******

**DEPARTMENT OF COMPUTER SCIENCE**

SOFTWARE ENGINEERING PROJECT

Mentored By: **Dr. SANGEETA SHARMA**

Anuj A. Gupta

151112096

CSE -1

YouPlay

Documentation

**Index**

1. Software engineering and their phases 3
2. Problem Statement & detailed description 8
3. Software Requirements 10
4. Data Flow Diagrams (DFD) 11
5. Class Diagram 13
6. SRS 14
7. Software Design 22
8. Software Coding 27
9. Testing, Reliability and Maintenance 31
10. User Manual 32
11. Study of Rational Rose 34
12. Study of StartUML 36

**Software Engineering & Its Phases**

Software is instructions (computer programs) that when executed provide desired function and performance. Software are data structures that enable the programs to adequately manipulate information. Software are documents that describe the operation and use of the programs.

The work associated with software engineering can be categorized into **three generic phases**, regardless of application area, project size, or complexity.

The **Definition Phase**focuses on *what.* That is, during definition, the software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system. The key requirements of the system and the software are identified. Although the methods applied during the definition phase will vary depending on the software engineering paradigm (or combination of paradigms) that is applied, three major tasks will occur in some form: system or information engineering , software project planning, and requirements analysis.

The **Development Phase**focuses on *how*. That is, during development a software

engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how procedural details are to be implemented, how interfaces are to be characterized, how the design will be translated into a programming language (or nonprocedural language), and how testing will be performed. The methods applied during the development phase will vary, but three specific technical tasks should always occur: software design, code generation, and software testing.

The **Support Phase**focuses on *change* associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. The support phase reapplies the steps of the definition and development phases but does so in the context of existing software. Four types of change are encountered during the support phase:

**Correction:** Even with the best quality assurance activities, it is likely that the

Customer will uncover defects in the software. *Corrective maintenance* changes

the software to correct defects.

**Adaptation:** Over time, the original environment (e.g., CPU, operating system,

Business rules, external product characteristics) for which the software was developed is likely to change. *Adaptive maintenance* results in modification to

the software to accommodate changes to its external environment.

**Enhancement:** As software is used, the customer/user will recognize additional

functions that will provide benefit. *Perfective maintenance* extends the

software beyond its original functional requirements.

**Prevention:** Computer software deteriorates due to change, and because of

this, *preventive maintenance,* often called *software reengineering,* must be conducted to enable the software to serve the needs of its end users. In essence,

preventive maintenance makes changes to computer programs so that they can

be more easily corrected, adapted, and enhanced.

A development process consists of various phases, each phase ending with a defined output. The phases are performed in an order specified by the process model being followed. The main reason for having a phased process is that it breaks the problem of developing software into successfully performing a set of phase, each handling a different concern of software development.

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

**Basic step**

This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms. He submits his request to the service providing organization in writing.

**Requirement Gathering**

This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements.

**Software Design**

Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

**Coding**

This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

**Testing**

An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user’s end. Early discovery of errors and their remedy is the key to reliable software.

**Operation and Maintenance**

This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

**Problem Statement & Description**

“As the number of videos in a YouTube Playlist increases, downloading the whole playlist becomes cumbersome.”

**What does this mean?**

YouTube is the most popular website for video sharing and is used by a wide number of individuals and media corporations .YouTube playlists come in handy while learning certain topic or concept. Videos are arranged according to the concept and are really helpful.

To download videos from YouTube, we have to either use a website as a plugin or a video grabber. Even though downloading a large number of videos is quite cumbersome as videos need to be manually grabbed. YouTube playlists come in handy while learning certain topic or concept. Videos are arranged according to the concept and are really helpful.

**How can this problem be solved?**

The answer is simple, by using a download manager. A​ download manager is a computer program dedicated to the task of [downloading​](https://en.wikipedia.org/wiki/Downloading)​possibly unrelated stand-alone files from the [Internet ​](https://en.wikipedia.org/wiki/Internet)​for storage.

**How can such a Download Manager Developed?**

This can be accomplished by developing a web-crawler and using the crawled links to download the videos. Python provides a large number of packages for this task. A web crawler can be build which accepts the watch URL of the first video of playlist and the number of videos of the playlist to be downloaded. From this link the downloadable link of the video can be generated and the watch URL of next video can be captured. This concept is implemented in YouPlay.

