Triggered Guitar Effects Platform: Design Review

Ralph Carlo Quinto (CE), Bryan Guner(EE), Haley Scott(EE)

Advisor: Dr. Ambrose Adegbege

Overview

- Problem Identification
- Project Goals
- Team Breakdown
- Project Overview
- Video Demonstration
- Detailed Specifications
- Action Items
- Milestones
- Schedule
- Budgets

Problem Identification

- Guitar effects pedals restrain the guitar player to the area of the stage where their pedal board is located
- Analog and digital effects pedals are more expensive than digital effects software

Project Goals

- Analyze a sequence of notes (frequencies) played in time to cue control action
    - Trigger at user defined points
    - Concurrent effects
    - Tolerate inconsistencies in performance
    - Recognize trigger points within an acceptable latency

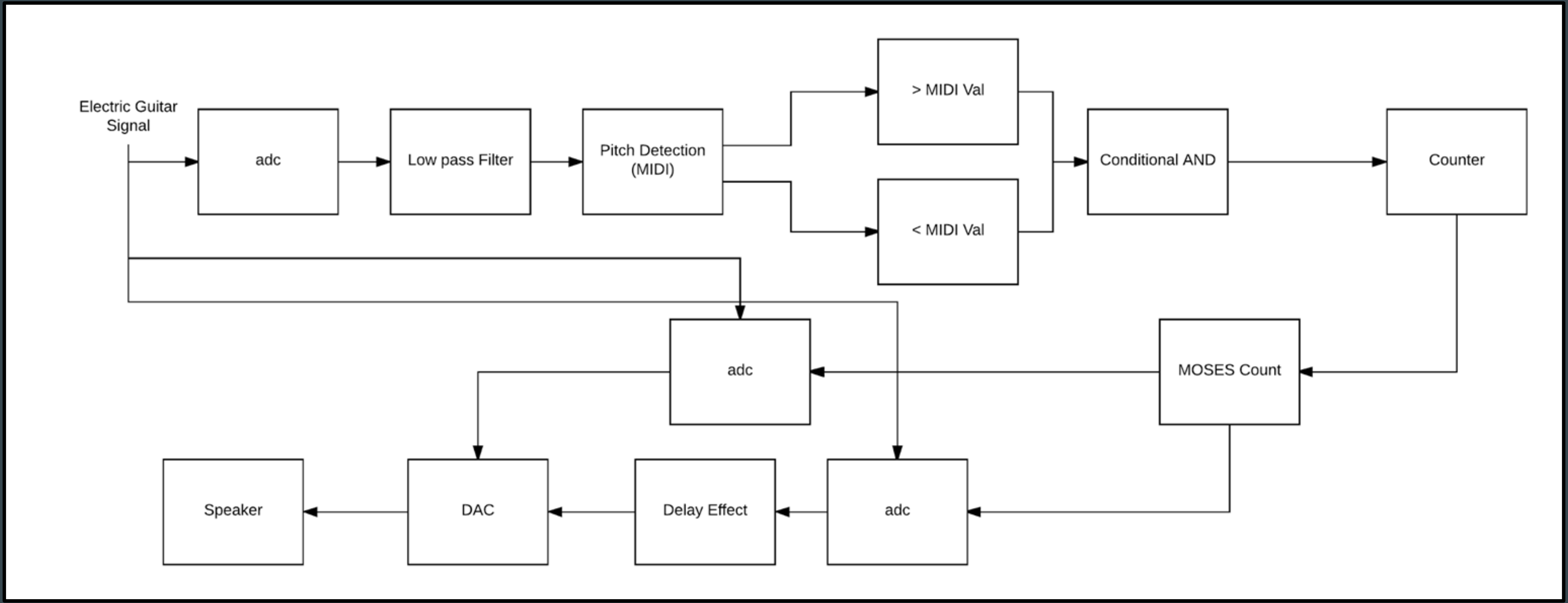
Team Breakdown

Bryan Guner	Team Lead - Develop a protocol for digital signal processing of the guitar signal in order to create a time-sequential record of the frequency content of the guitar signal and a comparison between pre recorded songs and live performances.
Ralph Quinto	Software Engineer - Research for possible programming platforms. Responsible for reading electric guitar signals, creating the signal analysis patches in pure data, and triggering digital guitar effects.
Haley Scott	Architectural Manager - Responsible for ensuring successful integration of project components, researching system methods, designing digital effects, Project and Organizational Management.

