Bilkent University

Department of Computer Engineering

Senior Design Project

Musync

Low-Level Design Report

Ahmet Çandıroğlu, Anıl Erken, Berk Mandıracıoğlu, Halil İbrahim Azak

Supervisor: Assoc. Prof. Dr. M. Mustafa Özdal
Jury Members: Prof. Dr. Özcan Öztürk, Prof. Dr. Cevdet Aykanat

Low-Level Design Report Feb 18, 2019

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project II course CS492.

Table of Content

1. Introduction

Music is a common interest that many people enjoy and rest their souls. It may become a cumbersome procedure to find music that many people enjoy. In our daily lives, we are surrounded with music by means of our environment such as restaurants, cafes, bars, etc. Therefore, it is even more important to create a suitable playlist to satisfy majority based on their collective music taste. It is also necessary to recommend new places that people might enjoy according to their music taste and even discover new songs.

The main purpose of this system is to let people choose what they listen according to their music taste in public places such as cafes and bars. Musync aims to facilitate creation of playlists that are dynamically modified by both analysing the music tastes of users and their feedback on the current playlist. Musync is basically an automated digital jukebox which collects data about people's music tastes from its current users and initialises a playlist. The playlist is dynamic and changes as people come on go so that everybody can listen what they generally like. Moreover, users are able to add songs to the playlist and have a chance to choose the next song by the power of bidding. Also, users are able to see nearby locations along with their music preferences, so they can choose where they want to hang out.

This report describes the low-level architecture and the design of the Musync. Report comprises Object design trade-offs, Engineering standards, Packages and Class interfaces sections. Finally, the report is concluded with the class diagrams and the detailed explanations of software components.

1.1 Object design trade-offs

1.1.1 Usability vs. Functionality

In Musync, our first goal is usability. Firstly, we want our users to start using it without any annoying, time consuming processes like downloading an app and registration. Then, we will provide a clean and simple menu that user will know how to use it at first sight, and fast navigation to make it easy to use. Therefore we will not implement too much functionality that will lead to complex, slow menus and make the application difficult to quickly understand. Even registration is not required, since we think that aim of our app is simple

and users don’t want to spend too much time for that. That’s why we want to keep it “as simple as possible, but not simpler”.

1.1.2 Efficiency vs. Accuracy

One of our goals it to make a common playlist for each place which contains songs from different users’ playlists. This procedure will be repeated for each place throughout the day and can be very difficult to compute when there are many users. Also, the playlist will be dynamic as it will be altered for each user. For these reasons, we aim to have an efficient algorithm instead of a highly accurate one.

1.1.3 Security vs. Usability

While high level of usability is one of our goals, we needed to lose some usability to provide more security. As in real life or any other field, being more secure may require sacrificing some conveniences. An example of this is ending sessions of users after a time of inactivity. We determined this duration to be not too long to prevent both an unintended person accessing to a user’s account on a device after the actual user stopped using it and also an ill intentional user continuing to affect the music flow of place long after leaving the place. This will require users to reconnect to the place if they stayed inactive long enough for their session to expire.

1.1.4 Reliability vs. Compatibility

Musync aims to provide a robust and reliable system where users can enjoy the music experience without any system crashes. Moreover, if the system is allowed to be compatible with more than one OS then the rate of failures and maintenance would increase and therefore the system could potentially be interrupted more than necessary. The ambition of Musync is to provide the optimal user experience and satisfaction, thus a reliable system has the utmost priority.

1.2 Interface documentation guidelines

In this report; all classes, attributes and methods are named in the camel case format. Class names start with a capital letter, while others start with lowercase. Class interface descriptions are given in the following format:

class ClassName
Explanation of the class
Attributes
typeOfAttribute nameOfAttribute
Methods
returnType methodName(parameters) Method explanation if necessary

1.3 Engineering standards

We have used the UML guidelines in this report for the class interfaces, diagrams, scenarios, subsystem compositions [1]. UML is a commonly used way to generate these diagrams, easy to use and since it is the method taught at Bilkent University, we chose to utilize it in the following pages. The report follows IEEE's standards for the citations [2]. Once again, this is a commonly used method and it is the preferred one in Bilkent University.

1.4 Definitions, acronyms, and abbreviations

API: Application Programming Interface.

MVC: Model View Controller architecture.

UI: User Interface.

