15pm, Call Auditorium
    - George Joblove (Sony Picture Imageworks) and Douglas Kay (Mondo Media)
    - *Digital Imagery in Entertainment*

Unix

- Original version by Ken Thompson (Bell Labs) in 1969
- An interactive, multi-user operating system (not the first such system, but an early one)
- Unix is closely tied to the development of C
  - Unix was originally written in PDP-7 Assembly Language
  - Then in B
  - Then in C
  - B and C were basically created to write Unix
- Philosophy
  - Almost everything is a text file
  - Little programs (utilities) to do little tasks
  - Connect programs with pipes & redirection
    - % who | sort | lpr
    - Print an alphabetical list of who is active on the system
- Linux is an open software version of Unix
  - Since 1991
    - Linus Torvalds (the kernel)
    - Richard Stallman (GNU)
  - Widely used for high-performance computing

Programming Languages

- Some of the languages used in CS Dept
  - C, C++, C#
    - Many of the upper level courses (networks, distributed computing)
  - Java
    - 100, 211, 212
  - Matlab
    - 100M, numerical analysis courses
  - ML
    - functional programming
    - 312, logic-related courses
  - …
- Some other languages (from a Yahoo list)

  ABC, ActiveX, Ada, AMOS, APL, AppleScript, Assembly, awk, BASIC, BETA, C and C++, C#, Cecil, Cilk, CLU, COBOL, ColdC, cT, Curl, Delphi, Dylan, Dynace, Eiffel, Forth, Fortran, Guile, Haskell, Icon, IDL, Infer, Intercal, J, Java, JavaScript, JCL, JOVIAL, Limbo, Lisp, Logo, M - MUMPS, Magma, ML, Modula-2, Modula-3, Oberon, Obliq, Occam, OpenGL, Pascal, Perl, PL/I, Pop, PostScript, Prograph, Prolog, Python, Rexx, Ruby, SAS, Sather, Scheme, ScriptEase, SDL, Self, SETL, Smalltalk, SQL, Tcl/Tk, TOM, Verilog, VHDL, VRML, Visual, Visual Basic, Z

Scripting Languages

- A *script* is a sequence of common commands made into a single program
  - Unix uses *shell scripts*
  - The *shell* is the interactive interface to Unix
  - You can combine commands from the *Unix shell* to create programs
- A *scripting language* is
  - Usually easy to learn
  - Interpreted instead of compiled
- Example scripting languages: Unix shell, Python, Perl, Tcl (Tool command language)
- Some Python code:

```
class Stack (object):
    def __init__ (self):
            self.stack = [ ]
    def put (self, item):
            self.stack.append(item)
    def get (self):
            return self.stack.pop()
    def isEmpty (self):
            return len(self.stack) == 0
```

Regular Expressions

- Common goal: search/match/do stuff with strings
- Idea: use special strings to match other strings
  - Some characters are meta-characters
- *Regular expressions* are closely related to *finite state automata* (CS 381/481)
- Some of the rules for regular expressions
  - A regular character matches itself
  - A . matches any character
  - * implies 0 or more occurrences (of preceding item)
  - + implies 1 or more occurrences
  - \ implies following character is treated as a regular character
  - […] matches any one character from within the brackets; - can be used to indicate a range
- ([0-9]+\. | \. [0-9]) [0-9]*

Makefiles

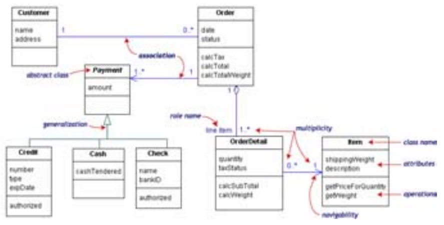
- Used when compiling/recompiling a large system (several interdependent files)
  - Checks which files have changed and only recompiles those that are necessary
  - Because of dependencies, more than just the changed files can need to be recompiled
  - Of course, can always recompile everything, but this can be too expensive
- Once you have a makefile
  - You recompile whatever is necessary by typing *make*
- To create a makefile
  - Usual strategy is to find some examples and modify them
  - There are automated tools for building makefiles

Version Control

- Allows you to keep track of changes for a large project
  - Can back up to old version if changes create problems
  - Multiple contributors can work on the system
- CVS (Concurrent Version System)
  - Open source
  - Widely used tool for version control
  - Maintains a history of all changes made
  - Supports branching, allowing several lines of development
  - Provides mechanisms for merging branches back together when desired

UML

- UML
  - = Unified Modeling Language
  - Design tool for object oriented programming
  - System for showing the interaction of objects



Profiling

- The goal is to make a program run faster
  - Rule of thumb: 80% of the time is spent in 20% of the code
  - No use improving the code that isn't executed often
  - How do you determine where your program is spending its time?
- People are notoriously bad at predicting the most computationally expensive parts of a program
- Part of the data produced by a profiler (Python)

```
 2649853 function calls (2319029 primitive calls) in 53.502 CPU seconds
   Ordered by: standard name
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    2521    0.227    0.000    1.734    0.001 Drawing.py:102(update)
    7333    0.355    0.000    0.983    0.000 Drawing.py:244(transform)
    4347    0.324    0.000    4.176    0.001 Drawing.py:64(draw)
    3649    0.212    0.000    1.570    0.000 Geometry.py:106(angles)
      56    0.001    0.000    0.001    0.000 Geometry.py:16(__init__)
343160/34316    9.818    0.000   12.759    0.000 Geometry.py:162(_determinant)
    8579    0.816    0.000   13.928    0.002 Geometry.py:171(cross)
    4279    0.132    0.000    0.447    0.000 Geometry.py:184(transpose)
```

Bali for Part 4

- Adds classes (and fields and methods) with single inheritance
- *Does not* remove functions
  - There is still a main-function, executed when program is run

```
class Stack
     { private Node top; } {}
     { public void put (int i) {}
               { top = Node(i, top);
               return; }
       public int get ()
               { Node n; }
               { n = top;
               top = top.link;
               return n.data; }
  }
```

```
int main ( )
     {int n; Stack s;} {
     n = 0;
     while n < 10 do {
               s.put(n); n = n + 1; }
     n = 0;
     while n < 10 do {
               print s.get();
               n = n + 1; }
      }

class Node
     { public int data; public Node link; }
     { public Node (data, line) {}
               { this.data = data;
               this.link = link;
               }
     }{}
```

New Bali Syntax

class -> class ***name*** [(***name***)] { fieldDeclaration* }
{ constructor* }
{ method* }
fieldDeclaration -> modifier variableDeclaration
constructor -> modifier ***name*** ([parameters]) functionBody
method -> modifier function
modifier -> public | private