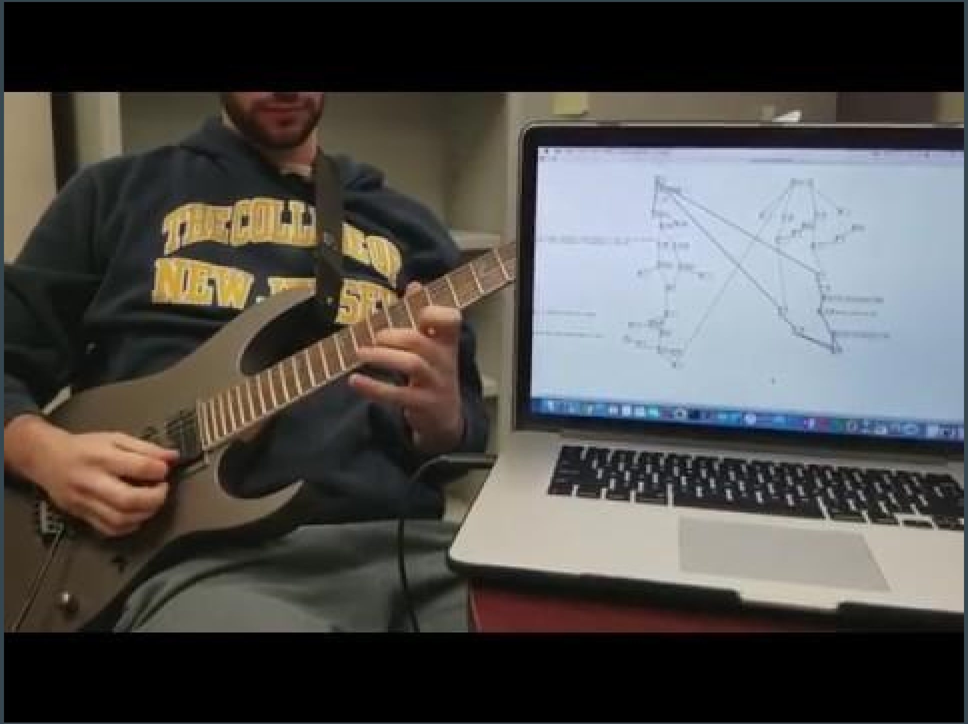
Detailed Specifications

- Pure Data Specs
  - Sampling rate of 44,100 Hz
  - Block size of 1024
- Electric Guitar: Ibanez RG5EX1
  - Bridge Pickup: Infinity 4
    - Magnet: Ceramic
    - DC Resistance: 15.6 KΩ
  - Gauges: .009/.011/.016/.024/.032/.042

Project Overview



Video Demonstration (Current Model)



Open Action Items

- Convert pseudo code algorithm into C
- Import C code as Pure Data external
- Design more complex digital effects

