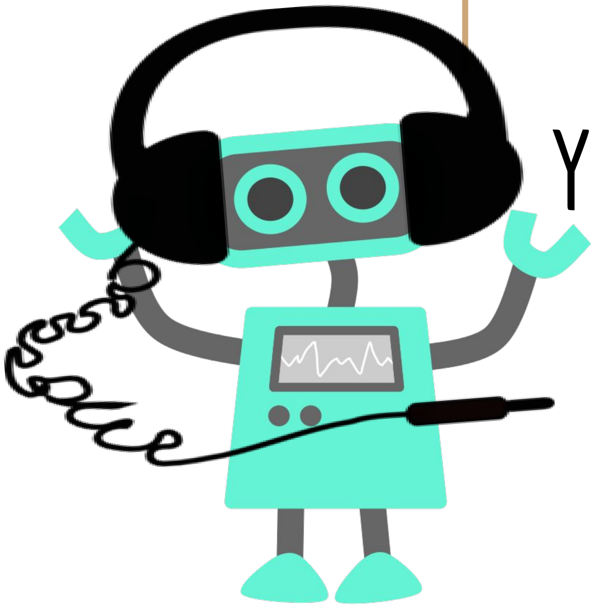


Music Mashup: Your In-House DJ

Medha Shrivastava
Tapan Bohra

Group 7

Idea

Problem Statement

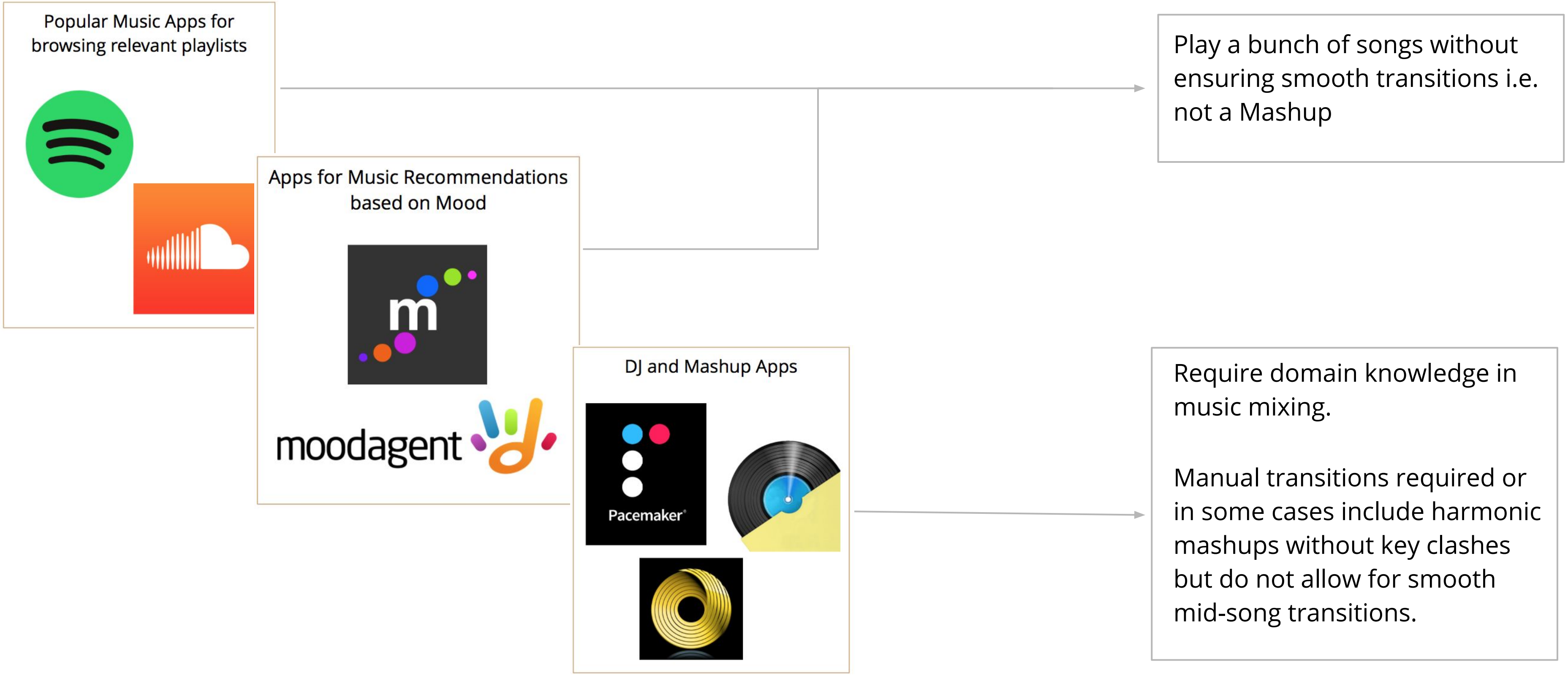
Creating a Music Mashup Application which allows for smooth ***mid-song transitions*** while ensuring ***lyrical and emotional relevance*** tailored to your personal preferences aka your own “in-house DJ”.

Motivation

- Although apps like Spotify, SoundCloud etc. provide great playlists for every mood and event, they *DO NOT* allow smooth mid-song transitions.

