**Cross Platform Design**

**Overview**
- What A Platform Is
- The Problem
- Naïve Solutions
- Feasibility Concerns
- Recommended Solutions
- Some Common Pitfalls

---

**What is a Platform?**
- Combination of Several Factors
  - Hardware Architecture (x86, ppc, arm)
  - Operating System
  - Compiler
  - Standard Libraries

---

**The Problem**
- How do you design an application for multiple platforms?
  - Which platforms to support?
  - How much does each platform vary?
  - How much consistency is needed between platforms? Same results? Share data files? Similarity of interface? Direct communication between applications on different platforms?
Koé: An example

- Koé is a cross-platform audio generation application.
- Needs:
  - Linux, MacOS, Windows audio support
  - Unified GUI
  - Interchangeable data
  - Unified Plugin Architecture
  - Unified Scripting Architecture

Naïve Solution

- Program in Java (or other "multi-platform" language)
  - Allows single code base
  - "Write Once, Run Anywhere"
- Rewrite program for each platform
  - Good implementation for each platform
  - Easy to optimize
  - Allows one to write in language of choice
  - Can leverage platform specific features

Feasibility Concerns

- Java
  - Performance Concerns
  - Inability to access hardware without customized platform native code.
- Multiple code bases
  - N-platforms = n times the work
  - Little to no code-reuse
  - Difficult to synchronize features
  - Maintainance is a nightmare

Several Solutions

- The "#ifdef" method
- Rewrite only the parts needed
- Use only cross-platform libraries
- Some combination of these is frequently needed
The “#ifdef” Method

```c
#ifdef WIN32
long int GetTDiff(TDiff* t)
{
    /* do windows implementation */
} #endif

#ifdef UNIX
long int GetTDiff(TDiff* t)
{
    /* do unix implementation */
} #endif
```

Rewrite Parts Needed

- Break apart platform specific and non-platform specific code
  - Each part gets its own files.
  - Compile only what is needed on any given platform, and link only what is needed.
- Increases code reuse
- Insures similar results across platforms
- Still, needs lots of platform specific code

Use cross-platform libraries

- Many libraries are available supporting common features
  - Threads (pthreads, ZThreads)
  - Graphics (SDL, OpenGL)
  - Networking (libnet, netlib)
  - Math (BLAS, gmp, fftw)
  - Windowing Systems (gtk, tk)
- Some support almost everything
**wxWindows**

- “the open source, cross-platform GUI framework”
- www.wxwindows.org
- Supports many features and platforms

**wxWindows Platforms**

- Windows
- *nix/GTK+
- *nix/Motif and X11
- MacOS 9, MacOS X
- OS/2
- Embedded systems (Nano-X, Windows CE, MGL)

**wxWindows Features**

- Basic windows and widgets
- Common dialogs
- Graphics contexts and OpenGL
- Networking and IPC
- Threading
- Printing
- And much much more...

**Koé: Implementation**

- Unified plugin architecture allows single code base for multiple platforms.
  - Platform-specific code exists only in pluggable modules
  - Standard plugins can be written once, and compiled for each platform
- wxWindows makes it possible
  - GUI code is consistent across platforms
  - File loading and saving handled by wxWindows
  - Shared object code handled automatically
Common Pitfalls

- Don’t assume that all compilers have the same features.
- Even if the library is cross platform, test on all platforms, don’t assume that if it works on one, it will work on all of them.
- Sometimes it is just easier to rewrite a small section of code for each platform.
- Bad code that runs on multiple platforms is still bad code.

Outtakes

- Use wxWindows!
- Use cvs!
- OpenGL “rulez” (so does Python)
- Come see New Wave Vaudeville on Saturday, at Historic Ithaca’s State Theatre, 8:00 PM.