Universidade de Brasília
Nome: José Pedro de Santana Neto **Matricula:** 09/0119355
Rasumo TCC 2
29 de março de 2015

1 Linha de Ciclos de Desenvolvimento do TCC

1.1 Ciclo PDCA-Scrum adaptado para o TCC

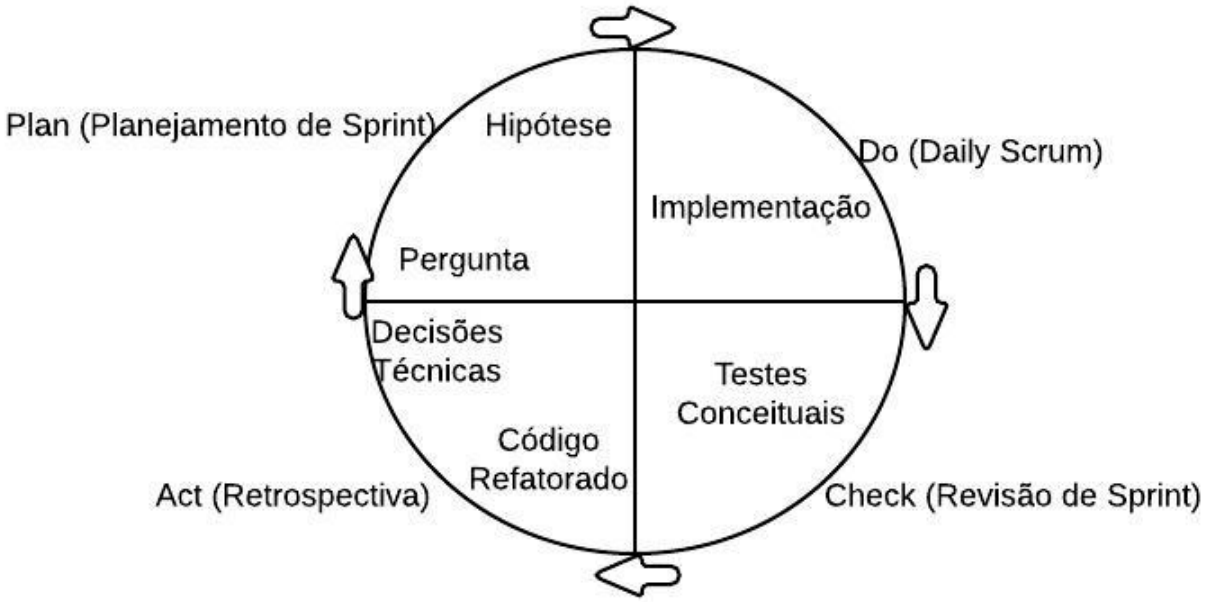

Figura 1: Modelo de Ciclo customizado para o TCC

Fases do ciclo de desenvolvimento:

- **Pergunta:** No início de cada ciclo (sprint adaptada) é feito uma pergunta a ser respondida que se adere aos objetivos do trabalho.
- **Hipótese:** A partir dessa pergunta é feita hipóteses que possam responder;
- **Implementação:** As hipóteses são pensadas, construidas e implementadas num script;
- **Testes Conceituais:** Cada hipótese implementada é testada conforme a teoria usada;
- **Código Refatorado e Decisões Técnicas:** Dado os resultados dos testes conceituais e código refatorado, é feitto uma avaliação do que foi produzido e decisões técnicas são tomadas.

1.2 Objetivos do TCC

- **Objetivo principal:** desenvolver uma solução computacional para reconhecimento de harmonias musicais
- **Objetivos específicos:**
  - Desenvolver solução computacional para reconhecimento de acordes e suas inversões;
  - Desenvolver solução computacional para reconhecimento de acordes ao longo do tempo;
  - Desenvolver solução computacional para extração do tom da música;

1.3 TCC 1 - Ciclo 1

- **Pergunta:** Para saber da harmonia da música é preciso saber as notas dela. Se cada nota é uma frequência de vibração sonora, como analisar o sinal no ponto de vista de frequências? É possível?
- **Hipótese:** A Transformada de Fourier pode construir o espectro de frequências do sinal.
- **Implementação:**

```
som = som(1:length(som));
som = som/max(som);

t = fft(som);
SINAL=sqrt(t.*conj(t));
SINAL=SINAL/max(SINAL);
```

- **Testes Conceituais:** Testes foram feitos para ver se os picos de frequência correspondem ao sinal de entrada. O resultado foi positivo e os picos representam a energia das frequências;
- **Código Refatorado e Decisões Técnicas:** A Transformada de Fourier, em específico a $FastFourierTransform$, realmente produz resultados satisfatórios em determinar o espectro de energia da frequências. Mas como configurar essas informações para localizar as notas musicais?