**What is YouPlay?**

YouPlay is a download manager written in python 3.0.It is designed to be used on windows operating system. It is a sort of web-crawler cum downloader. It can be used to download a stand-alone video to the whole playlist. It can be used to find the number of videos in a playlist to downloading only few of them.

YouPlay asks the user for the URL of playlist. It then shows the number of videos in the playlist and list of videos. User is supposed to enter the video number from which to begin the download & video number of last video. It also asks for the directory location in which the videos are supposed to be saved. It even creates the directory for you if it does not exist.

YouPlay works in the background and notifies the user on completion of the download by ringing an alert tone. YouPlay always downloads the videos in the best file format available.

**Software Requirements**

**Active Internet Connection**

As YouPlay is a download manager, it requires an active internet connection.

**Python 3.0 Interpreter**

YouPlay has been coded in python3.0. Hence it requires python3.0 interpreter (IDLE) for its execution.

**Windows Operating System**

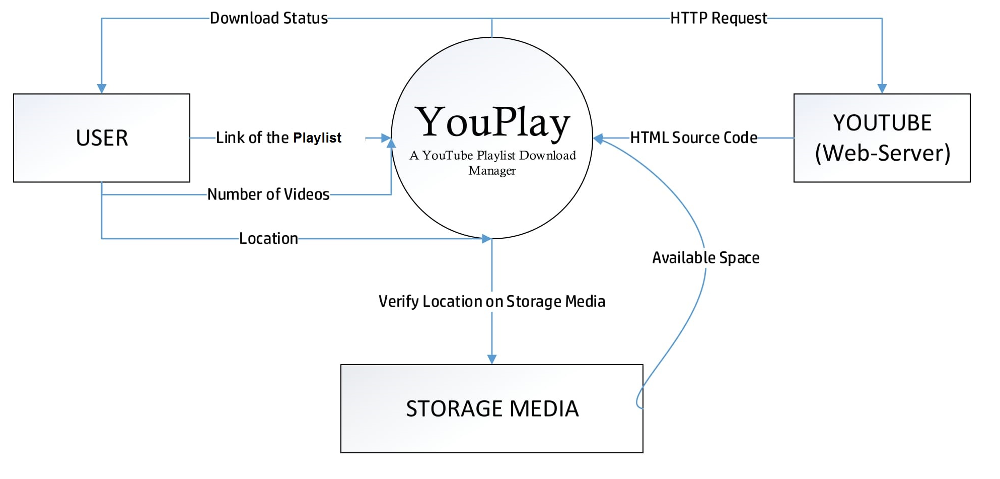
Any version of windows operating system is required for the software.

