MINOR PROJECT REPORT

Twitter Trend Analyser

Submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology
Computer Science and Engineering

Guide(s):
Deepika Rawat

(Assistant Professor, HMRITM)

Submitted by:

- Tejsi Sharma (01996592717)
- Shivam Singhal (40596502717)
- Mansi Jindal (35496502717)
- Anshul Gupta (00296507218)

HMR INSTITUTE OF TECHNOLOGY & MANAGEMENT
HAMIDPUR, DELHI 110 036

Affiliated to
GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY
Sector - 16C Dwarka, Delhi - 110075, India

2017-21

Table of Contents

DECLARATION

I hereby declare that the project work entitled “Twitter Trend analyser” submitted to the HMR Institute of Technology and Management, is a record of an original work done by me under the guidance of Ms. Deepika Rawat, Assistant Professor, HMR Institute of Technology and Management, and this project work is submitted in the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering. The results embodied in this report and project have not been submitted to any other University or Institute for the award of any degree.

TEJSI SHARMA
01996502717

DECLARATION

I hereby declare that the project work entitled “Twitter Trend analyser” submitted to the HMR Institute of Technology and Management, is a record of an original work done by me under the guidance of Ms. Deepika Rawat, Assistant Professor, HMR Institute of Technology and Management, and this project work is submitted in the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering. The results embodied in this report and project have not been submitted to any other University or Institute for the award of any degree.

SHIVAM SINGHAL
40596502717

DECLARATION

I hereby declare that the project work entitled “Twitter Trend analyser” submitted to the HMR Institute of Technology and Management, is a record of an original work done by me under the guidance of Ms. Deepika Rawat, Assistant Professor, HMR Institute of Technology and Management, and this project work is submitted in the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering. The results embodied in this report and project have not been submitted to any other University or Institute for the award of any degree.

MANSI JINDAL
35496502717

DECLARATION

I hereby declare that the project work entitled “Twitter Trend analyser” submitted to the HMR Institute of Technology and Management, is a record of an original work done by me under the guidance of Ms. Deepika Rawat, Assistant Professor, HMR Institute of Technology and Management, and this project work is submitted in the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering. The results embodied in this report and project have not been submitted to any other University or Institute for the award of any degree.

ANSHUL GUPTA
00296507218

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Ms. Deepika for their guidance as well as for providing necessary information regarding the project & also for their support in completing the project.
I would like to express my gratitude towards my parents & member of HMR Institute of Technology and Management for their kind co-operation and encouragement which help me in completion of this project.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

ABSTRACT

The ability to monitor trends in real time is an incredibly valuable tool for anticipating changes in the market and remaining at the forefront of competition. Differentiating a brand’s products and services from those of competitors is a primary concern for marketers in any industry. However, analysis of market trends must extend beyond the activities of competitors - brands must be able to detect new consumer behaviors, habits, and lifestyles and analyze their potential impact on the brand’s products and overall industry market. Such data enables brands to develop new products, collect feedback regarding existing products, and effectively segment their service offerings to target new audiences. Analyzing online conversations is an invaluable method for understanding public perception, the specific expectations, and emerging market trends relevant to a brand’s products and services. However, analyzing online conversations is not enough: brands should connect with opinion leaders, specialized media researchers, journalists, and market experts in order to discover emerging trends and to understand pre-existing trends.

The first step in researching trends should always be to define your goals. For example, a brand’s goals might involve discovering macro trends in its industry, gain insights on the target audience, research consumer behavior, or collect feedback on products and services. Once these goals have been defined, it’s important to select tools which will enable the brand to detect and process relevant data.

An excellent method for discovering new topics of interests and sources to monitor is to analyze the evolution of social media conversation on industry-related concepts. Ideally, research should begin with general concepts before analysis is refined to keywords which are consistently used over a relatively long time period. Finally, relevant ideas can be segmented into clusters of similar topics for easier collection of data most relevant to the study’s interest.In order to be at the forefront of industry trends and developments, brands innovate by listening to the needs and concerns of consumers, industry experts, and opinion leaders. Once the primary sources of information are identified, it’s important to determine the main ideas which will be helpful for identifying emerging, growing, maturing or declining trends.

INTRODUCTION

Social media is dramatically changing the landscape as to how companies communicate, market and sell their products and service. As a new type of relationship is being formed between companies and their customers, smart businesses are learning how to talk, share, listen, participate and network with potential and existing customers. Done well, companies can dramatically increase their bottom line by creating customer ambassadors who then go out and create more customers. Done poorly, companies can destroy their brand in a day and curtail their potential for growth. Due to the potential power of social media, businesses are investing in everything from content management systems, forums, writing armies, and multi-channel applications.

