

muS

<http://code.google.com/p/mus/source/browse/#svn%2Ftrunk%2Fsrc>

Irene Alvarado – Project Manajer
Jonathan Dunn – Language Guru
Richard Boyle – System Integrator
Farbound Tai – Verification & Validation
Taylor Owens - System Architect



What Is muS?

♪ muS is a tool to help anyone build and analyze a piece of music in a simple, intuitive way.



Why muS?

♪ Reading sheet music is confusing, even for the experienced musician

♪ muS provides immediate visual feedback to the creator of the piece

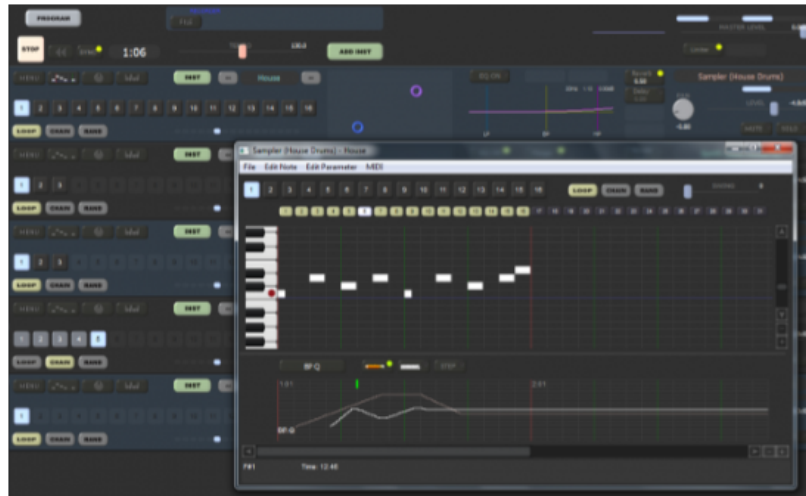
FAERIE'S AIRE and DEATH WALTZ
(from "A Tribute to Zdenko G. Fibich")
Words and Music by John Stump
Arranged by Accidion

Based on a Cro-magnon skinning chant



Why muS?

♪ Other digital music software allows creation of music, but almost none provide a means to create useful visual analysis



Why muS?

♪ MuS attempts to address this void by allowing the programmer to specify color and shape to notes in order to see music in a different way

♪ More control than other visual editors, but better visual analysis than robust .midi programming



Music Composition

♪ Manually write out each note/GUI with software



♪ Insert notes?

♪ Change the pitch for every other note?

♪ Increase octave of last 2 notes in every measure?

♪ Generate new song w/ similar sub-sequences?



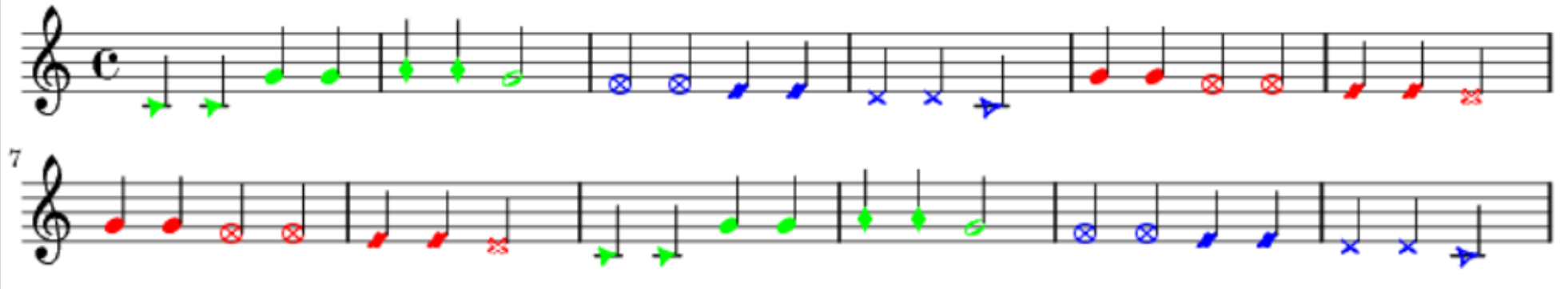
Our language: muS

- ♪ Easily change attributes of a set of notes
- ♪ A brand new way to explore music composition
- ♪ Use appropriate data structures to represent music
 1. *Efficient storage for notes, chords, measures, and attributes*
 2. *Ease of access*
- ♪ Provide suitable operators and built-in functions
`seq1 << seq2[3:10] + seq2[0,4,5] + [seq3,seq4]`