1.4 TCC 1 - Ciclo 2

- **Pergunta:** Como configurar essas informações para localizar as notas musicais?
- **Hipótese:** Dado que cada nota musical é um conjunto de frequências, realocar as energias frequenciais da transformada de fourier afim de que cada posição do vetor seje 1 unidedade de frequência (Hz) pode mapear a energia de cada nota.

- **Implementação:**

```
fs = 44100;
f = (0:length(som)-1)*fs/length(som);
freq = f(1:round(length(f)/2));
SOM = abs(fft(som));
SOM = SOM/max(SOM);
SOM = SOM(1:round(length(f)/2));
l = 1;
j = 0;
i = 1;
SOMA = 0;
while (i<length(freq))
    if (round(freq(i)) == round(freq(i+1)))
        SOMA = SOM(i+1) + SOMA;
        j = j + 1;
    else
        respfreq(l) = SOMA/(j+1);
        j = 0;
        SOMA = SOM(i+1);
        l = l+1;
    end
    i = i+1;
end
l = 0; j = 0; i = 0;
```

- **Testes Conceituais:** De fato as notas musicais foram localizadas com mais facilidade em determinados grupos de frequências. Tal ordenamento de frequências resultou num vetor de 22050 posições, independentemente do tamanho da amostra.

- **Código Refatorado e Decisões Técnicas:** A estratégia de realocar as energias decimais das frequências numa unidade de frequência se adere corretamente ao objetivo de encontrar as notas musicais. Mas como associar as notas musicais com as frequências?

1.5 TCC 1 - Ciclo 3

- **Pergunta:** Dado um conjunto de frequências, como associar essas as notas musicais?

- **Hipótese:** Visto que associar as frequências as notas musicas é uma tarefa muito complexa para uma solução determinística, uma rede neural de aprendizado não supervisionado do tipo *ProbabilisticNeuralNetwork* (PNN) pode classificar um conjunto de frequências em sua respectiva nota musical.

- **Implementação:**

```
%BASE DE DADOS DE NOTAS MUSICAIS DA REDE NEURAL
%NOTAS

notas(12,22050) = 0; %matriz das notas

```

```
%Do grave
notas(1,61) = 0.1;
notas(1,62) = 0.2;
notas(1,63) = 0.4;
notas(1,64) = 0.6;
notas(1,65) = 0.8;
notas(1,66) = 1;
notas(1,67) = 0.8;
notas(1,68) = 0.6;
notas(1,69) = 0.4;
notas(1,70) = 0.2;
notas(1,71) = 0.1;

.
.
.
%(769 LINHAS DE NOTAS)

%RADIAL BASIS LAYER para BD notas

i = 1; %contador para andar ao longo do vetor
b = 0.15; %sensibilidade da rede


while (i <= 12)

     %S1(i) = exp(-(norm(rfeq - notas(i,:))*b));
     %correlacao = corrcoef(rfeq,notas(i,:));
     %S1(i) = correlacao(1,2);
     S1(i) = sum(abs(rfeq.* notas(i,:)));


    i = i + 1;
end
```

- **Testes Conceituais:** Foram testadas 3 funções de transferência do neurônio. A primeira função de transferência - a exponencial da subtração dos valores - não foi muito eficaz pois para notas adjacentes as mesmas eram confundidas pela rede. Esse fato se dá pelo retorno de baixa magnitude da subtração de valores. A segunda função de transferência - a correlação dos valores - foi bastante eficaz para caracterizar notas. Porém a operação de subtração da média faz com que a energia final seja baixa, além de requisitar mais operações. A terceira função de transferência - a multiplicação dos valores - foi bastante eficaz para caracterizar notas e é rápida pois é uma forma simples da segunda função de transferência.

- **Código Refatorado e Decisões Técnicas:** A rede neural PNN foi bastante eficaz em classificar as frequências em termos de notas musicais. Porém como adicionar as próximas camadas da rede para determinação dos acordes?

1.6 TCC 1 - Ciclo 4

- **Pergunta:** Como adicionar as próximas camadas da rede para determinação dos acordes?

- **Hipótese:** Para poder mapear as notas é preciso adicionar mais 2 camadas. Uma camada para classificação de acordes, dado um conjunto de notas musicais e a outra para classificação de um acorde dado os conjuntos possíveis de acordes.