In 2010, Ford gave 100 independent agents (social media influencers) a free Ford Fiesta at a cost of $5 million. Those social media influencers included bloggers, filmmakers and others who were in the public eye. Each of these influencers began to share their experiences online about driving a Ford Fiesta. Within a relatively short period of time, Fiesta reach 60% recognition before the cars even made it to the showroom. For the price of $5 million it was a bargain campaign compared to traditional advertising. Social media analytics is rapidly evolving and so it is important for companies to stay on top of their analytical game. The challenge is that there is so much data that it is easy to get lost in a sea of numbers. The key is to make sure that the company focuses on the data that really matters, and create a defined process and standard around measurement. Businesses need to articulate what success looks like, and then align their social media goals by tying them to the company's overall goals. Choose three to five Key Performance Indicators (KPIs) and focus on improving those.

Popular social media platforms such as Twitter has proven to be effective channels for disseminating falsified information, unverified claims, and fabricated attention-grabbing stories due to their wide reach and the speed at which this information can be shared. Recently,
there has been an increased number of disturbing incidents of fabricated stories proliferated through social media having a serious impact on real-world events. False news stories distributed in social media vary depending on the intent behind falsification. Unlike verified news, suspicious news tends to build narratives rather than report facts. On one extreme is disinformation which communicates false facts to deliberately deceive readers or promote a biased agenda. These include posts generated and retweeted from propaganda and so-called clickbait ("eye-catching" headlines) accounts.

Fake News focuses on classifying the credibility of a tweet post. It makes and presents some scores and their interpretation. Online news has the power to influence millions of people, but there are no ways to establish what it is true and what it is fake. We might guess what common sense says it's true, but sometime we might just need some impartial, trustable source of news.

Studies show that everybody has problems with identifying fake news, disregarding criteria's such as age or education. Some 82% of middle-schoolers could not distinguish between an ad labelled "sponsored content" and a real news story on a website, according to a Stanford University study of 7,804 students from middle school through college. Many students judged the credibility of newsy tweets based on how much detail they contained or whether a large photo was attached, rather than on the source.

A trend analysis, in its simplest terms, is a tool and technique you can use within the project management process. Okay, do you feel like you've got it now? Just joking – of course you don't, let's spend some time discussing trend analysis and why it is a good PMP topic to know.
Beyond the simple definition provided above, a trend analysis is a mathematical tool you can use to assess your project data to determine how it is doing and forecast how it will continue to do.

It is common to see a trend analysis represented in graphical form. This way you can easily see the way your project is trending. See that trending – this means is your project on track, improving, or falling behind, as we discussed above.

Please, disregard the subject of the analysis, but here is a sample of a trend analysis as well as thorough discussion on what the data indicates. Again, this is just a helpful example, the content is not relevant to the PMP exam.

1.1 General

Social media has now become synonymous with digital marketing, going hand-in-hand with most – if not all – digital campaigns. However, social media is far from static and what worked a few months ago may not get you the same good results now.

Habits change, platforms evolve, and new platforms come into existence. All of this influences how people use and react to social media marketing, as well as how marketers are able to reach their audience. It is more important than ever before for marketers to understand and stay ahead of the curve when it comes to social media. Doing so ensures you have the right tools at your disposal, an up-to-date strategy, and the required skills to make the most of social media.

Remember in addition to understanding what a trend analysis is, it is equally important to understand how each topic relates back to the overall project management processes.

A trend analysis is a tool and technique used during the monitor and control phase of the project life-cycle (or process group). You will use it in the integration, time, cost, and risk knowledge areas. If you are asking yourself, what is a tool and technique? These are the actions or items that are applied to an input to get the intend output.

Think about it, the integration knowledge area wants to understand how the 47 processes of project management work together – thus it would make sense that you want to analyse how that work is progressing. Same with time, cost, and risk. You will want to know if your project is tracking in the direction you planned. You would not want your schedule or budget to be falling behind and know nothing about it.

Back to the definition of tool and technique, if we use the time knowledge area, we can use a trend analysis to assess our project schedule and create an output of a project change request if our project is currently falling behind schedule. Using the trend analysis is critical to help your project complete on time, within budget, with the intended scope.

On the basis of regions, North America dominated the market in 2018 due to the adoption of emerging technologies. The Asia-Pacific region is expected to witness the fastest growth during the forecast period, owing to the increasing number of social media users in the region. On the basis of analytics type, the social media analytics market is segmented as prescriptive analytics, predictive analytics, descriptive analytics, and diagnostic analytics. Predictive analytics is used to understand the positive and negative sentiments of customers.

1.2 Overview of Project

Social media is dramatically changing the landscape as to how companies communicate, market and sell their products and service. As a new type of relationship is being formed between companies and their customers, smart businesses are learning how to talk, share, listen, participate and network with potential and existing customers.