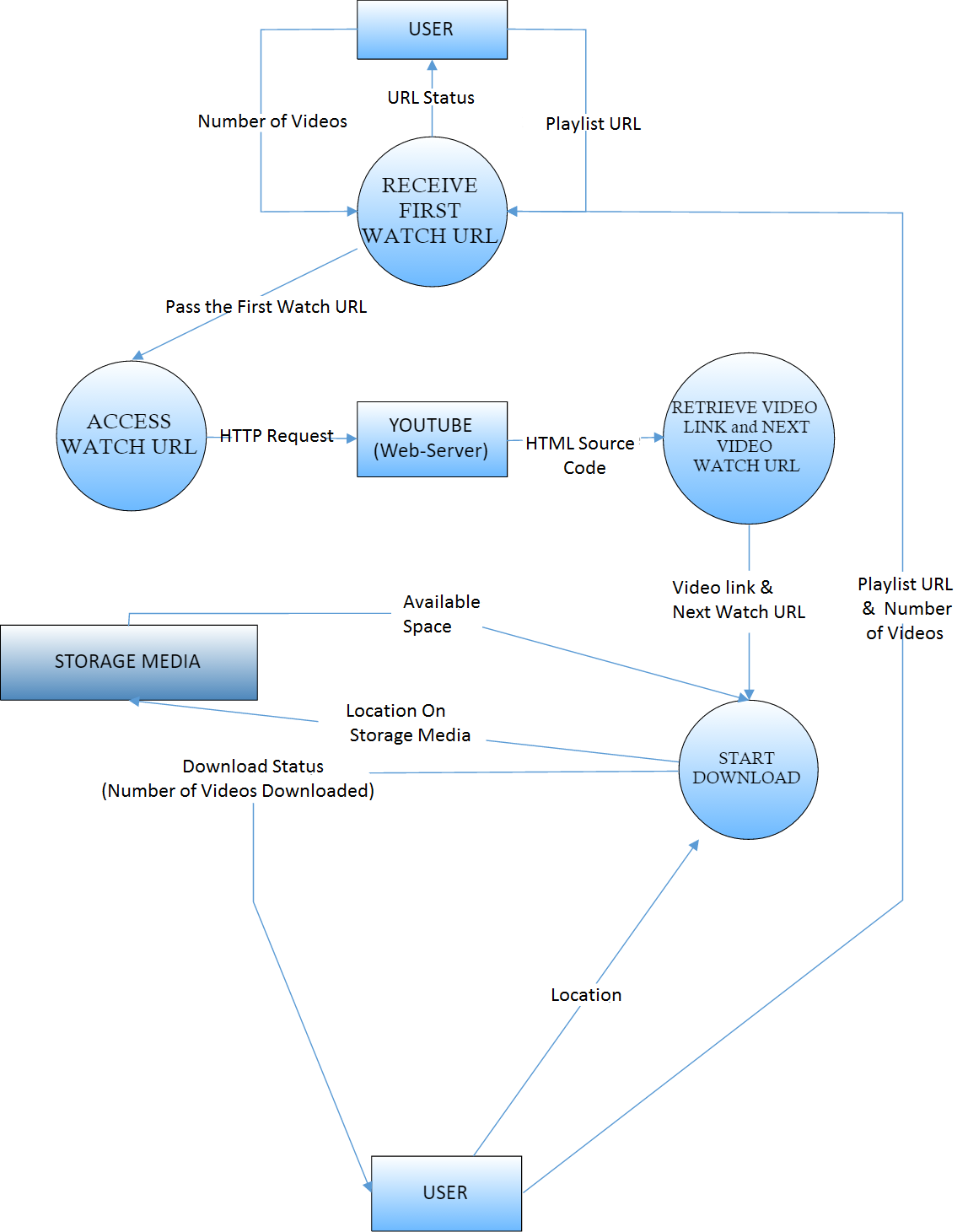
**Space on Storage Media**

YouPlay requires at least 100Mb of space for its execution as the size of videos differ from playlist to playlist.

**Data Flow Diagram**

**Context Free Diagram**

**Logical Data Flow Diagram**



**Class Diagram**

PLAYLIST

* URL
* No. of Videos
* Source Code
* Location
* Quality
* List of all the videos

get\_url()

isPlaylist()

count\_videos()

total\_videos()

get\_video\_links()

get\_path()

display\_links()

get\_quality()

save()

VIDEO

* URL
* Location
* File Format
* Resolution

set\_playlist\_video\_url()

get\_url()

get\_details()

set\_from\_quality()

save()

**Has**

**Software Requirements & Specifications**

**Table of Contents**

[1. Introduction](#page5) [16](#page5)

[1.1 Purpose](#page5) [16](#page5)

[1.2 Scope](#page5) [16](#page5)

[1.3 Definitions, Acronyms and Abbreviations](#page5) [16](#page5)

[1.4 References](#page6) [17](#page6)

[1.5 Overview](#page6) [17](#page6)

[2. Overall Description](#page6) [18](#page6)

[2.1 Product Perspective](#page6) [18](#page6)

[2.1.1 System Interface](#page6) [18](#page6)

[2.1.2 User interface](#page6) [18](#page6)

[2.2 Product functions](#page8) [18](#page8)

[2.2.1 Video](#page8) Download [18](#page8)

[2.2.2 Playlist](#page8) Download [18](#page8)

[2.3 User Characteristics](#page16)... ..18

[2.4 Constraints](#page16) ..19

[2.4.1 User Interface Constraints](#page16) ..[19](#page16)

[2.4.2 Hardware Constraints](#page17) ..19

[2.4.3 Software Constraints](#page17) ..19

[3. Specific Requirements](#page18) 20

[3.1 External Interface](#page18) [20](#page18)

