Top 20 Tools of All Time
(http://uk.gizmodo.com/)

Software Tools

Lecture 9
CS 212 – Fall 2007

Integrated Development Environments

- An IDE usually includes
  - Source code editor (usually with color highlighting)
  - Compiler or interpreter
  - Tools for "build automation" (i.e., keeps track of what needs to be recompiled)
  - Debugger
  - Class browser (for languages with classes)
- Examples: DrJava, Eclipse
  - In Eclipse: As you type, gives you list of options + documentation
- You should know how to use a debugger!
  - Place breakpoints
  - Step through code
    - Step over
    - Step into
    - Step out of…
  - Examine current call-stack
  - Examine values of active variables
    - Some debuggers allow you to change a variable value
- Debuggers are usually *much more effective* than placing print-statements

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
- Mac OS X is built on Unix

Programming Languages

- Some of the languages used in Cornell's CS Dept
  - Java
    - 100, 211, 212
  - C, C++, C#
    - Many of the upper level courses (networks, distributed computing)
  - Matlab
    - 100M, numerical analysis courses
  - ML
    - Functional programming
    - 312, logic-related courses
  - …
- Fortran, C, C++ are used widely in Engineering
- 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
- A regular expression in Java

  "((\\.[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

Memory Management

- Modern programs are
  - Long running
  - Make dynamic use of memory
- Garbage collector
  - Some languages (e.g., Java, C#) use a garbage collector to reclaim unused memory
  - Other languages (e.g., C, C++) require programmers to manage their own memory
- Manual memory management bugs
  - Dangling pointers
    - Memory has been freed, but part of the code is still trying to use it
  - Memory leaks
    - Memory that is no longer used, but is not freed
    - Long running program ⇒ run out of memory
- There are tools to help catch such bugs
  - E.g., *purify* for C, C++

Garbage Collection

- Want to keep any object that can be reached from program's variables
  - Either directly or through other objects that can be reached
  - *Program's variables* = anything in the *call stack*
- Once "not-in-use" objects are found
  - Can reclaim the memory for re-use
  - Can also compact memory
    - I.e., move all the "in-use" objects to another memory block (without gaps between objects)

Garbage Collector Schemes

- Mark and Sweep
  - Mark every object as "not-in-use"
  - Starting from the call stack, visit every reachable object, marking it as "in-use"
  - Everything still marked "not-in-use" can be reclaimed
- Reference Counting
  - Every object keeps a count of how many pointers reference it
  - When count is zero, memory can be reclaimed
  - Problem: cycles!
- For either scheme
  - Can "stop the world"
  - Can interleave (i.e., take turns)
  - Can run concurrently
- Java's current garbage collector
  - A 2-tier scheme (old generation; new generation)
  - A mark-and-sweep method
  - With compaction
- Java's garbage collection scheme has changed as new Java versions were released

Use of Standard Data Structures

- Packages for widely-useful data structures
  - Java Collections Framework
  - C++ STL (Standard Template Library)
  - Provide tools for
    - Sorting & searching
    - Iteration
    - List
    - Set
    - Map (or dictionary)
    - Stack
    - Queue
    - Priority Queue
- For example, Java provides
  - Interfaces
    - List, Map, Set
  - Classes
    - ArrayList, LinkedList, HashMap, TreeMap, HashSet, TreeSet
  - Algorithms
    - Arrays.sort, Arrays.search,...

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
- SVN (Subversion)
  - An alternative to CVS
- 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

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)
```

More Advanced Profiling

- Need additional profiling tools for applications that
  - Are multithreaded
  - Use multiple cores

- Example: *VTune Performance Analyzer* (from Intel)
  - Can monitor
    - Memory usage
    - Performance during file I/O
    - Thread overhead and synchronization
    - Load balancing
    - Idle time
    - Communication bottlenecks

A List of Software Tools

(from Wikipedia)

- Revision control: Bazaar, Bitkeeper, Bonsai, ClearCase, CVS, Git, GNU arch, Mercurial, Monotone, PVCS, RCS, SCM, SCCS, SourceSafe, SVN, LibreSource Synchronizer
- Interface generators: Swig
- Build Tools: Make, automake, Apache Ant, SCons, Rake, Flowtracer
- Compilation and linking tools: GNU toolchain, gcc, Microsoft Visual Studio, CodeWarrior, Xcode, ICC
- Static code analysis: lint, Splint
- Search: grep, find
- Text editors: emacs, vi
- Scripting languages: Awk, Perl, Python, REXX, Ruby, Shell, Tcl
- Parser generators: Lex, Yacc, Parsec
- Bug Databases: gnats, Bugzilla, Trac, Atlassian Jira, LibreSource
- Debuggers: gdb, GNU Binutils, valgrind
- Memory Leaks/Corruptions Detection: dmalloc, Electric Fence, duma, Insure++
- Memory use: Aard
- Code coverage: GCT, CCover
- Source-Code Clones/Duplications Finding: CCFinderX
- Refactoring Browser
- Code Sharing Sites: Freshmeat, Krugle, Sourceforge, ByteMyCode, UCodit
- Source code generation tools
- Documentation generators: Doxygen, help2man, POD, Javadoc, Pydoc/Epydoc

- No hammer? No screw or screwdriver?
- Why the rifle and not the cannon? Why the watch and not the clock?
- No electricity?