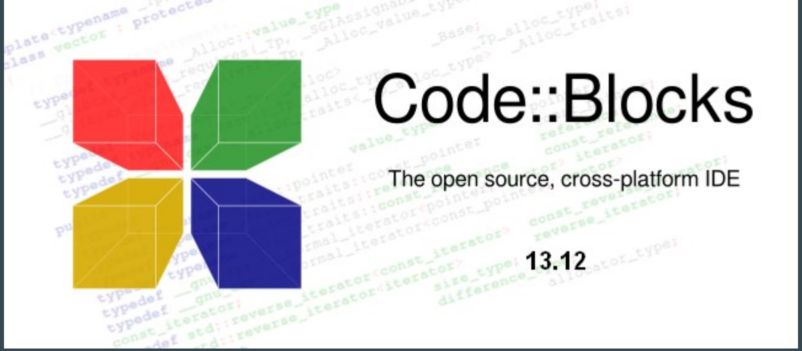
Converting Pseudo Code Algorithm to C

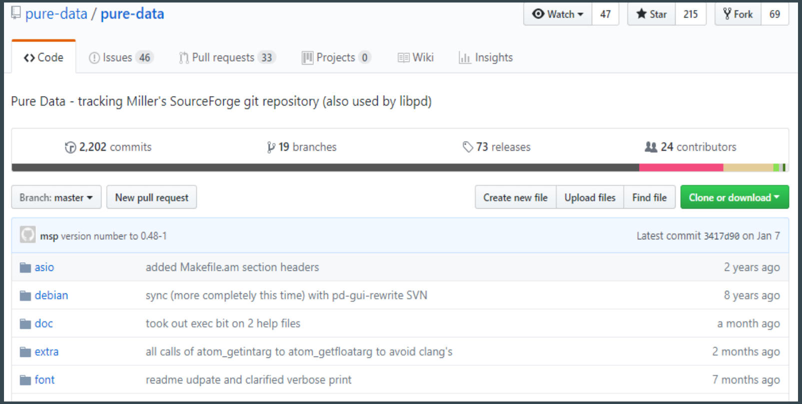
```
public static void main(String[] args) {
    double [] horizontalInput1 = {1.0, 1.0, 2.0, 3.0, 2.0, 0.0};
    double [] verticalInput1 = {0.0, 1.0, 1.0, 2.0, 3.0, 2.0, 1.0};
    DTW similarInputs = new DTW(horizontalInput1, verticalInput1);
    double result = similarInputs.compute();

    double [] horizontalInput2 = {0.0, 1.0, 1.0, 2.0, 3.0, 2.0, 1.0};
    double [] verticalInput2 = {0.0, 1.0, 1.0, 2.0, 3.0, 2.0, 1.0};
    DTW sameInputs = new DTW(horizontalInput2, verticalInput2);
    double result2 = sameInputs.compute();

    System.out.println(result2);

    double [] horizontalInput3 = {1.0, 1.0, 2.0, 3.0, 2.0, 0.0};
    double [] verticalInput3 = {0.0, 0.0, 1.0, 2.0, 1.0, -1.0};
    DTW sameInputsOutPhase = new DTW(horizontalInput3, verticalInput3);
    double result3 = sameInputsOutPhase.compute();

    System.out.println(result3);

    double [] horizontalInput4 = {1.0, 2.0, 3.0, 4.0, 2.0, 1.0};
    double [] verticalInput4 = {5.0, 0.0, 1.0, 4.0, 2.0, 0.0, 3.0};
    DTW differentInputs = new DTW(horizontalInput4, verticalInput4);
    double result4 = differentInputs.compute();
}
```

```
private double compute() {
    double[][] distanceMatrix = buildDistanceMatrix();
    MatrixTriplet[][] leftAndBottomValuesTripletMatrix =
            buildTripleMatrixWithLeftAndBottomValues(distanceMatrix);
    MatrixTriplet[][] finalMatrix = buildFinalMatrix(leftAndBottomValuesTripletMatrix);
    return min(finalMatrix[verticalInput.length][horizontalInput.length]);
}
```

```
private static MatrixTriplet[][] buildFinalMatrix(MatrixTriplet[][] leftAndBottomMatrix){
    int rows = leftAndBottomMatrix.length;
    int cols = leftAndBottomMatrix.length;

    MatrixTriplet[][] finalMatrix = new MatrixTriplet[rows][cols];

    for(int row = 0; row < rows; ++row) {
        for(int col = 0; col<cols; ++col) {
            finalMatrix[row][col] = new
                    MatrixTriplet(leftAndBottomMatrix[row][col].initial,
                            leftAndBottomMatrix[row][col].left,
                            leftAndBottomMatrix[row][col].bottom,
                            leftAndBottomMatrix[row][col].bottomLeft);
        }
    }
    for(int row = 2; row < rows; ++row) {
        for(int col = 2; col < cols; ++col) {
            finalMatrix[row][col].left = min(finalMatrix[row][col -1])
                    + finalMatrix[row][col].initial;
            finalMatrix[row][col].bottom = min(finalMatrix[row-1][col])
                    + finalMatrix[row][col].initial;
            finalMatrix[row][col].bottomLeft = min(finalMatrix[row - 1][col - 1])
                    + finalMatrix[row][col].initial;
        }
    }
    return finalMatrix;
}
```