[3.1.1 User](#page18) Interface 20

[3.1.2 Hardware](#page18) Interface 20

[3.1.3 Software](#page18) Interface 20

[3.1.4 Communication](#page18) Interface 20

[3.2 Functional Requirements](#page18) 20

[3.3 Performance Requirements](#page25) 20

[3.4 Design Constraints](#page25) [20](#page25)

[3.5 Software System Attributes](#page25) [2](#page25)1

[3.5.1 Reliability](#page25) [21](#page25)

[3.5.2 Portability](#page25) [2](#page25)1

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to capture, in natural language and at a functional level, the description and requirements of a download manager to download videos and playlist from YouTube (<https://youtube.com>). The focus here is the complexity of the problem. This is a functional description of those features required to address current user requirements. A short discussion accompanies each requirement, to add the background and framework necessary to explain the functionality. It also describes nonfunctional requirements and other factors necessary to provide a complete and comprehensive description of the requirements for the software.

**1.2 Scope**

The scope of the project will be the system on which the software is installed, i.e. the project is developed as a desktop application, and it will work for a particular user. This software system will be a web crawler-cum-web scrapper-cum-download manager.

**1.3 Definitions, Acronyms and Abbreviations**

**1.3.1 URL**

URL stands for Uniform Resource Locator. It is a reference to a [web resource](https://en.wikipedia.org/wiki/Web_resource) that specifies its location on a [computer network](https://en.wikipedia.org/wiki/Computer_network) and a mechanism for retrieving it. Here, it refers to the URL of video or the playlist.

**1.3.2 Video**

The standalone video to download from YouTube.

**1.3.3 Playlist**

A collection of the YouTube videos.

**1.3.4 System**

System refers to the existing software.

**1.3.5 Server/Remote Server**

Server/ remote server refers to YouTube.com.

1. **References**
   1. IEEE 830 Template
   2. pyTube (<https://github.com/nficano/pytube>)
2. **Overview**

This is a working document and, as such, is subject to change. In its initial form, it is incomplete by definition, and will require continuing refinement. Requirements may be modified and additional requirements may be added as development progresses and the system description becomes more refined. This information will serve as a framework for the current definition and future evolution of the YouPlay.

**2. Overall Description**

**2.1 Product Perspective**

YouPlay is meant to serve as an independent platform where various tasks associated with handling videos of the playlist organizing the downloaded videos according to their playlist sequence will be automated. Our goal is to develop a user friendly environment for downloading large number of videos from the site(YouTube).

**2.1.1 System Interface**

The program that will perform all the operations is written in python 3.6.1.

**2.1.2 Memory Constraints**

Memory constraints will come into play when the size of playlist grows to a considerable size. YouPlay will require at least 100 Mb of storage space to run

**2.2 Product functions**

**2.2.1 Video Download**

The system shall provide the user the ability to download a single video from the server.

**2.2.2 Playlist Download**

The system shall provide the functionality to download the playlist from the server.

**Download all the videos**

The user shall be provided with the functionality to download all the videos of the playlist by default.

**Download selected videos**

The user shall be provided with the functionality to download only the selected videos of the playlist.

**2.3 User Characteristics**

Not Applicable

**2.4 General Constraints**

**2.4.1 User Interface Constraints**

Using this system is fairly simple and intuitive. The system provides necessary prompts at each point of execution. Thus a user familiar with basic command line navigation skills should be able to understand all functionality provided by the system.

**2.4.2 Hardware Constraints**

The system should work on most home desktop and laptop computers which have Python 3.6.1 installed.

**2.4.3 Software Constraints**

The system will be intended to run on Windows Operating System 7 or above.

**2.5 Assumptions and dependencies**

Most of the download managers have a lot of redundant features which are rarely used in downloading a playlist. Our new system focuses on the ease with which playlist can be downloaded.

**3. Specific Requirements**

**3.1 External interface**

**3.1.1 User Interface**

The system will provide console interface to the user. Over the time, system can be modified to offer a GUI to the user which will be very intuitive.

**3.1.2 Hardware Interface**

Monitor screen – the software shall display information to the user via the monitor screen

Keyboard – the software shall interact with the keystrokes of the keyboard. The keyboard will input data into the active area of the console.

**3.1.3 Software Interface**

The system requires windows operating system and python 3.6.1 interpreter to run on the user’s computer.