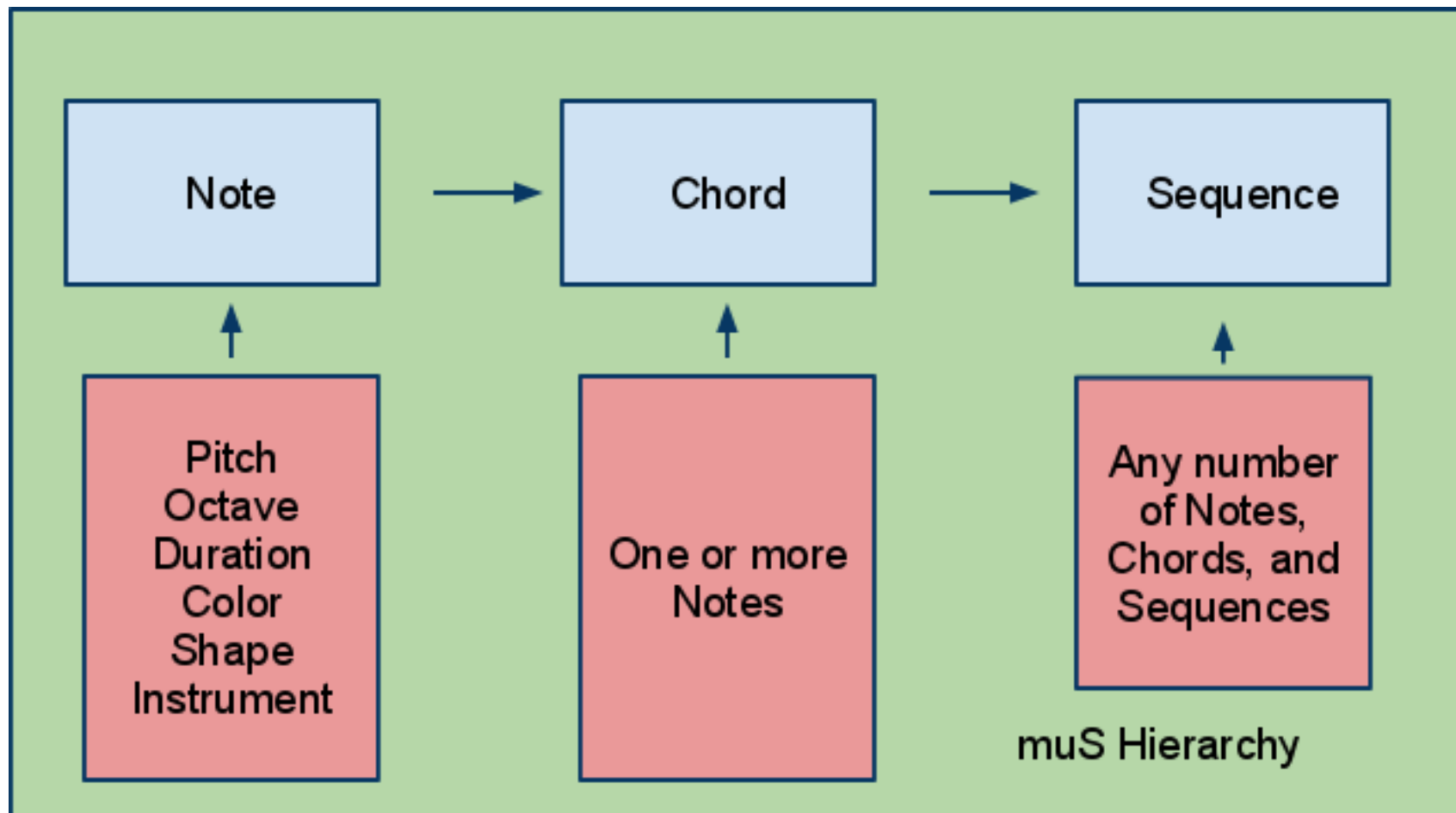


Graphical Representation

- ♪ Manually change the shape/color of any note
- ♪ Change representation for entire sequences
- ♪ Immediate visual clues to help analyze the music that has just been created



muS Hierarchy



Notes

♪ Attributes

♪ Pitch

♪ Duration

♪ Octave

♪ Shape

♪ Color

♪ Instrument

♪ Example syntax

```
Note n1 = new Note (A,4,4);
```

```
n1<Instrument('guitar');
```

```
n1<Color('green');
```

```
n1<Shape('triangle');
```



Chords

- ♪ Comprised of any number of Notes
- ♪ Played simultaneously
- ♪ Example syntax:

```
Note a = new Note(A,4,4);
```

```
Note b = new Note(B,4,4);
```

```
Note c = new Note(C,4,4);
```

```
Chord c1 = new Chord (a,b,c);
```



Sequences

♪ Creation

- ♪ Sequences are built up from Notes, Chords and Sequences
- ♪ Allow for Repeating Melodies