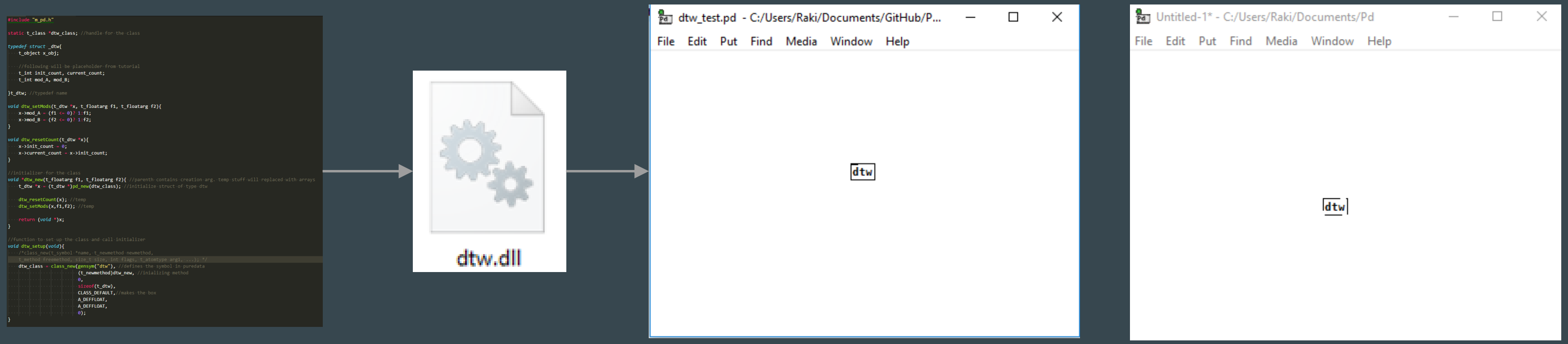
Converting Pseudo Code Algorithm to C Cont.



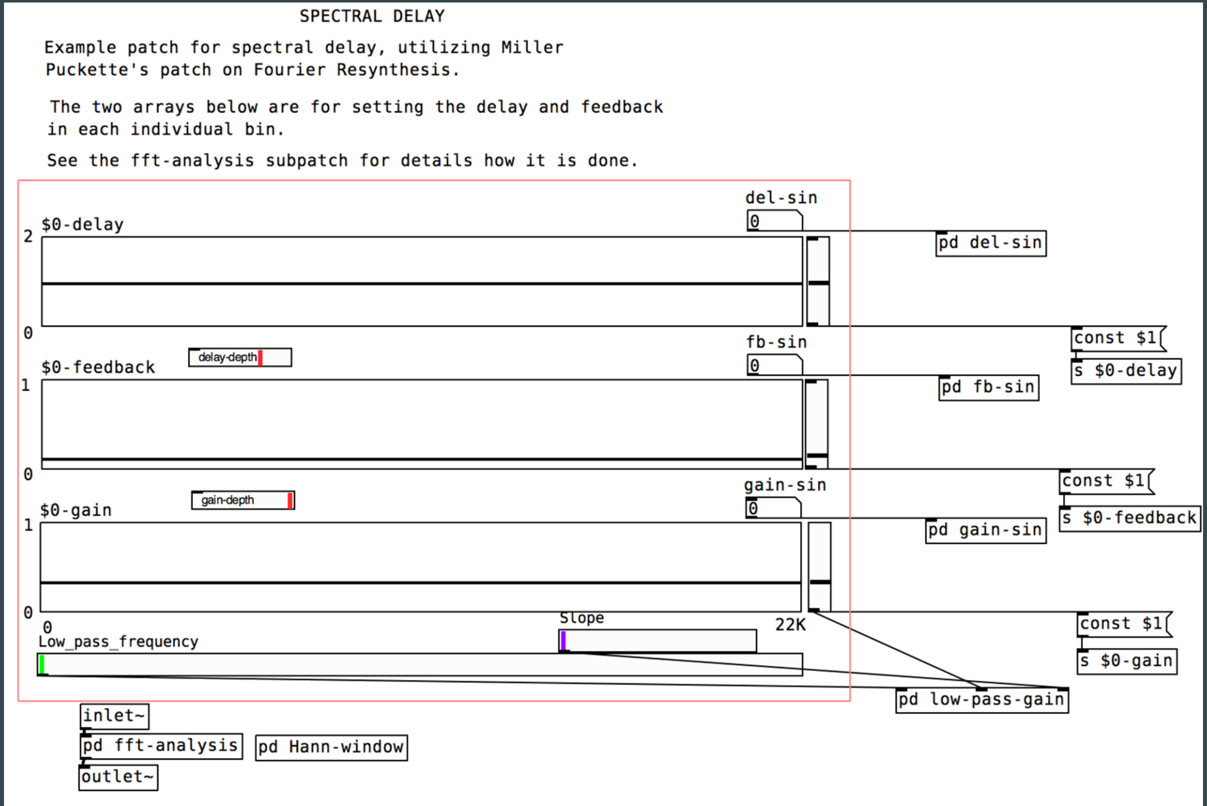



```c
#include "m_pd.h"

static t_class *dtw_class; //handle for the class

typedef struct _dtw{
    t_object x_obj;

    //following will be placeholder from tutorial
    t_int init_count, current_count;
    t_int mod_A, mod_B;

}t_dtw; //typedef name

void dtw_setMods(t_dtw *x, t_floatarg f1, t_floatarg f2){
    x->mod_A = (f1 <= 0)? 1:f1;
    x->mod_B = (f2 <= 0)? 1:f2;
}

void dtw_resetCount(t_dtw *x){
    x->init_count = 0;
    x->current_count = x->init_count;
}

//initializer for the class
void *dtw_new(t_floatarg f1, t_floatarg f2){ //parenth contains creation arg. temp stuff will replaced with arrays
    t_dtw *x = (t_dtw *)pd_new(dtw_class); //initialize struct of type dtw

    dtw_resetCount(x); //temp
    dtw_setMods(x,f1,f2); //temp

    return (void *)x;
}

//function to set up the class and call initializer
void dtw_setup(void){
    /*class_new(t_symbol *name, t_newmethod newmethod,
    t_method freemethod, size_t size, int flags, t_atomtype arg1, ...); */
    dtw_class = class_new(gensym("dtw"), //defines the symbol in puredata
                          (t_newmethod)dtw_new, //inializing method
                          0,
                          sizeof(t_dtw),
                          CLASS_DEFAULT,//makes the box
                          A_DEFFLOAT,
                          A_DEFFLOAT,
                          0);
}
```