- **Implementação:**

```
% BASE DE DADOS PARA ACORDES

%------------------------------
BD(12,48) = 0;
%------------------------------

afin1 = 0; afin2 = 0;

%C
%CM
BD(12,1) = afin1;
BD(1,1) = 1; %baixo
BD(2,1) = afin2;
BD(4,1) = afin1;
BD(5,1) = 1; %terca
BD(6,1) = afin2;
BD(7,1) = afin1;
BD(8,1) = 1; %quinta
BD(9,1) = afin2;

.
.
.
%(430 LINHAS DE ACORDES)

%RADIAL BASIS LAYER para BD notas

while (i <= 48)
    S2(i) = sum(abs(S1.* BD(i,:)));
    i = i + 1;
end
```

- **Testes Conceituais:** Foram testadas todas as possibilidades de acordes e os que não foram evetivamente reconhecido foram as inversões e os acordes aumentados.

- **Código Refatorado e Decisões Técnicas:** De certo o acerto não foi total pois falta implementar uma camada que reconheça inversões.

1.7 TCC 2 - Ciclo 5

- **Pergunta:** Como reconhecer acordes no tempo de tal forma a saber onde eles ocorrem?
- **Hipótese:** Uma solução de reconhecimento de energias ao longo do tempo pode ser eficaz para a determinação do rítmo. Em tese é calcular a energia do sinal e aplicar um filtro passa-baixas para identificação dos picos de energia.
- **Implementação:**

```
%filtering the pulses of minor energy
signal_filtered = filter_signal(bpm_music);
signal_pulses = signal_filtered;

% Building array with means movies
signal_pulses = decrease_resolution(signal_filtered, file.fs, 1000);

% Beginnnig the correlation
array_correlation = correlate_moments(signal_pulses);

% Graph with correlation done! Now I will calculate the number of
peaks
array_peaks = filter_peak(signal_pulses);

peaks = findpeaks(array_peaks);

number_of_peaks = length(peaks) + 1;
```

- **Testes Conceituais:** Para um caso específico a solução funcionou, porém os outros casos ela não se aderiu.
- **Código Refatorado e Decisões Técnicas:** De certo modo detectar onde acorde ocorre no sinal não agrega valor para o escopo desse trabalho pois os níveis de energia são muito variáveis e não há um padrão como para a detecção. Uma solução de leitura de todo o sinal agregaria então no trabalho em ver de ler o sinal por partes?

1.8 TCC 2 - Ciclo 6

- **Pergunta:** Como ler o sinal todo e ter a visilibilidade dele em tempo e frequência?
- **Hipótese:** Para se ter uma resolução completa do sinal em tempo em frequência é preciso de ter uma transformada que agregue esses dois aspectos. Poder testar isso com a transformada wavelets.
- **Implementação:**

```
function [signal, imin, imax, iterations, energy] = tree_iterator(
signal, mini, maxi, imin, imax, iterations, energy)

if iterations == 0
```

```
    if mini >= 1 && maxi <= 22050
      [signal, imin, imax, iterations, energy] = tree_iterator(signal,
  mini, maxi, 1, 22050, 1, 0);
    else
      imin = 1;
      imax = 22050;
      return;
    end
  elseif iterations > 0
    imean = (imax - imin)/2 + imin;
    %Low
    if mini >= imin && maxi <= imean
      [h0, h1] = wfilters('bior6.8');
      [signal, y1] = decomposition_1level_qmf(h0, h1, signal);
      energy = sum(abs(signal)) + energy;
      iterations = iterations + 1;
      imax = imean;
      [signal, imin, imax, iterations, energy] = tree_iterator(signal,
  mini, maxi, imin, imax, iterations, energy);
    %High
    elseif mini >= imean && maxi <= imax
      [h0, h1] = wfilters('bior6.8');
      [y0, signal] = decomposition_1level_qmf(h0, h1, signal);
      energy = sum(abs(signal)) + energy;
      iterations = iterations + 1;
      imin = imean;
      [signal, imin, imax, iterations, energy] = tree_iterator(signal,
  mini, maxi, imin, imax, iterations, energy);
    else
      return;
    end

  end
```

- **Testes Conceituais:** A solução falhou num teste muito simples. Ao submeter um sinal puro numa frequencia determinada e constante o banco de filtros wavelets distorcia o sinal, deslocando a fase do sinal para frequencias adjacentes da original.
- **Código Refatorado e Decisões Técnicas:** Dado a barreira técnica de deslocamento de fase do sinal, como achar uma outra solução de resolução tempo-frequência?