- ♪ Built in functions allow for easy manipulation
 - ♪ Subsequences
 - ♪ Subsets
 - ♪ Single Chords, or Notes
 - ♪ Changing attributes

♪ Analysis

- ♪ midi Output
- ♪ Visual Patterns



Built in functions

♪ foreach

♪ Allows for changing of an attribute of an entire sequence

♪ `foreach(seq1)<Instrument('bird tweet');`

♪ Subsequence and Subset

♪ Allows the programmer to get a certain portion of a sequence

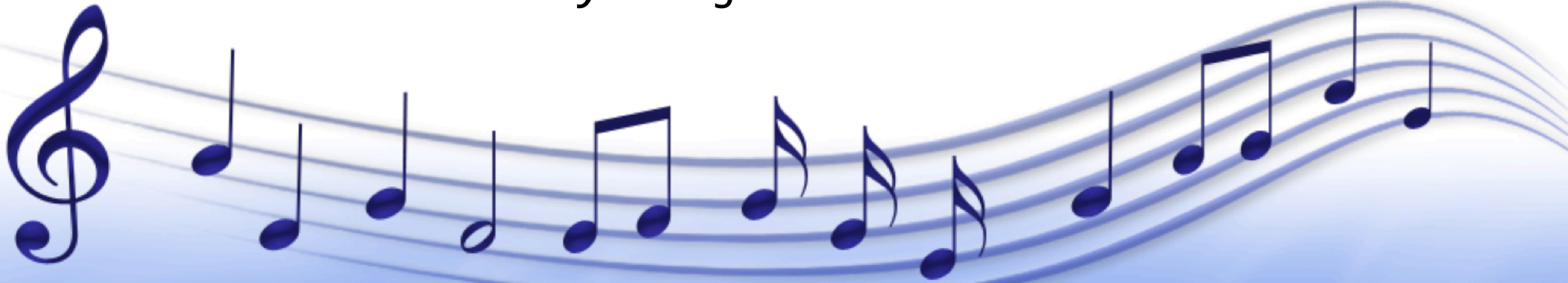
♪ `seq[0:4]; //Subsequence`

♪ `seq[0,4,7]; //Subset`



Lexical Analyzer

- ♪ Built using JLex (.lex file)
 - ♪ The Java equivalent of Lex for C
- ♪ Breaks muS code into token
 - ♪ ID
 - ♪ Numbers
 - ♪ Keywords
 - ♪ Grammatical symbols and operators
 - ♪ Quoted Text
 - ♪ Comments (ignored)
- `next_token()` returns a `java_cup.runtime.Symbol` object (compatible with CUP)
- Generates file called `Yylex.java`