2. Packages

2.1 Server

2.1.1 Model

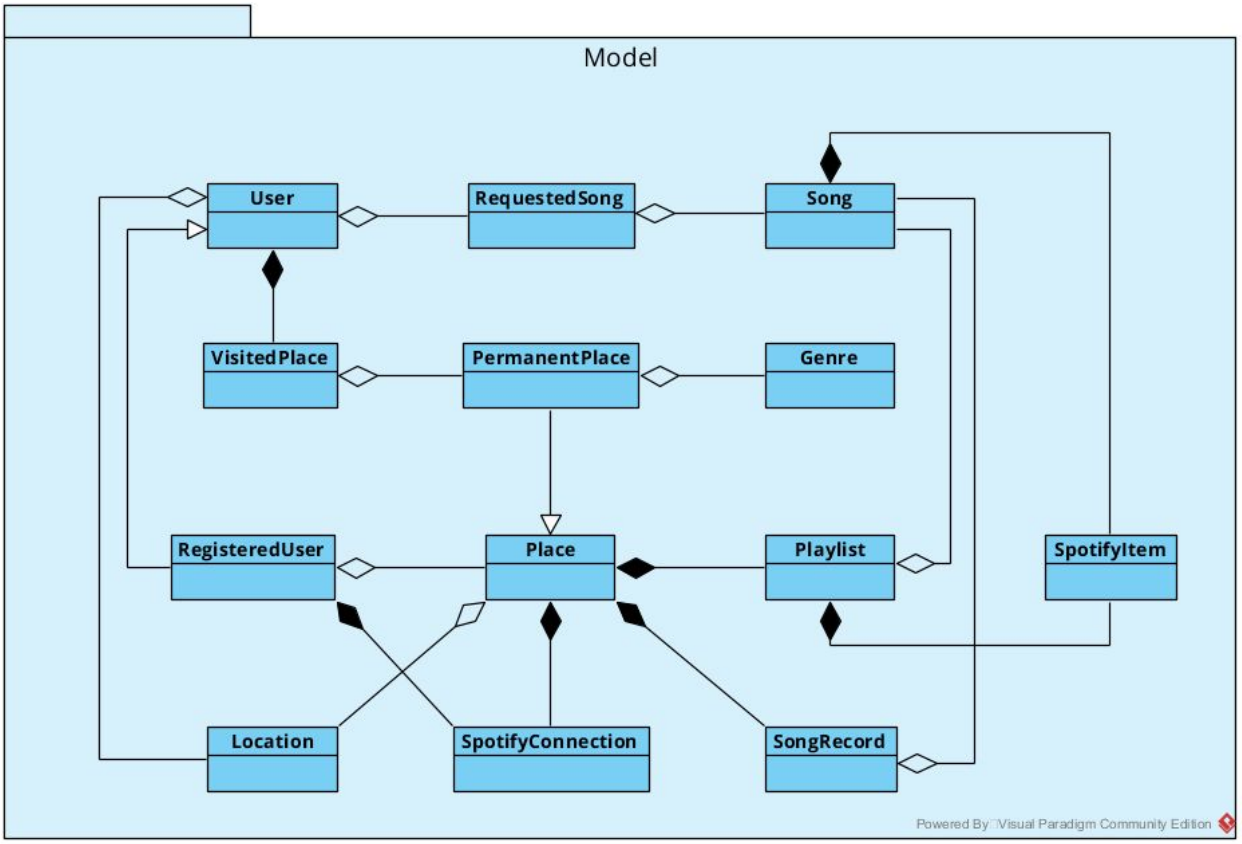

Figure 1: Model package

User: This class is the base class of all user types. It has the basic attributes such as password, id, location, last login date informations. Moreover, *User* has the points attribute in order to enable the visitor to bid on music. *User* has visited places and requested songs attributes in order to keep track of the visitor activity and increase points if the visitor is a regular visitor of the place. Aforementioned attributes serve as necessary data to make place recommendations to the user.

RegisteredUser: This class inherits *User* class. *Users* that registered through either email or Spotify will be a registered user. Registered users will be able to create and control places.

To allow this functionality, *RegisteredUser* stores user information such as email and Spotify auth token, and place related information such as owned places and premium account information.