Done well, companies can dramatically increase their bottom line by creating customer ambassadors who then go out and create more customers. Done poorly, companies can destroy their brand in a day and curtail their potential for growth. Due to the potential power of social media, businesses are investing in everything from content management systems, forums, writing armies, and multi-channel applications.

Social media analytics is rapidly evolving and so it is important for companies to stay on top of their analytical game. The challenge is that there is so much data that it is easy to get lost in a sea of numbers. The key is to make sure that the company focuses on the data that really matters, and create a defined process and standard around measurement. Businesses need to articulate what success looks like, and then align their social media goals by tying them to the company's overall goals. Choose three to five Key Performance Indicators (KPIs) and focus on improving those.

Social media analytics is an approach of collecting data from various social media platforms and then analysing the data to make relevant business decisions. It uses social media analytics tools such as analytics, icon square, tailwind, google analytics, and others, to extract data related to customer sentiments, in order to support an organization's marketing activities and customer service. Social media provides opportunities to organizations to connect with their customers. It is essential to test and track the results generated through social media in order to identify the most effective strategies.

The global social media analytics market is expected to grow at a CAGR of 28.7% during the forecast period 2016-2023. The Social Media Analytics market is growing due to the increasing need for effective brand promotions among market competitors. The increasing use of smartphones and social media by people is driving the demand for social media analytics market. Moreover, the growing demand for customer engagement by organizations to build a strong customer base and increase customer retention has further generated demand for social media analytics. The data available on social media can be false at times and this can result in a wrong analysis of data, which in turn, can hinder the market demand. The market will further witness significant growth during the forecast period, owing to an increase in the use of social media analytics by SMEs.

Global Social Media Platforms Market report 2024 focuses on the major Types and Applications for the key players. Global Social Media Platforms market research report also provides analysis of the market share, segmentation, revenue forecasts and geographic regions of the market. The Social Media Platforms market research report is a professional and in-depth study on the current state of Global Industry.

The Social Media Platforms market report accounts pivotal information pertaining to the growth drivers and restraints that will define the industry growth in the upcoming years. Moreover, it identifies the opportunities existing across the various regions to further aid business expansion.

In recent times, the coronavirus outbreak is peaking in some markets while its lingering impact continues to challenge others. Amid the uncertainties, companies are revising their budget for reopening and reinventing with full force but now they must consider the pandemic’s progression and its recurrence across the various geographies. Our deep dive analysis of this business sphere will not only help you chart a plan of action for recovery but will empower you in crafting strategies to remain profitable.

Apart from this, the research report also delivers an in-depth evaluation of the various sub-markets to impart a better understanding of the revenue prospects of this industry.

Main pointers from the Social Media Platforms market report:

- Covid-19 impact on remuneration scale of the industry.
- Database of the sales volume, overall market revenue and size.
- Key industry trends.
- Opportunity windows.
- Projected values for the growth rate of the market.
- Advantages & disadvantages of direct and indirect sales channels.
- Major distributors, traders, and dealers.

Social Media Platforms market segments included in the report:

- Regional segmentation: North America, Europe, Asia-Pacific, South America, Middle East and Africa
- Country-level analysis.
- Total sales, revenue, and market share of each region.
- Projected returns and growth rate of each geography over the analysis period.
- Product gamut: Web-based and Cloud-based
- Market share prediction based on the sales and revenue generated by each product category.
- Pricing patterns of each product type.
- Application spectrum: Personal and Commercial
- Revenue and sales accrued by each application segment over the estimated timeframe.
- Product pricing based on their application scope.
- Competitive outlook: The major players covered in Social Media Platforms are: Facebook, YouTube, Twitter, Sine, Google, Tencent, Yelp, LinkedIn, Instagram, Pinterest and Foursquare

- Business overview, manufacturing facilities, and top competitors of the listed companies.
- Products and services offered by the leading players.
- Pricing model, sales, revenue, gross margins, and market share of each company.
- SWOT analysis for each company.
- Assessment of business centric aspects like commercialization rate, market concentration ratio, and popular marketing strategies.

Development Trend of Analysis of Social Media Platforms Market

- Global Social Media Platforms Market Trend Analysis
- Global Social Media Platforms Market Size (Volume and Value) Forecast 2019-2025
- Marketing Channel

Direct Marketing

- Indirect Marketing
- Social Media Platforms Customers
- Market Dynamics

Market Trends

- Opportunities
- Market Drivers
- Challenges
- Influence Factors
- Methodology/Research Approach

Research Programs/Design

- Market Size Estimation
- Market Breakdown and Data Triangulation
- Data Source

1.3 Literature Survey

As with any other business investment, Social Media Strategies need to be evaluated to see whether they are achieving the company's primary goals. As a result, more and more companies are focused on social media business analysis. They are looking for better ways to measure, monitor and improve their Social Media Impact in attracting and retaining customers. Here are just a few of the measurements that are commonly used to track social media effectiveness:

Monitoring the conversations of a company’s products or brand, or competition on social media. When and where did the conversation happen? What triggered it? Who is talking? What was said? Was it positive or negative?