- Listening to one song for its entirety could get boring and selecting the correct time to transition *without cutting off lyrics* while keeping in mind the *tempo* could be very tricky.

Current Solutions



Popular Music Apps for browsing relevant playlists

Apps for Music Recommendations based on Mood

DJ and Mashup Apps

Play a bunch of songs without ensuring smooth transitions i.e. not a Mashup

Require domain knowledge in music mixing.

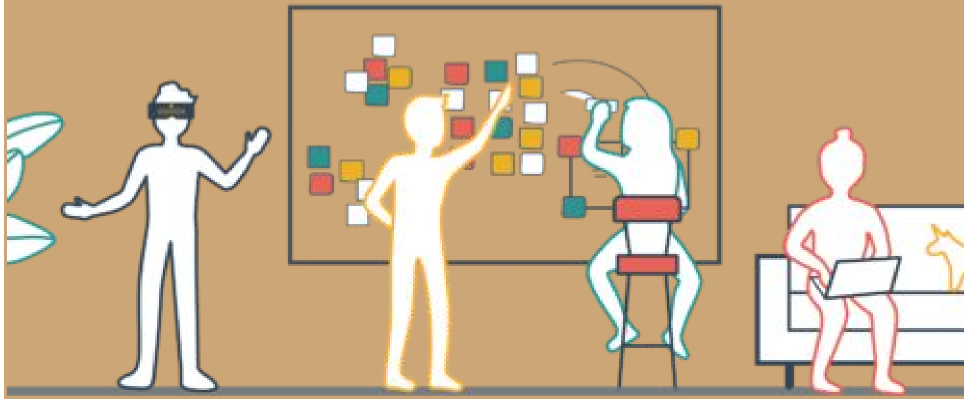
Manual transitions required or in some cases include harmonic mashups without key clashes but do not allow for smooth mid-song transitions.

Related Academic Work

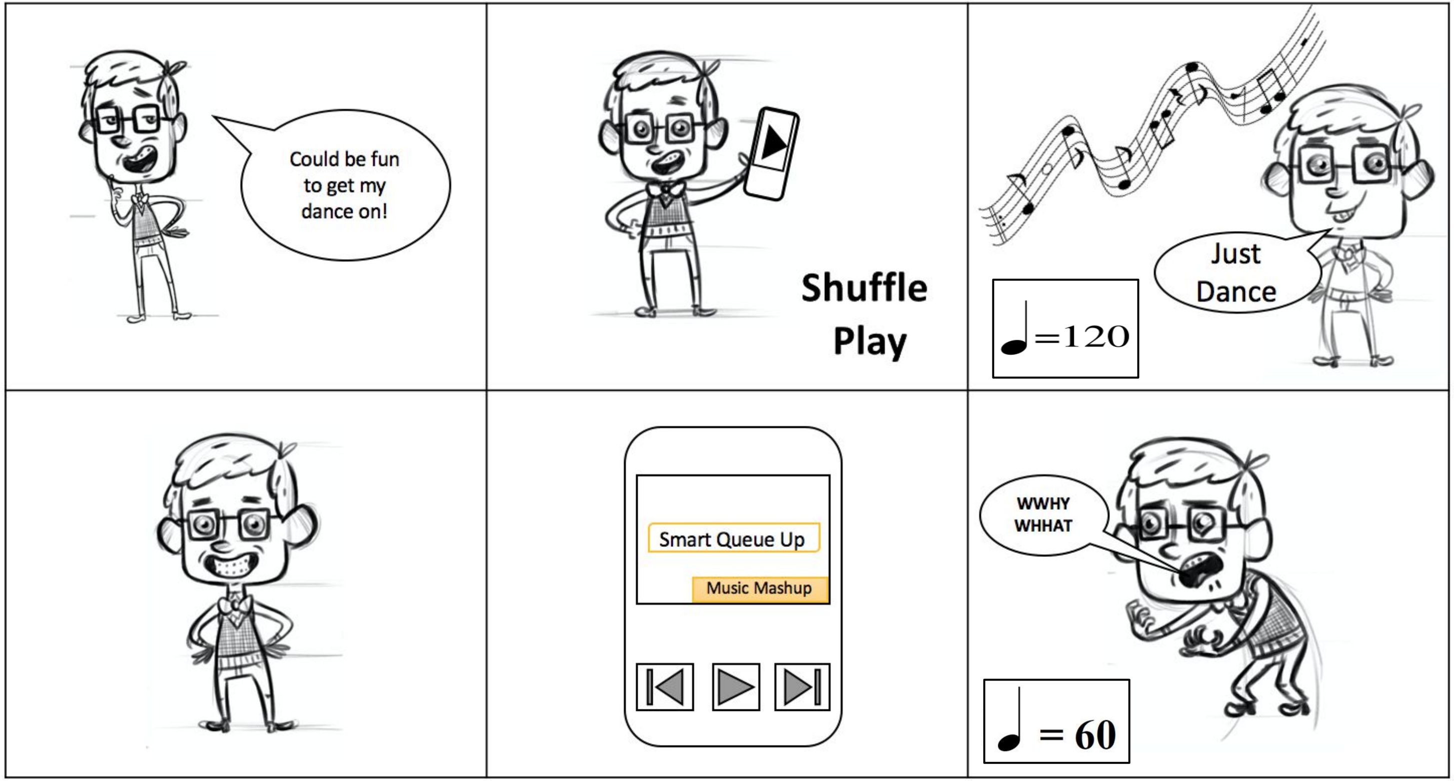
- **Music Emotion Classification**
  - Multilabel classification task
  - Music highlight detection using energy information
  - Targeting music segments: determines how likely the song segment belongs to an emotion class