Semantic Analyzer

- ♪ Built using CUP (.cup file)
- ♪ Constructor of Useful Parsers
 - ♪ Defines terminals for each token in Lexical Analyzer
 - ♪ Defines non-terminals used in grammar
 - ♪ Constructs a new ParseTree object
- ♪ Defines grammar of muS and invokes Java code (in ParseTree.java)
- ♪ Generates two classes:
 - ♪ ParserSym.java → constant declarations for each token type
 - ♪ Parser.java → actually executes the parsing



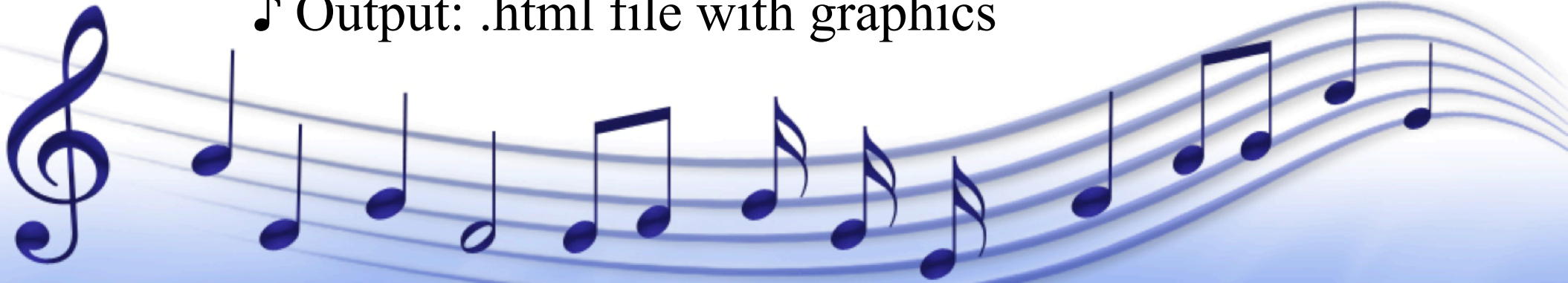
ParseTree.java

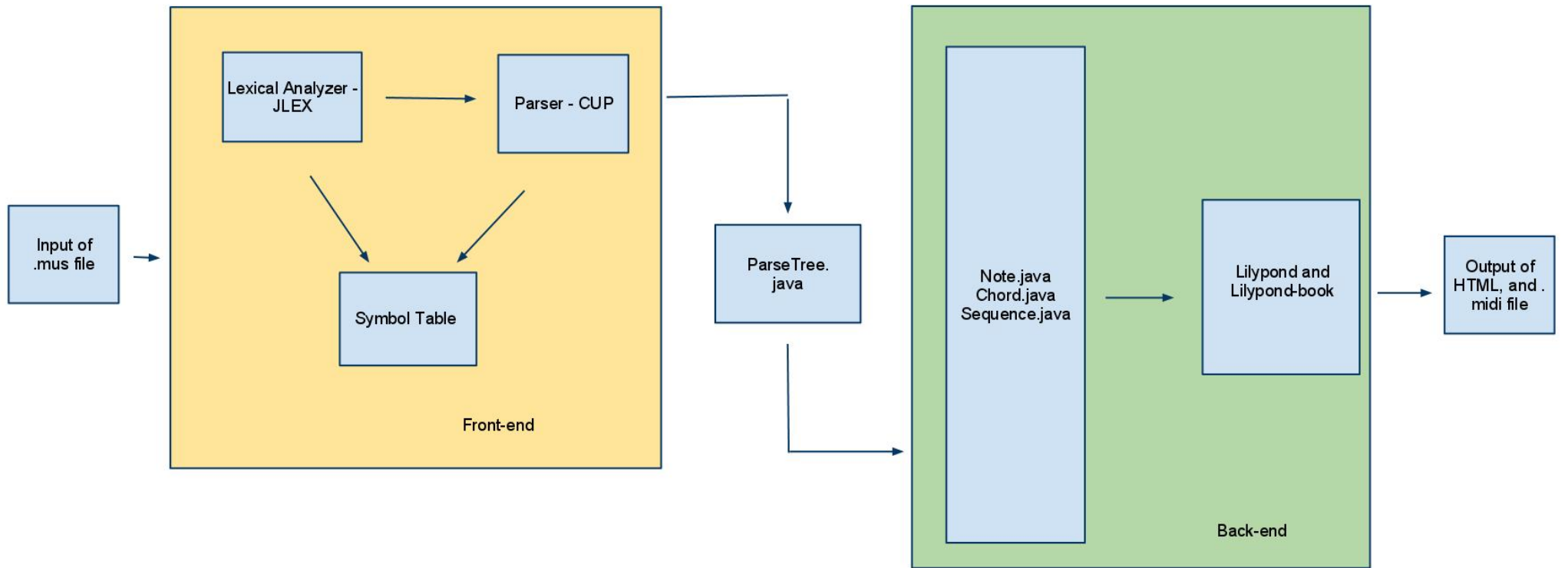
- ♪ Declares, initializes, and stores variables
- ♪ Code for built-in functions and operators
- ♪ Checks for errors, declarations, and initialization
- ♪ Works with all the other Java classes
 - ♪ Note.java → represents a Note
 - ♪ Chord.java → represents a Chord
 - ♪ Sequence.java → represents a Sequence
 - ♪ Reference.java → stores available colors, shapes, and instruments