Place: This class represents places. A place in the context of this report is a point on earth that has a shared list of songs that people can participate in controlling if they are in required proximity of the point. Musync allows both commercial place owners and individual users to create such a place and share their list. A *Place* needs a name, a location represented by *Location* class and a shared playlist represented by *Playlist* class. This class also stores other data like pin to connect, votes for songs, records of previously played songs to provide functionalities Musync offers.

PermanentPlace: This class inherits the *Place* class. Owners of commercial places can create permanent places that can provide service continuingly. Also a permanent place can have a list of allowed music genres to both guide customers on what to expect and restrict the type of music requested by customers.

VisitedPlace: This class represents a record of a user visiting a place. It stores the place, the user and the date of visit to provide users with data like their previous visits.

Song: This class represents the songs as an entity in the entire system. It has the basic attributes (duration, genre, name, etc.) and the functionalities to make the system work and play music. Whenever a song is searched, added or requested to the system an instance of this class will be created.

RequestedSong: This class is basically a different representation of a *Song* instance such that it has a date, a place and user associated with it. It is the most essential entity of the system. Whenever a user requests a song, an instance of this class is initiated and assigned to a playlist. Later, users can bid on requested songs to play them as next *Song* which is *SongRecord* class.

SongRecord: This class represents songs that are played in a place. An instance of this class is initiated whenever a novel song is played in the playlist of a place. This class keeps track of the statistical information about songs so that when a user views a place they can see the genres of songs played in a place and trending songs.

SpotifyItem: This class stores data corresponding to Spotify counterparts of items like songs and playlists.

SpotifyConnection: This class stores Spotify connection related data.

Playlist: This class represents Spotify playlist of the *Place*. Requested songs will be added to the instance of this class and next songs will be selected from those songs.

Genre: This class represents genre of music and will be compatible with genres on Spotify (e.g. Metal, Rock). This class is used to determine genres of *Songs* and *Places.*

Location: This class represents physical location of *Places* and *Users.* This class will be used to determine in which *Place*, *Users* are. Also, this class will be compatible with *Google Maps* as we will be using it as main location data source.