- **Smooth Transitions**
  - Lyrical Analysis
  - Chord Recognition: matching chords
  - Beat-Matching Technology: matches the tempo and the phase to match the beats

User Scenarios



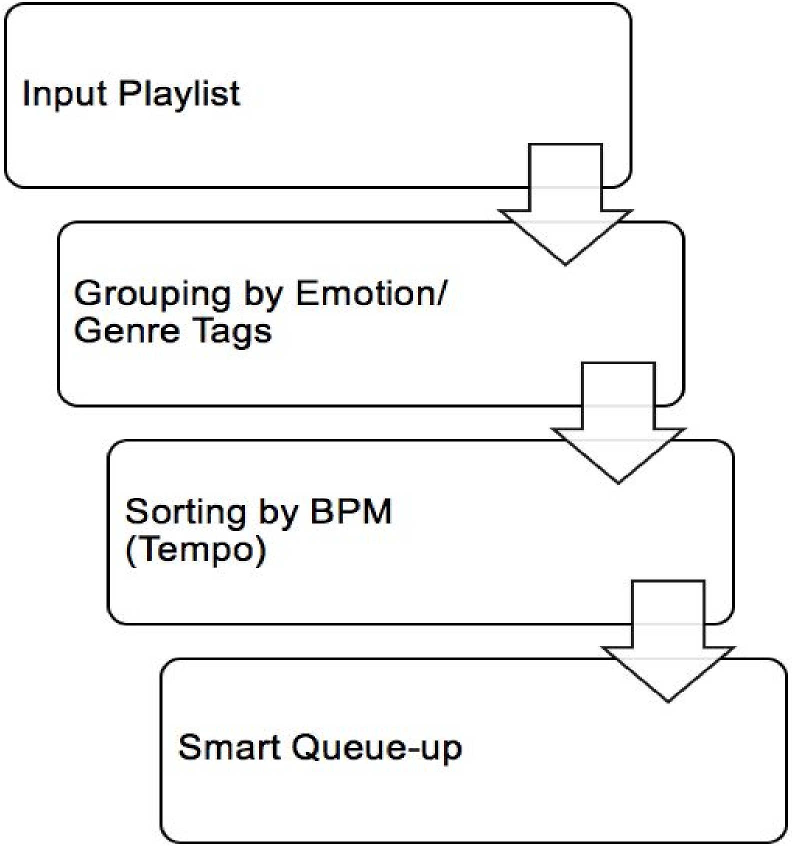
User Scenario - 1



Use Case - 1

I. Smart Queue-up

- Allows queueing up of a playlist using BPM and Emotion Analysis.
- First groups together relevant songs based on the genre and mood.
- Then relatively queues up similarly grouped songs based on tempo.