Lilypond

- ♪ Program used to produce displayed music score
- ♪ LilypondConvert.java
 - ♪ Converts sequence into acceptable format for lilypond
- ♪ Lilypond:
 - ♪ Input: .ly file
 - ♪ Output: .midi file
- ♪ Lilypond-book:
 - ♪ Input: .html file without graphics
 - ♪ Output: .html file with graphics





Example – Pachelbel.Canon.mus

Pachelbel Canon

Musical score for Pachelbel's Canon in D major, measures 1-19. The score is written in treble clef with a common time signature (C). The key signature has two sharps (F# and C#). The notation includes various note values and rests, with specific notes highlighted in pink, blue, cyan, red, and green. Measure 10 is marked with a '10' at the beginning of the second line.

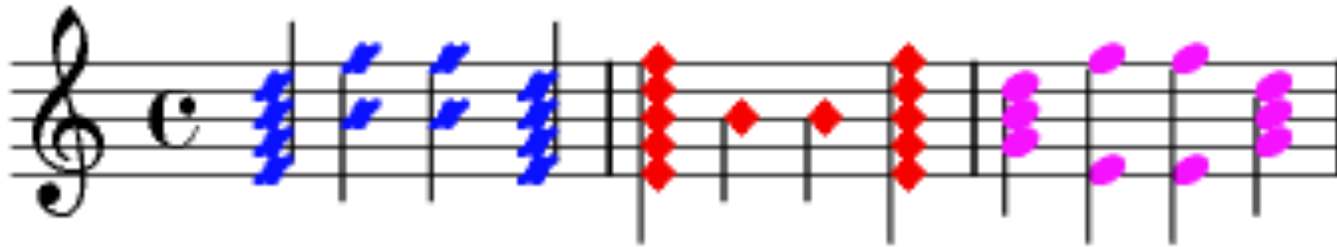


<http://mus.googlecode.com/svn/trunk/src/testfiles/Pachelbel.Canon/Pachelbel.Canon.html>



Example – Aho.mus

AHO



<http://mus.googlecode.com/svn/trunk/src/testfiles/Aho/Aho.html>



Lessons Learned

- ♪ Communication and Version control
- ♪ Insufficient Planning → (Implementation Effort)^N
- ♪ Planning and Implementation is an Iterative Process
- ♪ Start Early!