1.9 TCC 2 - Ciclo 7

- **Pergunta:** Como ler o sinal todo e ter a visilibilidade dele em tempo e frequência?
- **Hipótese:** Para se ter uma resolução completa do sinal em tempo em frequência é preciso de ter uma transformada que agregue esses dois aspectos. Poder testar isso com a $ShortFourierTransform$ (Transformada de Fourier Janelada).
- **Implementação:**

```
function [notes_time, chords_time, chord_pitch] = DA3(signal, fs)

load_notes;
load_chords;

dictionary_chords = { 'C M', 'C m', 'C aum', 'C dim', ...
      'C# M', 'C# m', 'C# aum', 'C# dim', 'D M', 'D m', 'D aum', 'D dim',
    ...
      'Eb M', 'Eb m', 'Eb aum', 'Eb dim', 'E M', 'E m', 'E aum', 'E dim',
    ...
      'F M', 'F m', 'F aum', 'F dim', 'F# M', 'F# m', 'F# aum', 'F# dim',
    ...
      'G M', 'G m', 'G aum', 'G dim', 'G# M', 'G# m', 'G# aum', 'G# dim',
    ...
      'A M', 'A m', 'A aum', 'A dim', 'Bb M', 'Bb m', 'Bb aum', 'Bb dim',
    ...
      'B M', 'B m', 'B aum', 'B dim' };


% begin to analyse music
time_seconds_total = fix((length(signal)/fs));
notes_time(time_seconds_total, 60) = 0;
chords_time = {};
for time = 1:time_seconds_total

    % building a window to short fft
    signal_time = signal(1+((time-1)*fs):time*fs);
    window = blackman(length(signal_time));
    signal_time = window'.*signal_time;

    % make downsample to put frequency max in 1050 Hz
    signal_time = downsample(signal_time, 21);
    fs_time = fs/21;

    % get fourier transform
    module_fft = abs(fft(signal_time));
    respfreq(1:fs_time) = 0;
    window_mean = length(signal_time)/fs_time;
    for frequency = 1:fs_time
        respfreq(frequency) = sum(module_fft(1+((frequency-1)*window_mean
    ):frequency*window_mean))/window_mean;
    end

    respfreq = respfreq(1:fix(length(respfreq)/2));

    % get energy of notes
    for note = 1:60
        notes_time(time, note) = sum(respfreq.*notes(note,:));
    end

    % apply threshold

```

```
    % get energy of chords
    energy_chords(1:48) = 0;
    for chord = 1:48
        energy_chords(chord) = sum(notes_time(time, :).*chords(chord,:));
    end

    chords_time{time} = dictionary_chords{find(energy_chords==max(
    energy_chords))};

end

notes_energy_total = notes_time(1,:);
for time = 2:time_seconds_total
    notes_energy_total = notes_energy_total + notes_time(time,:);
end

energy_chords(1:48) = 0;
for chord = 1:48
    energy_chords(chord) = sum(notes_energy_total.*chords(chord,:));
end

chord_pitch = dictionary_chords{find(energy_chords==max(energy_chords))};
```

- **Testes Conceituais:** A solução da transformada de fourier janelada foi testada com acordes de violão e piano ao longo do tempo e o resultado foi satisfatório exceto para acordes de transição.
- **Código Refatorado e Decisões Técnicas:** A solução da transformada de fourier janelada se encaixou bem no conjunto.

1.10 TCC 2 - Ciclo 8

- **Pergunta:** Como extrair o tom da música?
- **Hipótese:** Para extrair o tom da música é preciso somar a energia das notas totais ao longo da música.
- **Implementação:**

```
dictionary_chords = { 'C', 'Cm', 'Caum', 'Cdim', ...
    'C#', 'C#m', 'C#aum', 'C#dim', 'D', 'Dm', 'Daum', 'Ddim', ...
    'Eb', 'Ebm', 'Ebaum', 'Ebdim', 'E', 'Em', 'Eaum', 'Edim', ...
    'F', 'Fm', 'Faum', 'Fdim', 'F#', 'F#m', 'F#aum', 'F#dim', ...
    'G', 'Gm', 'Gaum', 'Gdim', 'G#', 'G#m', 'G#aum', 'G#dim', ...
    'A', 'Am', 'Aaum', 'Adim', 'Bb', 'Bbm', 'Bbaum', 'Bbdim', ...
    'B', 'Bm', 'Baum', 'Bdim' };

notes_energy_total(60) = 0;
for note = 1:60
```

```
    notes_energy_total(note) = sum([notes_time(:,note)]);
  end

  % discover tone music
  notes_energy_tone(12) = 0;
  for note = 1:12
    notes_energy_tone(note) = notes_energy_total(note) +
   notes_energy_total(note + 12) ...
     + notes_energy_total(note + 2*12) + notes_energy_total(note + 3*12)
    ...
       + notes_energy_total(note + 4*12);
  end

  % find chord tone
  load_chords_tone;
  chords_tone(48) = 0;
  for chord = 1:48
    chords_tone(chord) = sum((notes_energy_tone.*chords_tone_mask(:,
   chord)'.^2));
  end

  chord_pitch_number = find(chords_tone==max(chords_tone));
  chord_pitch = dictionary_chords{chord_pitch_number};
```