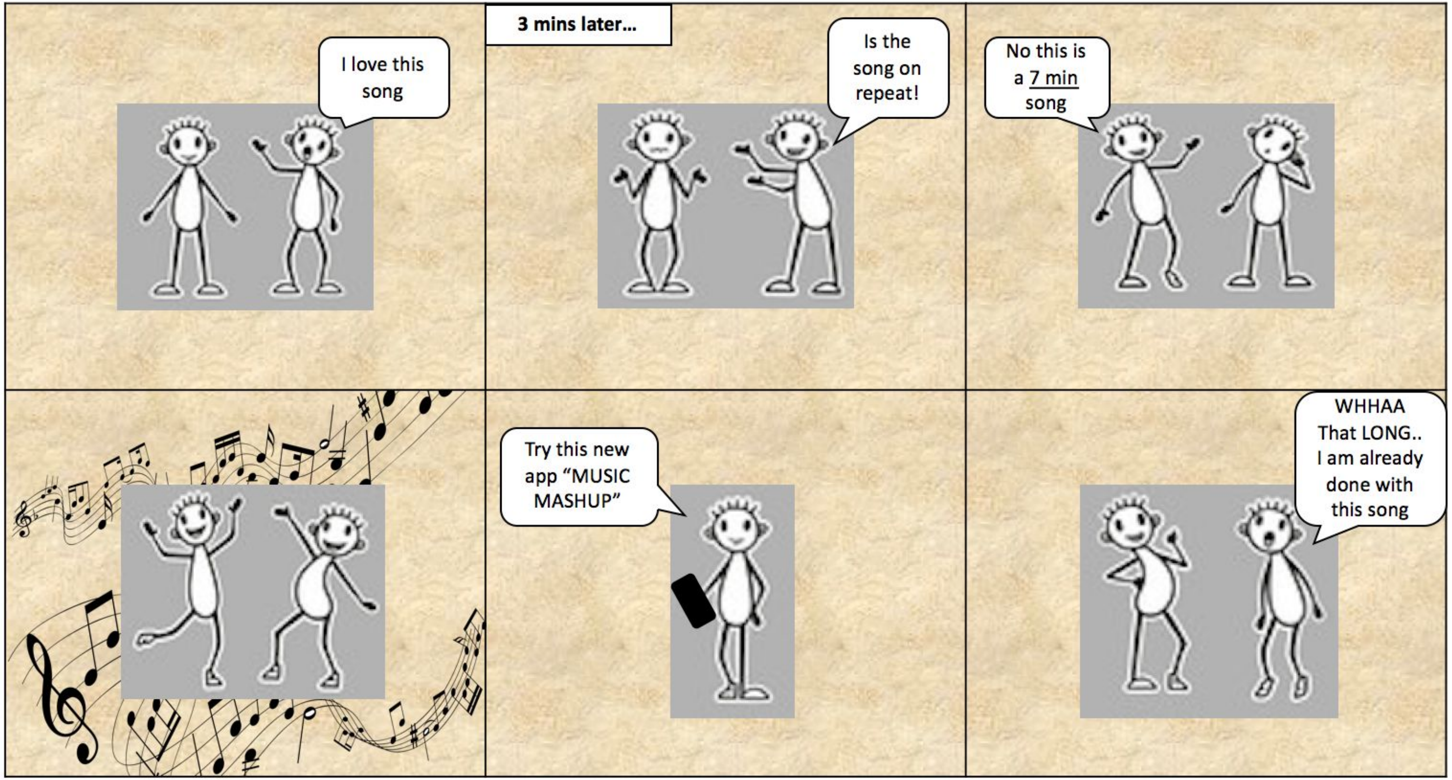

Input Playlist

Grouping by Emotion/ Genre Tags

Sorting by BPM (Tempo)

Smart Queue-up

User Scenario - 2

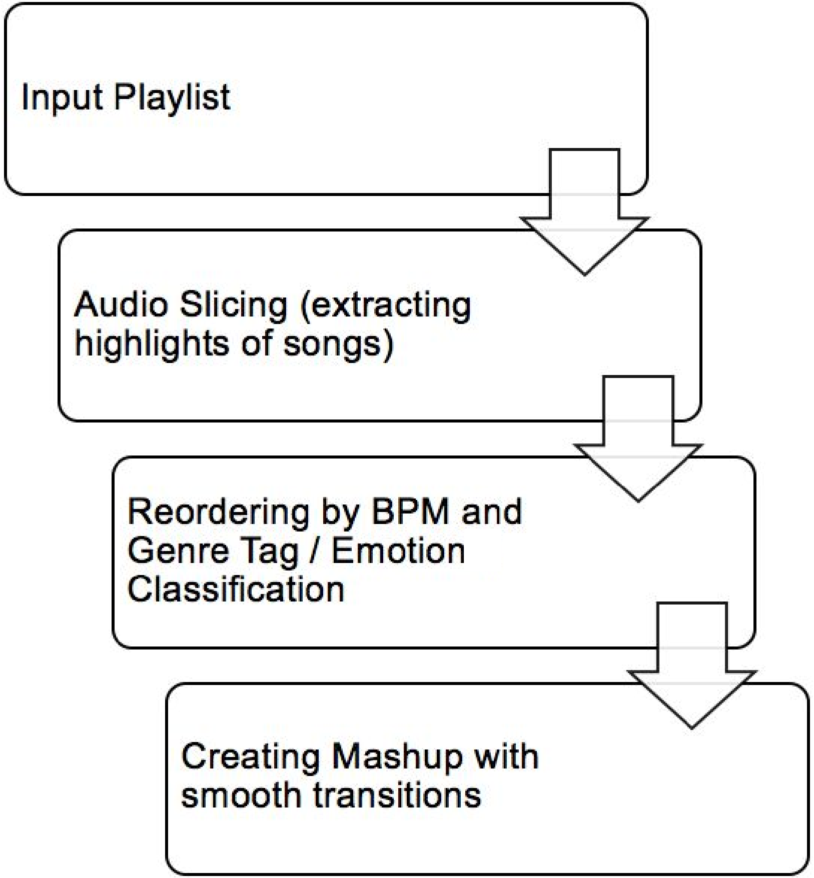

I love this song
3 mins later...
Is the song on repeat!
No this is a 7 min song
Try this new app "MUSIC MASHUP"
WHHAA That LONG.. I am already done with this song

Use Case - 2

II. Smooth transitions & Mashups

- When bored of listening to a song in its entirety the application allows creating a mashup by selecting the highlights of each song using lyrical analysis and repetition.

- Then it arranges them based on the tempo while ensuring that lyrics are not cut-off and transitions are smooth.



