Managing Code Variants

Steven J Zeil

March 19, 2013


Outline


Outline I

Code Variations

- Environment management, Previously identified as common SCM problems:
  Coping with change in
  - hardware environment
  - software environment
- Can lead to need for variant code to support different configurations


The Sad Story of C/C++ Portability

- Both C and C++ existed as popular languages long before being standardized
  - Widespread variations in the "system" headers
- Even after standardization, many common functions are not standardized
  - GUIs
  - multi-threading and distributed operations
  - netowrk communcations
- Even things covered by the standard aren't covered in enough detail


C Portability Quiz

How would you declare an integer counter capable of holding non-negative values up to one million? Up to one billion?


C Portability Quiz

How would you declare an integer counter capable of holding non-negative values up to one million? Up to one billion?

- C90 requires `sizeof(short)` $\leq$ `sizeof(int)` $\leq$ `sizeof(long)`
  Notice that's $\leq$, not $<$
  A textttchar must hold a "natural" byte (minimum addressable unit) on the machine architecture.


C Portability Quiz

How would you declare an integer counter capable of holding non-negative values up to one million? Up to one billion?

- C90 requires `sizeof(short)` $\leq$ `sizeof(int)` $\leq$ `sizeof(long)`
  Notice that's $\leq$, not $<$
  A textttchar must hold a "natural" byte (minimum addressable unit) on the machine architecture.
- The C99 specification added `long long` and set minimum sizes as

`char`	8
`short`	16
`int`	16
`long`	32
`long long`	64


C++ Portability Quiz

How would you declare an integer counter capable of holding non-negative values up to one million? Up to one billion?


C++ Portability Quiz

How would you declare an integer counter capable of holding non-negative values up to one million? Up to one billion?

- The C++ standard followed C90 (not 99!) until C++11
  `sizeof(short)` $\leq$ `sizeof(int)` $\leq$ `sizeof(long)`


C++ Portability Quiz

How would you declare an integer counter capable of holding non-negative values up to one million? Up to one billion?

- The C++ standard followed C90 (not 99!) until C++11
  `sizeof(short)` $\leq$ `sizeof(int)` $\leq$ `sizeof(long)`
- C++11 (no0t yet implemented by most compilers) adds the C99 standards


Coping With Variants in the C/C++ World

- Configuration headers used to define symbols describing selected variants, e.g.,
  config.h
- Code uses symbols defined in there
  - direct substitution, e.g.

```
#include MEM
```

    loads <alloc.h> or <mem.h>
  - or conditionally

```
#ifdef USE_WINSOCK
#include <winsock2.h>
#else
#include <netinet/in.h>
#include <sys/socket.h>
#endif
```


Outline I


Compiling Software the Unix Way

If you've ever installed a Unix/Linux package from a source distribution, you've probably gotten used to the two-step process:

```
./configure
make
make install
```

- The configure script runs a series of tests on the compilation environment, e.g.,
  - operating system
  - compiler name
  - availability of selected libraries/header files
  - availability and/or behavior of selected functions
- Produces a `Makefile` and a configuration header `config.h` based upon the test results
- Source code may use conditional compilation based on the header to select appropriate code


Generating The configure Script

A rough outline:

1. Create a `configure.ac`

```
AC_INIT(cppSpreadsheet, 1.0, zeil@cs.odu.edu)
AC_PREREQ([2.68])
AM_INIT_AUTOMAKE([1.16 foreign no-define])
AC_CONFIG_HEADERS([config.h])
AC_PROG_CXX
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```


Generating The configure Script

2. Set up config.h.in (template for eventual config.h file)
3. Set up Makefile.am

```
AM_INIT_AUTOMAKE([1.10 no-define foreign])

bin_PROGRAMS = testssheet

testssheet_SOURCES=testssheet.cpp exprparser.cpp tok
      cellname.cpp numericnode.cpp stringnode.cpp cell
      absnode.cpp sqrtnode.cpp sumnode.cpp lessnode.cp
      greaternode.cpp greatereqnode.cpp equalnode.cpp
      subtractnode.cpp timesnode.cpp dividesnode.cpp
      numvalue.cpp strvalue.cpp errvalue.cpp spreadsh
      observable.cpp observerptrseq.cpp cellptrseq.cpp
      absnode.h          control.h          lessnode.h
ssi.h \
      binarynode.h       dividesnode.h      minusnode.h
ssview.h \
```

Generating The configure Script

4. `touch NEWS README AUTHORS ChangeLog`
   or create real versions of these.
5. run `autoreconf -force -install`
   - Runs the sequence of programs: `aclocal autoconf autoheader automake`
   - Creates `config.h.in Makefile.in` & `configure`


Alternatives

- `imake` for X code


Outline I


autoconf is C/C++-centric

The `configure` approach relies heavily on conditional compilation features.

- Common in C++
- Only in Java via non-standard techniques


Java: Abstraction I

Java programs are more likely varied be altering entire classes at a time.
For example:

```
public abstract class OCRLauncher extends Thread {
    /**
     * Launch an OCR process to convert the input
     * PDF into some kind of File  of OCR output.
     *
     * @param inputPDFfile The PDF file to be converted
     * @param outputFile The raw OCR output
     * @return
     */
    public abstract boolean convertPDFtoOCR
        (File inputPDFfile, File outputFile)
            throws Exception;
```


Java: Abstraction II

```
    /**
     * Convert a file of OCR output into IDM
     *
     * @param inputOCRfile
     *
     * @return XML (IDM) document
     */
    public abstract Document convertOCRtoIDM
        (File inputOCRfile) throws Exception;
}
```

This class has distinct implementations for different OCR programs that might be installed on the running system.


Configuration via Property Files

A property file, loaded at run time, specifies which class is actually desired:

```
input.OCRLauncherClass=edu.odu.cs.extract.input.OCRBatch
input.OCRProgram=OCR
input.OCRBatch=Batch
input.ocr.in_dir=c:/Luratech/ocr_in
input.ocr.out_dir=c:/Luratech/ocr_out
```


Reflection: Dynamic Loading

And the desired class is loaded dynamically:

```
String OCRLauncherName
    = p.getProperty(Properties.Names.OCR_LAUNCH_CLASS);
Class<?> ocrLauncherClass
    = Class.forName(OCRLauncherName);
ocr = (OCRLauncher)ocrLauncherClass.newInstance();
idmDoc = ocr.convertOCRtoIDM(inputOCR);
```