f

- **Testes Conceituais:** A solução foi testada com sequencia de acordes do violão e as vezes o tom não é o certo.
- **Código Refatorado e Decisões Técnicas:** A quantidade de energia está atrapalhando a extração do tom. O tom é definido como a frequência de aparecimento das notas ao longo da música. Como trabalhar com a frequencia de aparecimento das notas?

1.11 TCC 2 - Ciclo 9

- **Pergunta:** Como trabalhar com a frequencia de aparecimento das notas?
- **Hipótese:** Se binarizar com 1 e 0 o mapa de notas no tempo a soma das notas será unitária, equivalente a frequência.
- **Implementação:**

```
% binarize set of notes
    for set = 1:5
        notes_time = set_of_notes_time{set};

        for time = 1:time_seconds_total
            for note = 1:60
                if notes_time(time, note) < max(max(notes_time))/180
                    notes_time(time, note) = 0;
                else
```

```
                    notes_time(time, note) = 1;
                end
            end
        end

        set_of_notes_time{set} = notes_time;
    end
```

- **Testes Conceituais:** A solução foi testada e verificada com picos somente de 1 e vales somente de 0.
- **Código Refatorado e Decisões Técnicas:** Com essa binarização o tom da música foi efetivamente corrigido.

1.12 TCC 2 - Ciclo 10

- **Pergunta:** Em relação aos acordes transitórios, como corrigir?
- **Hipótese:** Se ao deslocar a janela de tamanho de 1 segundo em passos de 0.2 segundos e calcular o espectro de frequência de cada passo pode-se fazer a média do acorde de cada tempo e poder cancelar os acordes transitórios.
- **Implementação:**

```
function set_of_windows_signals = build_window_short_fft(signal, time, fs
    )
  signal = [signal(:)];

    % part A
    time_start_A = round(1+((time-1)*fs));
    time_end_A = round(time*fs);
    signal_time_A = signal(time_start_A:time_end_A);
    window = blackman(length(signal_time_A));
    signal_time_A = window.*signal_time_A;

    % part B (displacement = + 0.2 seconds)
    time_start_B = round(1+((time-1)*fs+0.2*fs));
    time_end_B = round((time+0.2)*fs);
    if time_start_B < length(signal) && time_end_B <= length(signal)
        signal_time_B = signal(time_start_B:time_end_B);
        window = blackman(length(signal_time_B));
        signal_time_B = window.*signal_time_B;
    else
        signal_time_B(length(signal)) = 0;
    end

    % part C (displacement = + 0.4 seconds)
    time_start_C = round(1+((time-1)*fs+0.4*fs));
    time_end_C = round((time+0.4)*fs);
```

```
    if time_start_C < length(signal) && time_end_C <= length(signal)
        signal_time_C = signal(time_start_C:time_end_C);
        window = blackman(length(signal_time_C));
        signal_time_C = window.*signal_time_C;
    else
        signal_time_C(length(signal)) = 0;
    end

    % part D (displacement = + 0.6 seconds)
    time_start_D = round(1+((time-1)*fs+0.6*fs));
    time_end_D = round((time+0.6)*fs);
    if time_start_D < length(signal) && time_end_D <= length(signal)
        signal_time_D = signal(time_start_D:time_end_D);
        window = blackman(length(signal_time_D));
        signal_time_D = window.*signal_time_D;
    else
        signal_time_D(length(signal)) = 0;
    end

    % part E (displacement = + 0.8 seconds)
    time_start_E = round(1+((time-1)*fs+0.8*fs));
    time_end_E = round((time+0.8)*fs);
    if time_start_E < length(signal) && time_end_E <= length(signal)
        signal_time_E = signal(time_start_E:time_end_E);
        window = blackman(length(signal_time_E));
        signal_time_E = window.*signal_time_E;
    else
        signal_time_E(length(signal)) = 0;
    end

    set_of_windows_signals = {};
    if length(signal_time_A) == length(signal_time_B) && ...
        length(signal_time_A) == length(signal_time_C) && ...
         length(signal_time_A) == length(signal_time_D) &&  ...
          length(signal_time_A) == length(signal_time_E)
        set_of_windows_signals{1} = signal_time_A;
        set_of_windows_signals{2} = signal_time_B;
        set_of_windows_signals{3} = signal_time_C;
        set_of_windows_signals{4} = signal_time_D;
        set_of_windows_signals{5} = signal_time_E;
    else
        set_of_windows_signals{1} = signal_time_A;
        set_of_windows_signals{2} = signal_time_A;
        set_of_windows_signals{3} = signal_time_A;
        set_of_windows_signals{4} = signal_time_A;
        set_of_windows_signals{5} = signal_time_A;
    end
```