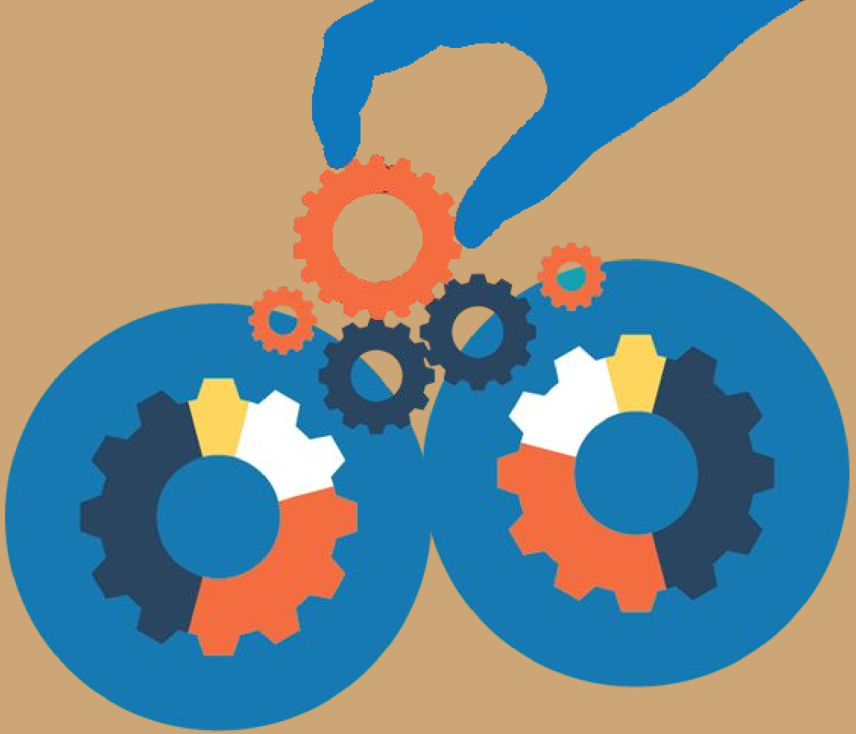
Input Playlist

Audio Slicing (extracting highlights of songs)

Reordering by BPM and Genre Tag / Emotion Classification

Creating Mashup with smooth transitions

Implementation



Core Features

- **Smart Queueing**: Grouping relevant songs based on tempo, genre tag and emotion analysis.

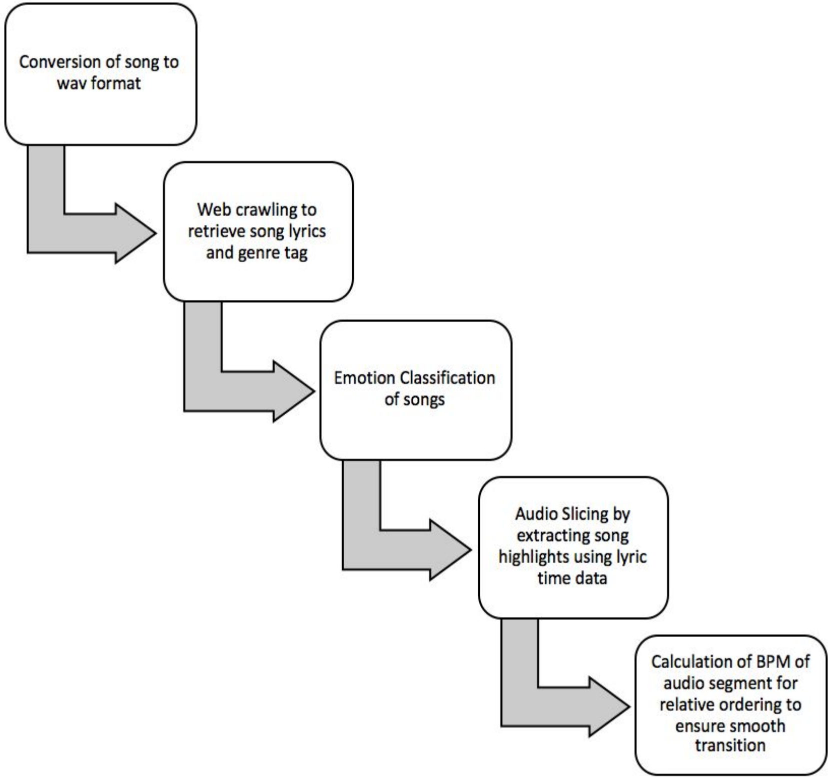
- **Identifying Highlights**: Selecting highlights of a song based on lyrical analysis and repetition to create audio slices.

- **Transitions**: Ensuring smooth transitions based on tempo.

- **Flexibility**: In case of multiple ordering options allows for refresh option which creates a unique mashup each time.

Workflow

- Read audio file and convert to wav format
- Retrieve the lyrics for a song by web crawling
- Use the genre tag/ emotion to group the songs
- Extract highlights of songs by lyrical analysis
- Calculate BPM for each frame of audio files and relatively order the audio slices using the avg. BPM to ensure smooth transitions
- Play the ordered audio slices using Pygame