2.1.2 Controller

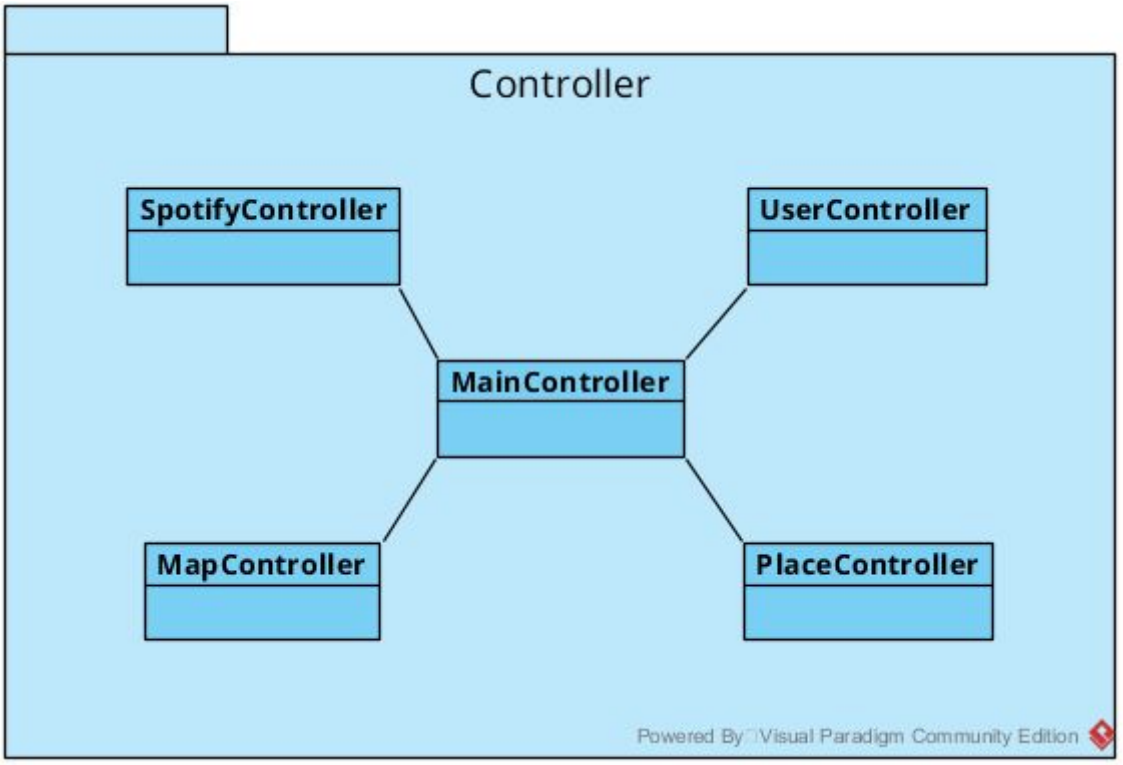

Figure 2: Controller package

MainController : This class is responsible for managing the input received from the user and collect necessary information to be displayed to user. This class utilizes other controller classes in order to convey the user information about places, playlist, songs and their

account. This class is the essential element of the system in order to maintain information flow between user and system.

UserController: This controller class is responsible for doing user specific operations like register and login.

PlaceController: This class is responsible for creating a new place and joining user to a place.

SpotifyController: This controller class is responsible for providing functionalities to communicate with Spotify.

LocationController: This class will be communicating with *Google Maps API* to determine locations of *Users* and *Places* and handling location related issues.

2.2 Client

2.2.1 View

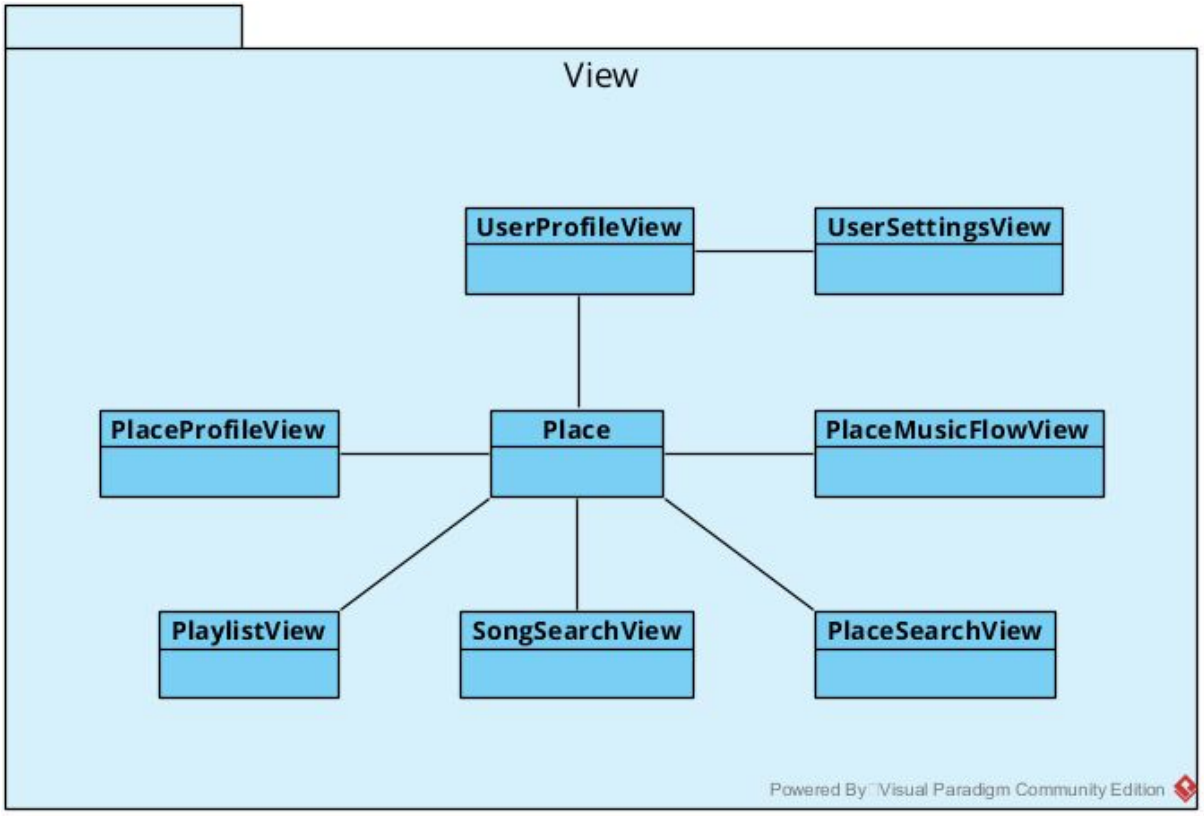

Figure 3: View package

UserProfileView: This view class will show details of the *User* and will be common for all kinds of users. However, details will be shown differently for different kinds of users. For instance, *RegisteredUsers* will be able to see and modify their email, password and Spotify information.