- **Testes Conceituais:** Os testes foram feitos em series de acordes tocados no violão e de fato os acordes transitórios desapareceram.
- **Código Refatorado e Decisões Técnicas:** Com essa correção os acordes maiores,

menores e diminutos estão sendo reconhecidos corretamente.

1.13 TCC 2 - Ciclo 11

- **Pergunta:** Como extrair a nota mais grave (baixo) de cada período do tempo?
- **Hipótese:** Com o mapa de ntoas binarizado é possível extrair o baixo pegando a primeira posição com valor 1 de cada tempo.
- **Implementação:**

```
function bass_time = get_bass(set_of_notes_time)

  notes_time_A = set_of_notes_time{1};
  notes_time_B = set_of_notes_time{2};
  notes_time_C = set_of_notes_time{3};
  notes_time_D = set_of_notes_time{4};
  notes_time_E = set_of_notes_time{5};

  total_seconds = length(notes_time_A(:,1));
  notes_time(total_seconds, 60) = 0;
  for time = 1:total_seconds
    for note = 1:60
      notes_to_analyse = [notes_time_A(time, note) notes_time_B(time,
   note) ...
     notes_time_C(time, note) notes_time_D(time, note) notes_time_E(time,
    note)];
      notes_time(time, note) = mode(notes_to_analyse);
    end
  end

    bass_time(1:total_seconds) = 0;
  for time = 1:total_seconds
    maxs = find(notes_time(time,:)==max(notes_time(time,:)));
    bass_time(time) = maxs(1);
  end

  for bass = 1:length(bass_time)
    bass_time(bass) = mod(bass_time(bass) - 1, 12) + 1;
  end

end
```

- **Testes Conceituais:** Os testes foram feitos com acordes e de fato ele reconhece as notas mais graves.
- **Código Refatorado e Decisões Técnicas:** Dado os baixos definidos, como incluir os acordes com inversões e aumentados?