Conversion of song to wav format

Web crawling to retrieve song lyrics and genre tag

Emotion Classification of songs

Audio Slicing by extracting song highlights using lyric time data

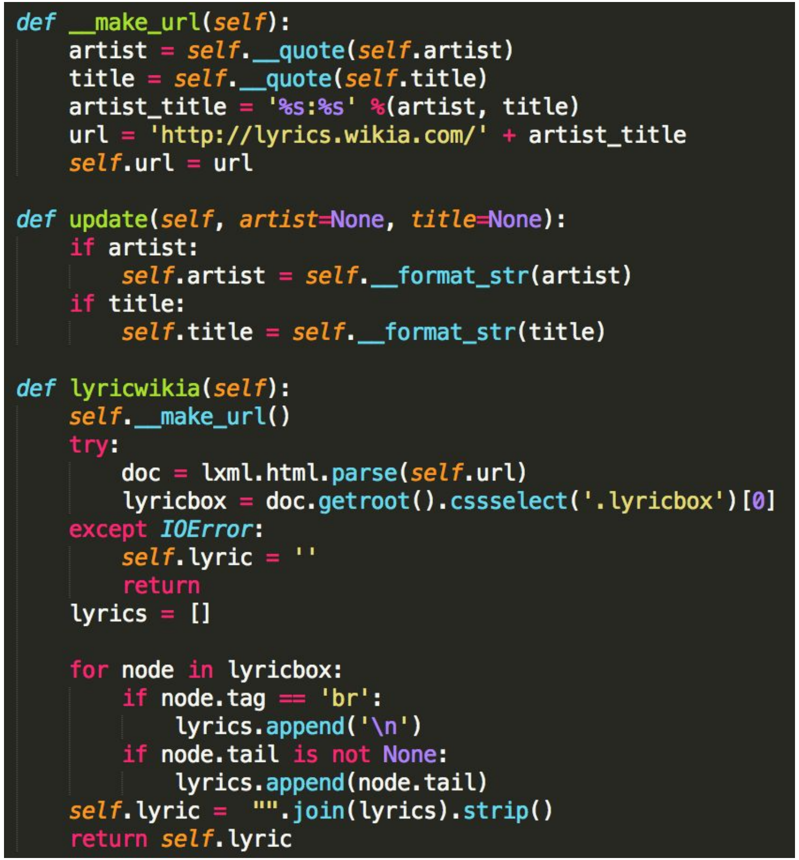
Calculation of BPM of audio segment for relative ordering to ensure smooth transition

Retrieving Lyrics

- Methods to retrieve lyrics include:
  - Web Crawling
  - Using APIs such as MetroLyrics
  - Using Million Song Dataset or other databases

- In our implementation we use Web Crawling to extract the required lyrics

- For this we crawl the website LyricWikia



```
def __make_url(self):
    artist = self.__quote(self.artist)
    title = self.__quote(self.title)
    artist_title = '%s:%s' %(artist, title)
    url = 'http://lyrics.wikia.com/' + artist_title
    self.url = url

def update(self, artist=None, title=None):
    if artist:
        self.artist = self.__format_str(artist)
    if title:
        self.title = self.__format_str(title)

def lyricwikia(self):
    self.__make_url()
    try:
        doc = lxml.html.parse(self.url)
        lyricbox = doc.getroot().cssselect('.lyricbox')[0]
    except IOError:
        self.lyric = ''
        return
    lyrics = []

    for node in lyricbox:
        if node.tag == 'br':
            lyrics.append('\n')
        if node.tail is not None:
            lyrics.append(node.tail)
    self.lyric =  "".join(lyrics).strip()
    return self.lyric
```

Music Classification

- Method for emotion analysis on music:
  - Fetching lyrics of the song and doing sentiment analysis of the lyrics using tools like IBM Watson Developer Cloud Alchemy Language and Tone Analyzer

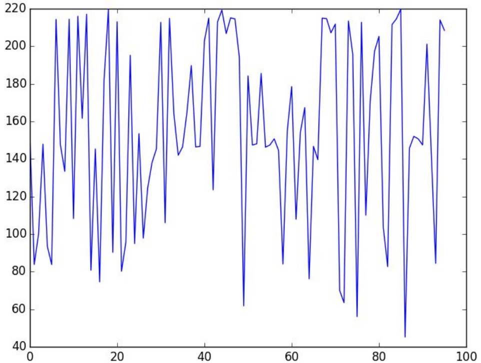
- Using Tagged Dataset including genre tags like pop, metal, rock etc:
  - Million Song Dataset
  - GTZAN Genre Collection
  - Spotify API, Echo Nest API

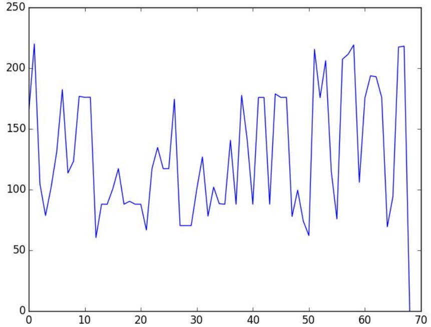
BPM Calculation

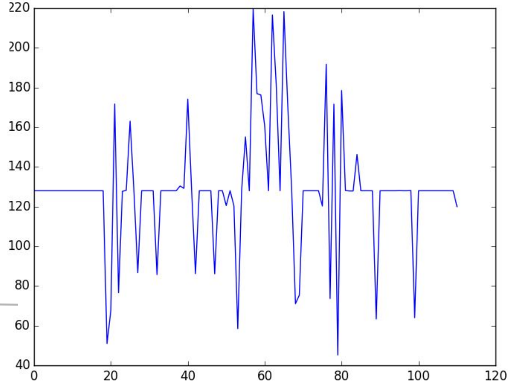
- **Beats per minute** (**bpm**) is used as a measure of tempo in music
- Methods to find the BPM:
  - Using tools like Audacity
  - Implementing a BPM calculator
- For our implementation we use a BPM detection algorithm as presented in the paper of G. Tzanetakis, G. Essl and P. Cook "Audio Analysis using the Discrete Wavelet Transform"
- This gives us the flexibility to find BPM per frame (since avg. BPM is not always indicative of tempo of song segment) for the selected highlights of song and order them accordingly to ensure smooth transitions