Generating DTW Pure Data External



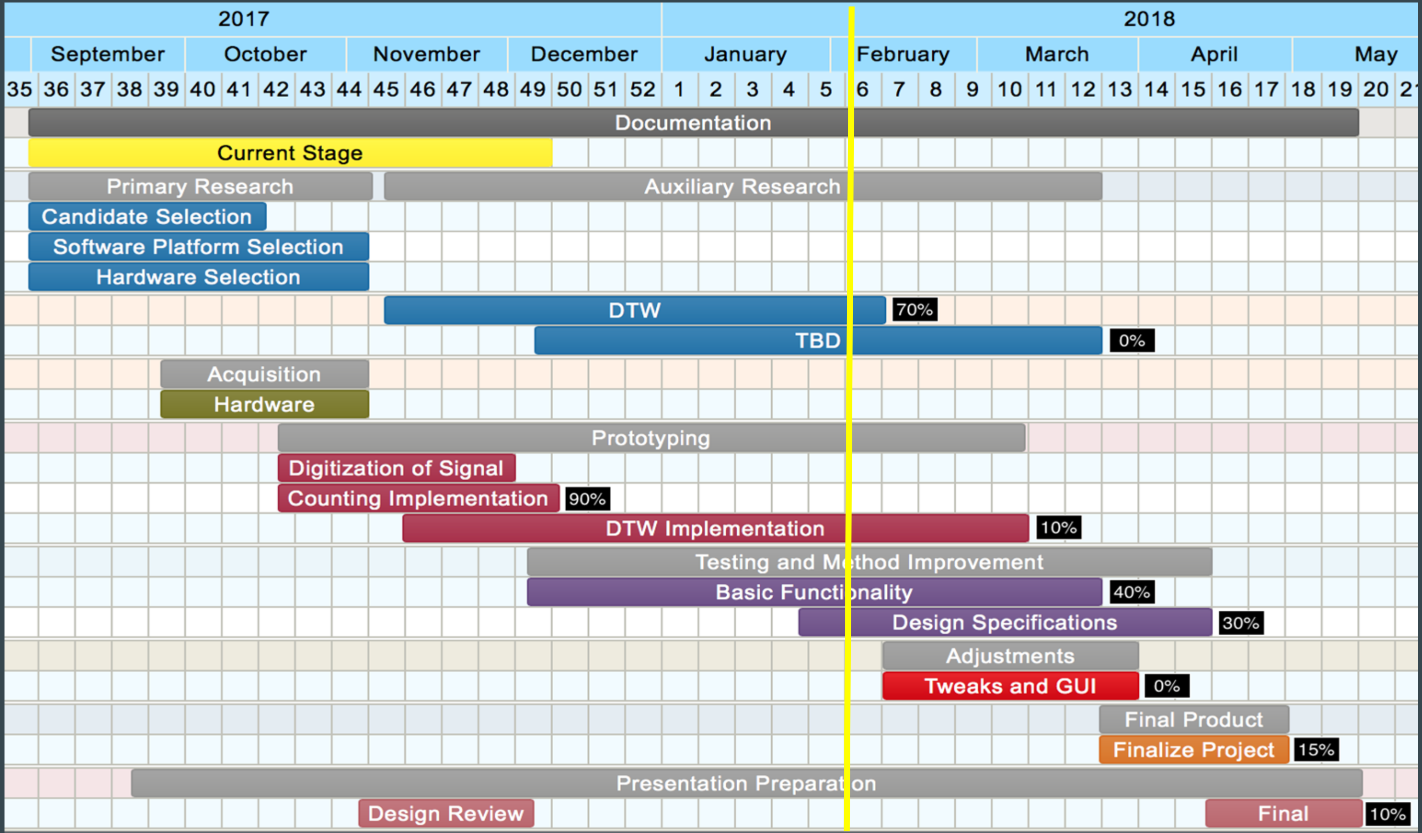
Designing Digital Effects



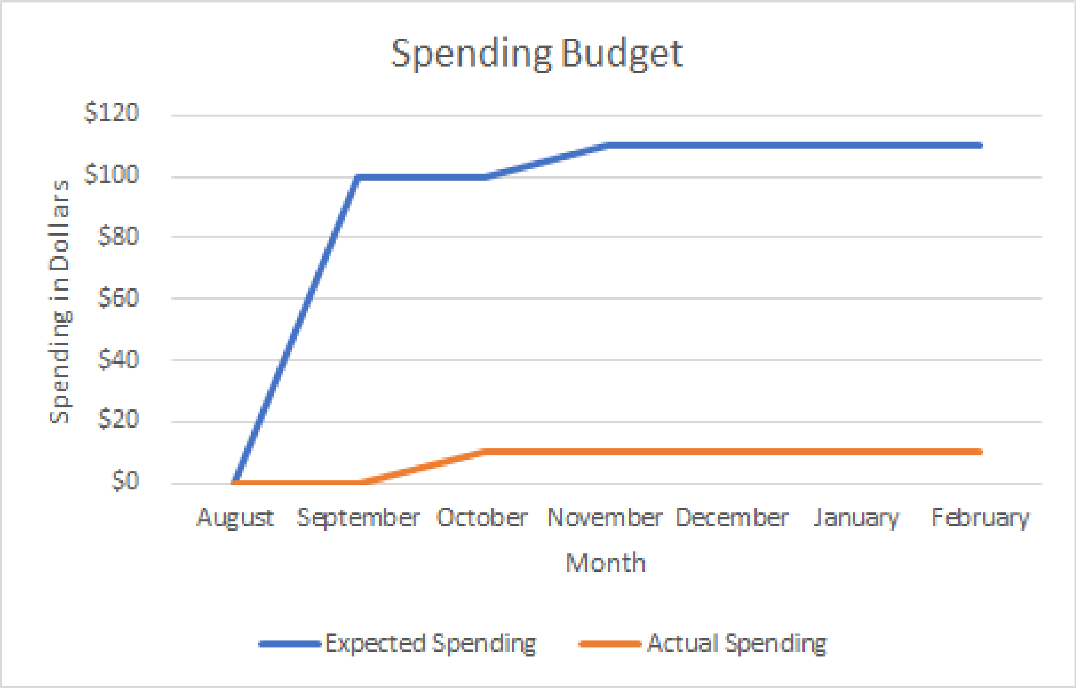
Anticipated Milestones

- Integration of system components
- Development of GUI
- Testing and validation
- Modification and improvement