**3.1.4 Communication Interfaces**

The HTPP or HTTPS protocol(s) will be used to facilitate communication between the client and server.

**3.2 Functional Requirements**

Not Applicable

**3.3 Performance Requirements**

The system should support at least 100 simultaneous download.

This statement provides a general sense of reliability when the system is under load. It is important that a substantial number of videos should be downloaded by the system at the same time, since our aim is to make the playlist downloading user-friendly. Therefore, it must be able to handle at least 100 simultaneous downloads.

**3.4 Design Constraints**

* 1. The communication between the system and the server will be in http/https.
  2. The product will be written in Python 3.6.1.
  3. The source code must follow the coding conventions of Python.
  4. Users must have access to comprehensive documentation.
  5. **Software System Attributes**

The software consists of a python program.

**3.5.1 Reliability**

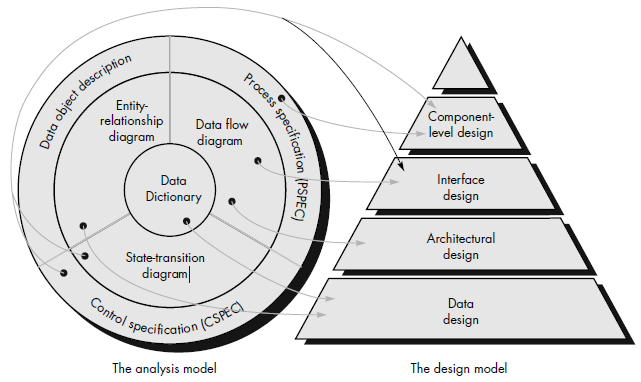
The reliability of the overall program depends on the reliability of the separate components.

**3.5.2 Portability**

The application is Windows-based and should be compatible with other systems. The end-user part is fully portable and any system using any web browser should be able to use the features of the application.

**Software Design**

Software design is an iterative process through which requirements are translated into a “blueprint” for constructing the software. Initially, the blueprint depicts a holistic view of software. That is, the design is represented at a high level of abstraction—a level that can be directly traced to the specific system objective and more detailed data, functional, and behavioral requirements. As design iterations occur, subsequent refinement leads to design representations at much lower levels of abstraction. These can still be traced to requirements, but the connection is more subtle.

****

Translating the analysis model into a software design

**1. Modules**

The modules present in the system are:

1. Functions
2. Third party module – pyTube
3. Main
4. GUI module (proposed)

**1.1 Functions**

Functions will contain two classes video and playlist. These classes will deal with downloading of a video and a playlist respectively. Playlist class will use video class for downloading its individual video. The data exchanged between the two classes will be string data items. Both classes will contain methods to manipulate the properties of the video or the playlist such as its url, location to be saved, quality, resolution, file formats etc.

**1.2 pyTube**

This module is a third party module and will help in downloading videos from YouTube. It contains classes and functions to manipulate the data required to download the video which user is unable to access.

**1.3 Main**

This module will contain the main function which will mark the beginning of execution of our system. It will prompt the user to select the video downloading or playlist downloading option.

**1.4 GUI**

The software can be updated over time and a graphical user interface can be given to it. GUI will contain functions to render different graphical user interface elements and will make the system more intuitive and user friendly. This module will help us to achieve our goal to make the downloading of playlist simple and user-friendly. It will contain function to render the windows, prompts, checkboxes to select the videos etc.

**2. Functional Independence**

Functional independence is achieved by developing modules with “single-minded” function and an "aversion" to excessive interaction with other modules. Stated another way, we want to design software so that each module addresses a specific sub-function of requirements and has a simple interface when viewed from other parts of the program structure. Independence is measured using two qualitative criteria: cohesion and coupling.

Cohesion is a measure of the relative functional strength of a module. Coupling is a measure of the relative interdependence among modules.

A module with high cohesion and low coupling is said to be functionally independent.

**1.1 Functions**

Functions will contain two classes video and playlist. These classes will deal with downloading of a video and a playlist respectively. Hence, both of them will populate to achieve a single function of managing and downloading video and playlist respectively. Thus, Functions module has *functional cohesion*.

**1.2 pyTube**

This module is a third party module and will help in downloading videos from YouTube. It contains classes and functions to manipulate the data required to download the video which user is unable to access. Thus, pyTube module has *functional cohesion*.