Experiment

Song	Average BPM
A Sky Full of Stars	126.166884379
Blank Space	127.974109122
Fake Plastic Trees	150.719642043 ✗
Fight Song	117.265254811 ✓
Fix You	144.367152652
Halo	117.515263107
Hey Jude	147.394965939
Levels	127.974109122 ✗







Implementation Frustrations:
Chorus Detection (Audio Slicing)

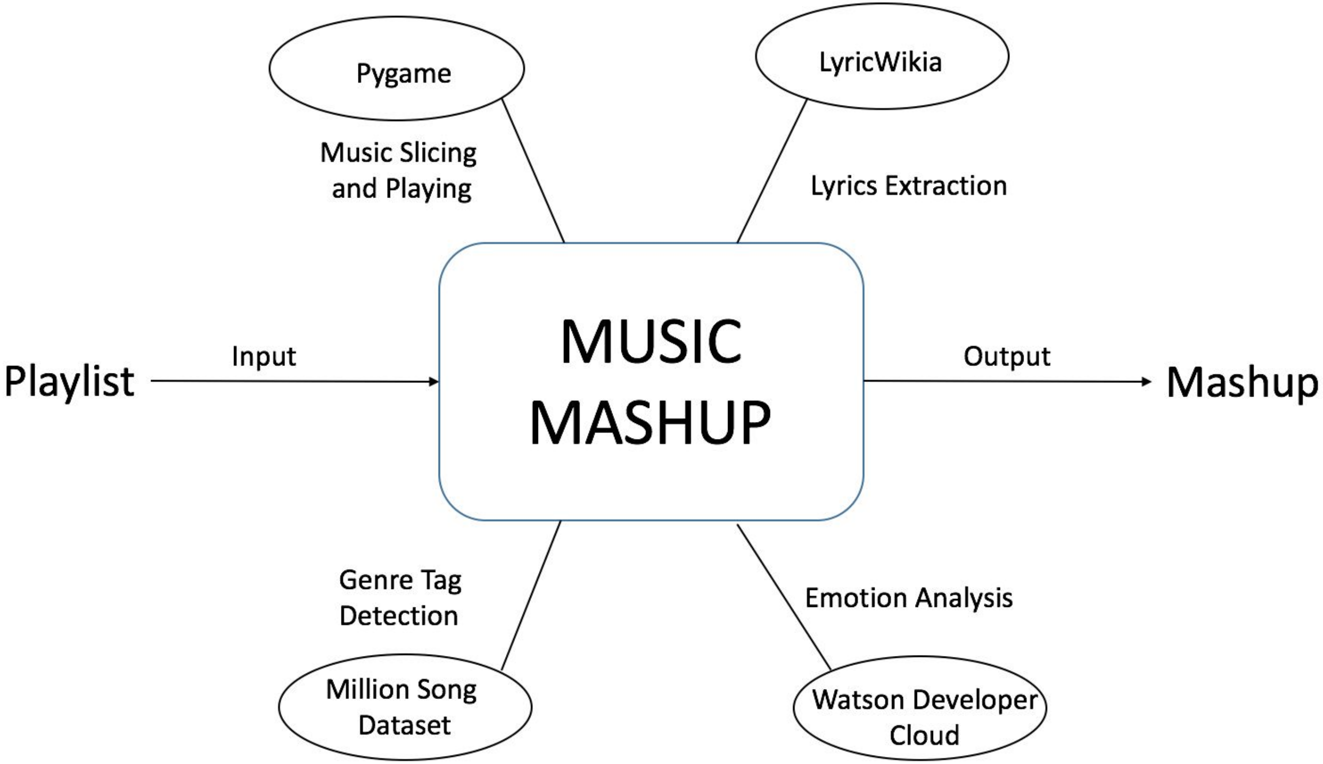
- Necessity for slicing audio files:
  - Address the original use case of mid-song transition
  - No pre-existing database that provides information on song highlights/ hooks
  - No concrete way to ensure lyrics are not cut-off

- Initial approach for chorus/ song highlight recognition:
  - Retrieving lyrics for desired song
  - Identifying chorus by finding the most frequently repeating set of lines

- **Bottleneck**: Engineering the start and end time using lyrics to slice the audio file for Mashup

- **Solution**: Using music lyrics timing data
  - Crawling for subtitles/ captions for music videos rather than lyrics since they have timestamps along with lyrics
  - Using Karaoke Datasets instead of Lyrics Dataset