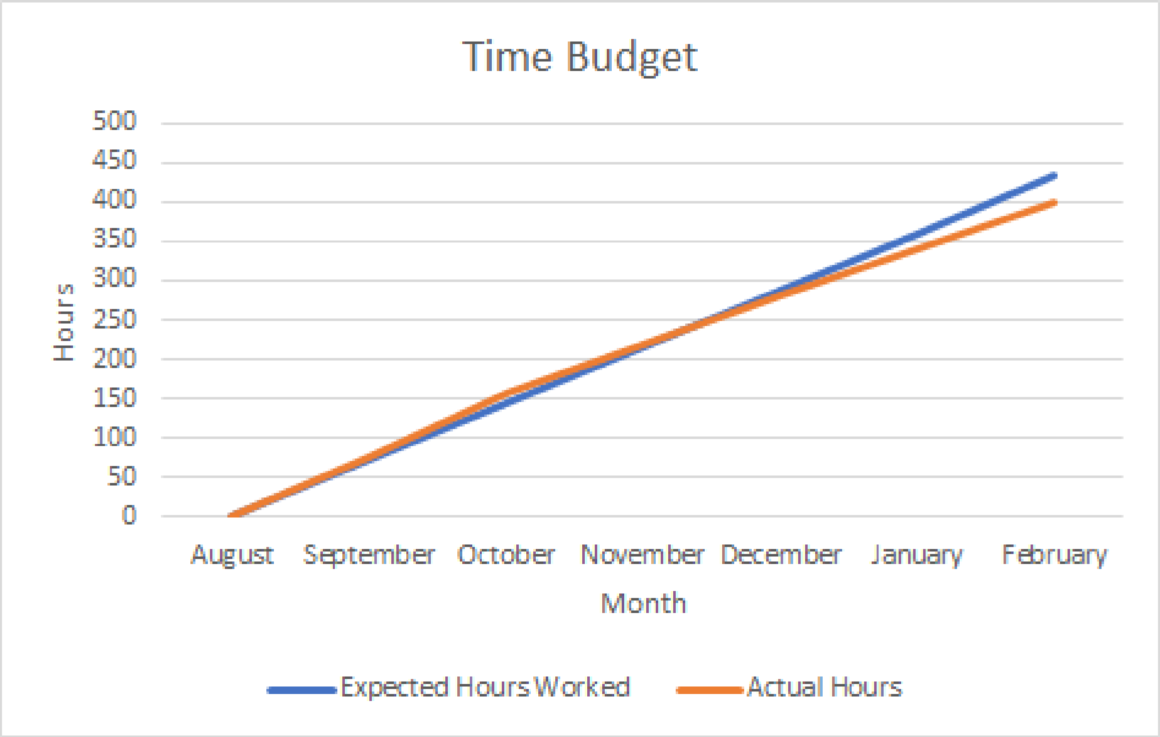
Schedule


2017
2018
September
October
November
December
January
February
March
April
May
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
Documentation
Current Stage
Primary Research
Auxiliary Research
Candidate Selection
Software Platform Selection
Hardware Selection
DTW
70%
TBD
0%
Acquisition
Hardware
Prototyping
Digitization of Signal
Counting Implementation
90%
DTW Implementation
10%
Testing and Method Improvement
Basic Functionality
40%
Design Specifications
30%
Adjustments
Tweaks and GUI
0%
Final Product
Finalize Project
15%
Presentation Preparation
Design Review
Final
10%

Spending Budget



Time Budget



Time Budget
500
450
400
350
300
250
200
150
100
50
0
Hours
August
September
October
November
December
January
February
Month
Expected Hours Worked
Actual Hours

Budget

- Items Purchased
  - USB to Quarter Inch Cable ($10)
- Items Available
  - Electric Guitar
  - Guitar Amplifier
  - Laptop
  - Pure Data Software

Summary

- To eliminate performance drawbacks posed by guitar stomp boxes
    - Created a simplified automatic guitar effects trigger system
        - Capable of triggering Delay Effect
    - Plan on implementing Dynamic Time Warping Patch
    - Meeting Design Criteria
        - Trigger Latency of <= 1 second
        - Minimum Note Onset separation of 10 notes per second
        - Concurrent/Multiple Effects Triggering Events

Questions