1.14 TCC 2 - Ciclo 12

- **Pergunta:** Como extrair a nota mais grave (baixo) de cada período do tempo?
- **Hipótese:** Com o mapa de ntoas binarizado é possível extrair o baixo pegando a primeira posição com valor 1 de cada tempo.
- **Implementação:**

```
function chords_with_bass = get_chords_bass(chords_number, bass_time)

dictionary_chords = { 'C', 'C', 'C', 'C', 'C/E', 'C', 'C', 'C/G', 'C', 'C
    ', 'C', 'C', ...
'Cm', 'Cm', 'Cm', 'Cm/Eb', 'Cm', 'Cm', 'Cm', 'Cm/G', 'Cm', 'Cm', 'Cm', '
    Cm', ...
'Caum', 'Caum', 'Caum', 'Caum', 'Eaum', 'Caum', 'Caum', 'Caum', 'G#aum',
    'Caum', 'Caum', 'Caum', ...
'Cdim', 'Cdim', 'Cdim', 'Cdim/Eb', 'Cdim', 'Cdim', 'Cdim/F#', 'Cdim', '
    Cdim', 'Cdim', 'Cdim', 'Cdim', ...
'C#', 'C#', 'C#', 'C#', 'C#', 'C#/F', 'C#', 'C#', 'C#/G#', 'C#', 'C#', 'C
    #', ...
'C#m', 'C#m', 'C#m', 'C#m', 'C#m/E', 'C#m', 'C#m', 'C#m', 'C#m/G#', 'C#m'
    , 'C#m', 'C#m', ...
'C#aum', 'C#aum', 'C#aum', 'C#aum', 'C#aum', 'Faum', 'C#aum', 'C#aum', 'C
    #aum', 'Aaum', 'C#aum', 'C#aum', ...
'C#dim', 'C#dim', 'C#dim', 'C#dim', 'C#dim/E', 'C#dim', 'C#dim', 'C#dim/G
    ', 'C#dim', 'C#dim', 'C#dim', 'C#dim', ...
'D', 'D', 'D', 'D', 'D', 'D', 'D/F#', 'D', 'D', 'D/A', 'D', 'D', ...
'Dm', 'Dm', 'Dm', 'Dm', 'Dm', 'Dm/F', 'Dm', 'Dm', 'Dm', 'Dm/A', 'Dm', 'Dm
    ', ...
'Daum', 'Daum', 'Daum', 'Daum', 'Daum', 'Daum', 'F#aum', 'Daum', 'Daum',
    'Daum', 'Bbaum', 'Daum', ...
'Ddim', 'Ddim', 'Ddim', 'Ddim', 'Ddim', 'Ddim/F', 'Ddim', 'Ddim', 'Ddim/G
    #', 'Ddim', 'Ddim', 'Ddim', ...
'Eb', 'Eb', 'Eb', 'Eb', 'Eb', 'Eb', 'Eb', 'Eb/G', 'Eb', 'Eb', 'Eb/Bb', '
    Eb', ...
'Ebm', 'Ebm', 'Ebm', 'Ebm', 'Ebm', 'Ebm', 'Ebm/F#', 'Ebm', 'Ebm', 'Ebm',
    'Ebm/Bb', 'Ebm', ...
'Ebaum', 'Ebaum', 'Ebaum', 'Ebaum', 'Ebaum', 'Ebaum', 'Ebaum', 'Gaum', '
    Ebaum', 'Ebaum', 'Ebaum', 'Baum', ...
'Ebdim', 'Ebdim', 'Ebdim', 'Ebdim', 'Ebdim', 'Ebdim', 'Ebdim/F#', 'Ebdim'
    , 'Ebdim', 'Ebdim/B', 'Ebdim', 'Ebdim', ...
'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E/G#', 'E', 'E', 'E/B', ...
'Em', 'Em', 'Em', 'Em', 'Em', 'Em', 'Em', 'Em/G', 'Em', 'Em', 'Em', 'Em/B
    ', ...
'Caum', 'Eaum', 'Eaum', 'Eaum', 'Eaum', 'Eaum', 'Eaum', 'Eaum', 'G#aum',
    'Eaum', 'Eaum', 'Eaum', ...
'Edim', 'Edim', 'Edim', 'Edim', 'Edim', 'Edim', 'Edim', 'Edim/G', 'Edim',
    'Edim', 'Edim/Bb', 'Edim', ...
'F/C', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F/A', 'F', 'F', ...
'Fm/C', 'Fm', 'Fm', 'Fm', 'Fm', 'Fm', 'Fm', 'Fm', 'Fm/G#', 'Fm', 'Fm', '
    Fm', ...
'Faum', 'C#aum', 'Faum', 'Faum', 'Faum', 'Faum', 'Faum', 'Faum', 'Faum',
    'Aaum', 'Faum', 'Faum', ...
```

```
'Fdim', 'Fdim', 'Fdim', 'Fdim', 'Fdim', 'Fdim', 'Fdim', 'Fdim', 'Fdim/G#', 'Fdim', 'Fdim', 'Fdim/B', ...
'F#', 'F#/C#', 'F#', 'F#', 'F#', 'F#', 'F#', 'F#', 'F#', 'F#', 'F#/Bb', 'F#', ...
'F#m', 'F#m/C#', 'F#m', 'F#m', 'F#m', 'F#m', 'F#m', 'F#m', 'F#m', 'F#m/A', 'F#m', 'F#m', ...
'F#aum', 'F#aum', 'Daum', 'F#aum', 'F#aum', 'F#aum', 'F#aum', 'F#aum', 'F#aum', 'Bbaum', 'F#aum', ...