Both this module will exchange data in the form of string for the display of prompt and integers to store the selected values and opted options. Thus, these modules are *data coupled.*

Hence, the modules are Functionally Independent.

**3. Interface**

The data items that will exchanged between the Functions module, the main module and the GUI module will be:

1. List: The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.
2. String: This are the characters enclosed within single-quotes or double-quotes.

**3. Data Structures**

The data structures that will be used in both the module are in built data structures of python:

1. List: The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.
2. String: This are the characters enclosed within single-quotes or double-quotes.

**Software Coding**

**Sample Code**

**PlayList class of functions.py Module**

class PlayList:

# Constructor - initializes the data members

def \_\_init\_\_(self):

self.url = None # To store the url of the playlist

self.num = 0 # To store the total number of videos in the playlist

self.list = [] # To store the individual url of the videos

self.source\_code = "" # To store the http response

self.quality = None # To store the video quality of the playlist

self.location = None # To store the file location

# Asks the URL and checks the internet

def get\_url(self):

print("Enter the URL of playlist")

self.url = input()

flag = 0

# Check the internet and hang the script until the connection activates

while True:

try:

response = urllib.request.urlopen(self.url, )

self.source\_code = str(response.read())

if flag:

print("Connected to the remote server")

return True

except Errors.URLError:

if flag == 0:

print("Could not connect to the remote server. Check your internet connection")

flag = 1

continue

# This function determines whether the entered url is of a valid playlist or not

def isPlaylist(self):

while True:

try:

self.source\_code.index('<meta property="og:site\_name" content="YouTube">')

break

except ValueError:

print("Invalid URL")

self.get\_url()

while True:

try:

self.source\_code.index("pl-header-details")

break

except ValueError:

print("Invalid URL")

self.get\_url()

# This function counts the total number of videos present in the playlist

def count\_videos(self):

temp\_index = self.source\_code.index("pl-header-details")

part\_of\_code = self.source\_code[temp\_index:]

temp\_index = part\_of\_code.index("</li><li>")

part\_of\_code = part\_of\_code[temp\_index:]

temp\_index = part\_of\_code.index("videos") - 1

part\_of\_code = part\_of\_code[0:temp\_index]

part\_of\_code = part\_of\_code[9:]

number\_of\_videos = int(part\_of\_code)

self.num = number\_of\_videos

# This function displays the number of videos in the playlist

def total\_videos(self):

print("The total number of videos in the playlist are ", self.num)

# This function scraps and stores the url of each video in list data member

def get\_video\_links(self):

code = self.source\_code

for i in range(self.num):

a = code.index("pl-video-title-link yt-uix-tile-link yt-uix-sessionlink")

code = code[a:] # list slicing

a = 0

b = code.index("pl-video-bottom-standalone-badge")

link = code[a:b]

x = link.index('href')

y = link.index("data-sessionlink") - 2

x += 6

link = link[x:y]

code = code[b:]

link = "https://www.youtube.com" + link

self.list.append(link) # inbuilt list function

# This function prompts the user for the location at which playlist is to be saved

def get\_path(self):

print("Enter the file location to be saved")

while True:

try:

path = input()

if not os.path.exists(path): # exists() check whether the entered path exists or not

os.mkdir(path) # if path does not exists mkdir() creates it

break

except FileNotFoundError:

print("Invalid Path. Enter the path again")

continue

self.location = path

# This functions prompts the user for the videos quality of the playlist

def get\_quality(self):

print("The playlist is available in following resolutions:")

print("High\nMedium\nLow")

while True:

try:

quality = input()

except KeyboardInterrupt:

continue

if quality != "High" and quality != "Medium" and quality != "Low":

print("Invalid input. Enter again.")

else:

break

self.quality = quality

# This functions downloads the individual video of the playlist

def save(self):

for video\_url in self.list:

temp\_video = video()

temp\_video.set\_playlist\_video\_url(video\_url)

temp\_video.set\_from\_quality(self.quality, self.location)

temp\_video.save()

Following observations can be drawn from the above sample code:

* Naming conventions for variables are followed.
* Exception handling mechanism has been included
* Code is well documented
* No goto statements have been used

**Testing, Reliability and Maintenance**

**Testing**

Every method of all the classes has been checked for all the possible inputs. Exceptions have been handled properly. Thus, there are no potential errors in system.