UserSettingsView: This view class will show details of *User* information and let them change their information such as password or Spotify account.

Place: This class will show the general information about a place such as the location so that users can see whether that place is nearby. Moreover, users will be able to see if the place has more than one branch by viewing their owner.

PlaceProfileView: This class will represent detailed information about a place so that a user can have an overall idea about the music taste of that place. Users are able to view the trending songs and general music genres of the place.

PlaceMusicFlowView: This view class will show user the music flow of the place he/she joined. It will display the currently playing song, the songs that are being voted to play next, their votes and ending time of the voting.

PlaylistView: This view class will display the current playlist of the place.

SongSearchView: This view class will be used for displaying the results of song search.

PlaceSearchView: This view class will be used for displaying the results of place search.

3. Class Interfaces

3.1 Server

3.1.1 Model

class User
This class is the base class of all user types. It has the basic attributes such as password, id, location, last login date informations. Moreover, User has the points attribute in order to enable the visitor to bid on music. User has visited places and requested songs attributes in order to keep track of the visitor activity and increase points if the visitor is a regular visitor of the place. Aforementioned attributes serve as necessary data to make place recommendations to the user.
Attributes
private int id
private String name
private Date lastLogin
private int points
private Location location
private List<VisitedPlace> visitedPlaces
private List<RequestedSong> requestedSongs
Methods
Getter and setter methods.

class RegisteredUser
This class inherits User class. Users that registered through either email or Spotify will be a registered user. Registered users will be able to create and control places. To allow this functionality, RegisteredUser stores user information such as email and Spotify auth token, and place related information such as owned places and premium account information.
Attributes
private SpotifyConnection spotifyConnection
private String email
private String password
private Date premiumEnd
private List<Place> places
private int premiumTier
Methods
Getter and setter methods.

class Place
Stores data related to places with shared lists.
Attributes
private int id
private String name
private SpotifyConnection spotifyConnection
private int pin

private Song[] votedSongs
private int[] votes
private Location location
private Playlist playlist
private SongRecord[] songRecords
Methods
Getter and setter methods
public void voteSong(int songIndex, int vote)
public void refreshVotedSongs()
public void refreshPin()

class PermanentPlace extends Place
Places that are permanent, like restaurants.
Attributes
private Genre[] genres
Methods
Getter and setter methods

class VisitedPlace
Stores record of a user visiting a place at a date.
Attributes
private DateTime date

private User user
private PermanentPlace place
Methods
Getter and setter methods

class Song
Stores data related to songs.
Attributes
private String name
private int duration
private List<Genre> genres
private SpotifyItem spotifySong
Methods
Getter and setter methods

class RequestedSong
Stores data related to requested songs.
Attributes
private Song song
private User requester
private Date date
private PermanentPlace place

Methods
Getter and setter methods

class SongRecord
Stores data related to played songs.
Attributes
private Song song
private int listenCount
Methods
Getter and setter methods

class SpotifyItem
This class stores data corresponding to Spotify counterparts of items like songs and playlists.
Attributes
private String id
private String uri
private String name
private String description
Methods
Getter and setter methods

class SpotifyConnection
This class stores Spotify connection related data.
Attributes
private String accessToken
private String refreshToken
private int expiressIn
Methods
Getter and setter methods

class Playlist
This class represents Spotify playlist of the *Place*. Requested songs will be added to the instance of this class and next songs will be selected from those songs.
Attributes
private int id
private Map<Integer, Song> songs
private int currentSong
private Time currentSongStartTime
private SpotifyItem spotifyPlaylist
Methods
public List <Song> getNextSongsForBidding()
public void changeCurrentSong(Song nextSong)
public void addSong(Song songToAdd)

class Genre
This class represents genre of music and will be compatible with genres on Spotify (e.g. Metal, Rock). This class is used to determine genres of *Songs* and *Places.*
Attributes
private int id
private String name
Methods
Get methods for attributes

class Location
This class represents physical location of *Places* and *Users.* This class will be used to determine in which *Place*, *Users* are. Also, this class will be compatible with *Google Maps* as we will be using it as main location data source.
Attributes
private int id
private double latitude
private double longtitude
private String district
private String city
private String country
Methods
Get methods for attributes

class PlackbackInfo
Stores playback information of currently connected user.
Attributes
private boolean isPlaying
private int timestamp
private int progress
private Song song
Methods
Getter and setter methods