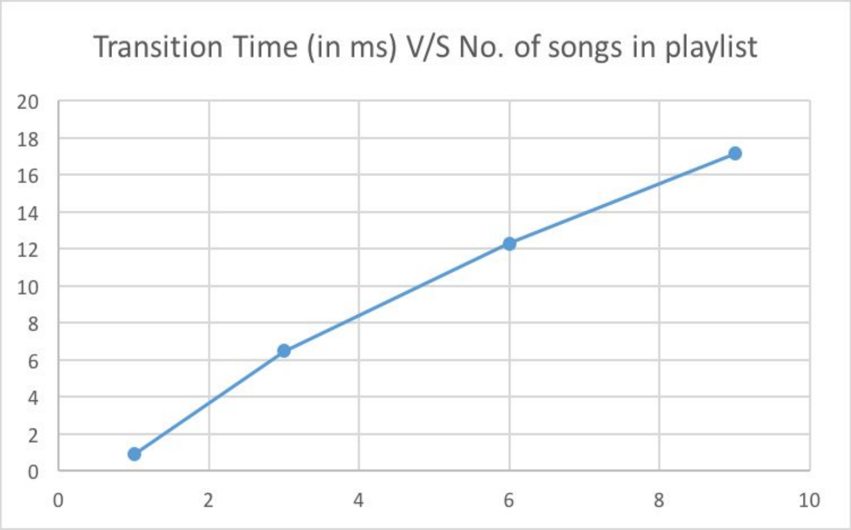
Architecture Overview



Performance

Plot is showing (Total Time Taken - Sum of actual length of songs) v/s the number of songs in the input playlist

No. of songs	Total Transition Time (in ms)
1	0.888109207
3	6.464004517
6	12.29310036
9	17.15087891



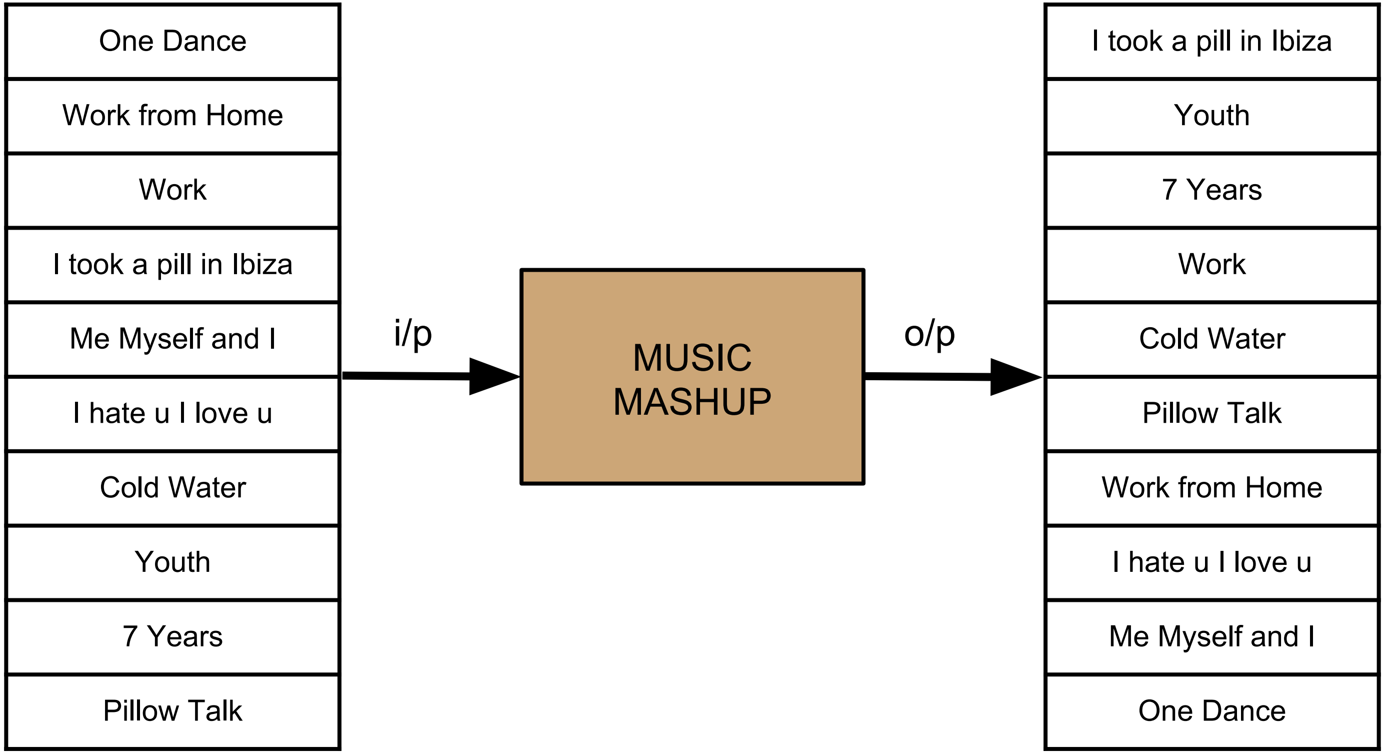
Testing and Evaluation

- Testing:
  - Response time: ensure that the engine is fast and efficient
  - Categorization: test the system across a number of songs incorporating various artists and different genres

- Evaluation:
  - Usability: by taking user feedback and surveys
  - Features: by comparing with existing popular mashups online

Demo

Result



One Dance
Work from Home
Work
I took a pill in Ibiza
Me Myself and I
I hate u I love u
Cold Water
Youth
7 Years
Pillow Talk

i/p

MUSIC MASHUP

o/p

I took a pill in Ibiza
Youth
7 Years
Work
Cold Water
Pillow Talk
Work from Home
I hate u I love u
Me Myself and I
One Dance

Thank You