**Reliability**

* YouPlay offers the user 90% reliability.
* It may not work due to bad internet connection.
* It may not be able to download videos due to unstable network.

**Maintenance**

If the system crashes due to some reasons and the download is not completed before then user is supposed to restart the system and enter command “resume”. The system will automatically resume the download.

**User Manual**

This user manual lists all the commands that a user can enter on starting the system and the prompts that appear on entering those commands.

**Commands**:

1. **video**

This command is used to download a single video from the server.

The prompts which appear are listed in sequence –

*“Enter the URL of video”*

https://www.youtube.com/watch?v=EwIljm4gFoE

*“The video is available in following formats:*

*144p*

*240p*

*360p*

*720p*

*Enter the resolution:”*

720p

*“Enter the file location to be saved”*

E:\YouPlay

*“Download completed”*

The text in italics shows prompts and underlined text shows sample input.

If the input is not appropriate then error will be displayed.

For e.g. –

“Could not connect to the remote sever. Check Your internet connection”, “Invalid URL” etc.

1. **playlist**

This command is used to download the whole video playlist from the server.

The prompts which appear are listed in sequence –

*“Enter the URL of playlist”*

https://www.youtube.com/playlist?list=PL32BC9C878BA72085

*“Enter the file location to be saved”*

E:\YouPlay\Tutorial

*“The playlist is available in following resolutions:*

*High*

*Medium*

*Low”*

Medium

The text in italics shows prompts and underlined text shows sample input.

If the input is not appropriate then error will be displayed.

For e.g. –

“Could not connect to the remote sever. Check Your internet connection”, “Invalid URL” etc.

1. **resume**

This command is used to complete the previous download i.e. to complete the downloading of the previous video or playlist.

**Study of Rational Rose**

ROSE = Rational Object Oriented Software Engineering. Rational Rose is a set of visual modeling tools for development of object oriented software.

Visual Modeling is the process of graphically depicting the system to be developed

• Presenting essential details

• Filtering out non-essential details

• Viewing the system from different perspectives

Rational Ross is used as-

• Modeling can be useful at any point in the application development process.

• Initial Design Work (Requirement Analysis and Definition)

• Use Cases

• Class Diagrams

• Sequence Diagram

• Rational Rose includes tools for reverse engineering as well as forward engineering of classes and component architectures.

• You can gain valuable insights to your actual constructed architecture and pinpoint deviations from the original design.

• Rose offers a fast way for clients and new employees to become familiar with system internals.

Diagrams drawn using Rational Ross-

1) Class Diagram

2) Object Diagram

3) Use Case Diagram

4) Sequence Diagram

5) Collaboration Diagram

6) State chart Diagram

7) Activity Diagram

**Study of StarUML**

StarUML™ is a software modeling platform which supports UML (Unified Modeling Language). It is based on UML version 1.4 and provides UML version 2.0 notations and eleven different types of diagram. It actively supports the MDA (Model Driven Architecture) approach by supporting the UML profile concept. StarUML™ is excellent in customizability to the user’s environment and has a high extensibility in its functionality.

StarUML™ provides maximum customization to the user’s environment by offering customizing variables that can be applied in the user’s software development methodology, project platform, and language. The UML models act as an architectural blueprint for software development.

We need a modeling platform rather than just a UML tool because of-

• End users want customizable tools. Providing a variety of customizing variables to meet the requirements of the user environment can ensure high productivity and quality.

• No modeling tool provides a complete set of all possible functionalities. A good tool must allow future addition of functions to protect the user’s investment costs in purchasing the tool.

• MDA (Model Driven Architecture) technology requires not only independent platforms but multi-platform functionality. Modeling tools confined to specific development environments are not suitable for MDA. The tool itself should become a modeling platform to provide functionality for various platform technologies and tools.

• Integration with other tools is vital for maximization of the tool’s efficiency. The tool must provide a high level of extensibility, and allow integration with existing tools or user’s legacy tools.

StarUML models provides the following:

• Identify requirements and communicate information

• Allows focus on how system components interact, without get bogged out in specific details

• Allows you to see relationships among design components

• Improves communication across your team through the use of common graphical language.