3.1.2 Controller

class MainController
Stores playback information of currently connected user.
Attributes
private UserController userController
private PlaceController placeController
private SpotifyController spotifyController
private Song[] songs
private Genre[] genreList
Methods
Getter and setter methods

class UserController
This controller class is responsible for doing user specific operations like register and login.
Attributes
private List<User> users
private List<RegisteredUser> registeredUsers
Methods
public User createUser(struct UserInfo)
public String loginUser(String email, String password)

class PlaceController
This class is responsible for creating a new place and joining user to a place.
Attributes
private List<Place> places
Methods
public Place createPlace(struct PlaceInfo) Registers a new place in the system.
public boolean validatePlace(struct PlaceInfo) Checks the validity of the place user trying to join.

class SpotifyController
This class provides functionalities to communicate with Spotify Web API.
Attributes

private String clientId
private SpotifyConnection spotifyConnection
private String userId
private Playlist playlist
Methods
Getter and setter methods
public Playlist createPlaylist(string name, string description)
public void addSongToPlaylist(Song song, int position)
public void removeSongFromPlaylist(int position)
public void moveRangeInPlaylist(int start, int length, int insertBefore)
public PlaybackInfo getPlaybackInfo()
public void startPlayback()

class LocationController
This class will be communicating with *Google Maps API* to determine locations of *Users* and *Places* and handling location related issues.
Attributes
private List<Location> locations
Methods
public Location determineLocation()
public List<Place> findNearbyPlaces(Location location, List<Place> places)

3.2 Client

3.2.1 View

class UserProfileView
This view class will show details of the *User* and will be common for all kinds of users. However, details will be shown differently for different kinds of users. For instance, *RegisteredUsers* will be able to see and modify their email, password and Spotify information.
Attributes
private int id
private DateTime premiumStartDate
private DateTime premiumEndDate
private Map<Place, Integer> visitHistory
private Map<Song, Place> requestHistory
private List<Place> recommendedPlaces
Methods
private void goToUserSettingsView()

class UserSettingsView
This view class will show details of *User* information and let them change their information such as password or Spotify account.
Attributes
private int id
private String spotifyId

private String email
private DateTime premiumStartDate
private DateTime premiumEndDate
Methods
private void changeEmail()
private void changeSpotifyAccount()
private void removeSpotifyAccount()
private void subscribeToPremium()

class PlaceMusicFlowView
This view class will show user the music flow of the place he/she joined. It will display the currently playing song, the songs that are being voted to play next, their votes and ending time of the voting.
Attributes
private String currentlyPlaying
private String[] nextSongs
private int[] nextSongVotes
private int voteEndingTime

class PlaylistView
This view class will display the current playlist of the place.
Attributes

private int playlistId
private List<Song> songs
private int currentSongNumber

class SongSearchView
Stores search term and the results of song search.
Attributes
private String searchTerm
private SpotifyItem[] results

class PlaceSearchView
Stores search term and the results of place search.
Attributes
private String searchTerm
private String[] results
private int[] resultIds

class Place
This class will show the general information about a place such as the location so that users can see whether that place is nearby. Moreover, users will be able to see if the place has more than one branch by viewing their owner.

Attributes
private int placeId
private int spotifyId
private String name
private String address
private double[] coordinates
private boolean isUserOwner

class PlaceProfileView
This class will represent detailed information about a place so that a user can have an overall idea about the music taste of that place. Users are able to view the trending songs and general music genres of the place.
Attributes
private Place place
private String[] preferredGenres
private String[] popularSongs

4. References

[1] IBM, "UML - Basics," June 2003. [Online]. Available: http://www.ibm.com/developerworks/rational/library/769.html. [Accessed 17-Feb-2019].

[2] IEEE, "IEEE Citation Reference," September 2009. [Online]. Available: https://m.ieee.org/documents/ieeecitationref.pdf. [Accessed 17-Feb-2019].