'F#dim/C', 'F#dim', 'F#dim', 'F#dim', 'F#dim', 'F#dim', 'F#dim', 'F#dim', 'F#dim', 'F#dim/A', 'F#dim', 'F#dim', ...
'G', 'G', 'G/D', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G/B', ...
'Gm', 'Gm', 'Gm/D', 'Gm', 'Gm', 'Gm', 'Gm', 'Gm', 'Gm', 'Gm', 'Gm/Bb', 'Gm', ...
'Gaum', 'Gaum', 'Gaum', 'Ebaum', 'Gaum', 'Gaum', 'Gaum', 'Gaum', 'Gaum', 'Gaum', 'Gaum', 'Baum', ...
'Gdim', 'Gdim/C#', 'Gdim', 'Gdim', 'Gdim', 'Gdim', 'Gdim', 'Gdim', 'Gdim', 'Gdim', 'Gdim/Bb', 'Gdim', ...
'G#/C', 'G#', 'G#', 'G#/Eb', 'G#', 'G#', 'G#', 'G#', 'G#', 'G#', 'G#', 'G#', ...
'G#m', 'G#m', 'G#m', 'G#m/Eb', 'G#m', 'G#m', 'G#m', 'G#m', 'G#m', 'G#m', 'G#m/B', ...
'Caum', 'G#aum', 'G#aum', 'G#aum', 'Eaum', 'G#aum', 'G#aum', 'G#aum', 'G#aum', 'G#aum', 'G#aum', ...
'G#dim', 'G#dim', 'G#dim/D', 'G#dim', 'G#dim', 'G#dim', 'G#dim', 'G#dim', 'G#dim', 'G#dim', 'G#dim/B', ...
'A', 'A/C#', 'A', 'A', 'A/E', 'A', 'A', 'A', 'A', 'A', 'A', ...
'Am/C', 'Am', 'Am', 'Am', 'Am/E', 'Am', 'Am', 'Am', 'Am', 'Am', 'Am', 'Am', ...
'Aaum', 'C#aum', 'Aaum', 'Aaum', 'Aaum', 'Faum', 'Aaum', 'Aaum', 'Aaum', 'Aaum', 'Aaum', 'Aaum', ...
'Adim/C', 'Adim', 'Adim', 'Adim/Eb', 'Adim', 'Adim', 'Adim', 'Adim', 'Adim', 'Adim', 'Adim', 'Adim', ...
'Bb', 'Bb', 'Bb/D', 'Bb', 'Bb', 'Bb/F', 'Bb', 'Bb', 'Bb', 'Bb', 'Bb', 'Bb', ...
'Bbm', 'Bbm/C#', 'Bbm', 'Bbm', 'Bbm', 'Bbm/F', 'Bbm', 'Bbm', 'Bbm', 'Bbm', 'Bbm', 'Bbm', ...
'Bbaum', 'Bbaum', 'Daum', 'Bbaum', 'Bbaum', 'Bbaum', 'F#aum', 'Bbaum', 'Bbaum', 'Bbaum', 'Bbaum', 'Bbaum', ...
'Bbdim', 'Bbdim/C#', 'Bbdim', 'Bbdim', 'Bbdim/E', 'Bbdim', 'Bbdim', 'Bbdim', 'Bbdim', 'Bbdim', 'Bbdim', 'Bbdim', ...
'B', 'B', 'B', 'B/Eb', 'B', 'B', 'B/F#', 'B', 'B', 'B', 'B', 'B', ...
'Bm', 'Bm', 'Bm/D', 'Bm', 'Bm', 'Bm', 'Bm/F#', 'Bm', 'Bm', 'Bm', 'Bm', 'Bm', ...
'Baum', 'Baum', 'Baum', 'Ebaum', 'Baum', 'Baum', 'Baum', 'Gaum', 'Baum', 'Baum', 'Baum', 'Baum', ...
'Bdim', 'Bdim', 'Bdim/D', 'Bdim', 'Bdim', 'Bdim/F', 'Bdim', 'Bdim', 'Bdim', 'Bdim', 'Bdim', 'Bdim', ...
};


% build chords with bass to translate to dictionary
chords_with_bass_number = {};
chord_iterator = 1;
for chord = 1:48
```

```
    for bass = 1:12
59    chords_with_bass_number{chord_iterator} = [chord, bass];
      chord_iterator = chord_iterator + 1;
61  end
  end
63
  %-------------------------------------------------------------
65  chords_with_bass = {};
  for time = 1:length(chords_number)
67    for chord = 1:length(chords_with_bass_number)
      peer_chord = chords_with_bass_number{chord};
69    if peer_chord(1) == chords_number(time) && peer_chord(2) == bass_time
    (time)
        %fprintf('%d e %d\n',chords_number(time),bass_time(time));
71      chords_with_bass{time} = dictionary_chords{chord};
        %fprintf('%s e numero acorde: %d\n',chords_with_bass{time}, chord);
73    end
    end
75  end

77 end
```

- **Testes Conceituais:** Todas as possibilidades de acordes foram reconhecidos com sucesso.
- **Código Refatorado e Decisões Técnicas:** Os objetivos do trabalho foram alcançados com sucesso.