Website Analytics:
Google Analytics is the optimal tool for measuring visitors, conversion rates, repeat visits, and time spent on site.

Campaign management:
Measure the effectiveness of social media campaigns and advertising.

Customer service:
Satisfaction, brand loyalty, product/service awareness.

Search engine optimization:
Effectiveness of SEO and level of organic visibility. Sales: Increased and shortened sales cycle.

Preliminary data from the field suggests that social media is becoming a powerful addition to the health communicators' toolkit. Although there is a great deal of interest in using social media as a tool for public health communications, the research evaluating its utility is still in its infancy. As of yet, few research studies have examined the broader utility of social media for the adoption of health promoting and protective behaviours. One of the chief conclusions of this report is that there is a paucity of peer-reviewed studies testing the utility of social media interventions for desired outcomes. Instead, research has focused on documenting the range of health-related behaviours and the content of health-related discourse on these platforms. In 2010, Ford gave 100 independent agents (social media influencers) a free Ford Fiesta at a cost of $5 million. Those social media influencers included bloggers, filmmakers and others who were in the public eye. Each of these influencers began to share their experiences online about driving a Ford Fiesta. Within a relatively short period of time, Fiesta reach 60% recognition before the cars even made it to the showroom. For the price of $5 million it was a bargain campaign compared to traditional advertising.

Observational studies show an abundance of both informal health conversations related to public health issues and organized health-related activities on leading social media platforms such as YouTube, Twitter, and Facebook. The quality of health information available to users on these platforms is highly variable raising some concerns that social media users are exposed to unopposed viewpoints that counter core public health recommendations and contemporary medical science, such as those opposing immunization and promoting smoking. Social media is currently utilized by public health organizations both as a broadcasting platform to amplify messages from traditional media sources (e.g., radio, television, print media) and as an entirely new way of collaborating and co-creating content with target audiences. In the latter approach, organizations have had to adapt their communications strategies to incorporate user generated content and feedback.

The process of engaging users to co-create content, to rate, rank and comment on communications, more so than the resulting message, is increasingly perceived to give a heightened authenticity to messages, improving trust in, and building users' relationships with, organizations. Social media, unlike traditional media campaigns, provides novel opportunities to embed and interject public health messaging into the daily online conversations of Canadians. In the future, it will also allow public health communicators to deliver a range of health promotion messages and self-monitoring tools through mobile applications, an innovation that will potentially increase the reach to those without computers, and will allow public health messaging to penetrate the day-to-day health conversations and activities of Canadians. The adoption of social media by leading public health organizations reflects a widespread sense that these tools are increasingly necessary to reach demographics who are abandoning traditional broadcast technologies (e.g., telephones, television) such as teens, or a significant portion of the public who are rapidly transforming the manner in which they interact with experts.

The term "social media" is used somewhat loosely to describe an array of new Web 2.0 platforms. Although they are not always clearly distinguished in the literature, the interactivity associated with "social media" should be differentiated from more generalized forms of online user engagement. For instance, many websites invite users to input their own information, customize the layout and look of a page, prioritize certain kinds of content, or keep track of their own online activities over time. Social media, by contrast, is characterized by interactivity across multiple horizontal connections, which produce in aggregate a mutable, collectively generated user experience (11, see also Appendix 2 for a description of the most popular social media platforms). Even within a single platform, users make use, to varying degrees, of the opportunities afforded for collaboration and social networking. YouTube, for instance, can be used simply as a broadcast medium for propagating a movie trailer or public service ad. It is only when other users begin to link to, remix, repurpose, and discuss posted content that YouTube's character as a social media platform comes fully into view.

Social media platforms are being studied by health researchers and mobilized for a variety of purposes: recruitment for clinical trials; professional development and training for clinicians; inter-professional communication and coordination; training simulations; health social networks and health and illness support groups; health advocacy and fundraising for health organizations; development of interactive, self-management tools and plugins to popular social media platforms; public health messaging; infectious disease monitoring.

This report targets original research, case studies, reviews, and commentaries related to public health communication, although there are at times significant overlaps between this subfield and those listed above. In addition, we summarize information from online sources related to notable public health campaigns (extracted from podcasts, interviews, PowerPoint presentations, and key public health organizations' websites).

1.4 Problem Statement

Social Trend Analyzer that will analyze the different keywords of twitter and give a descriptive analysis along with statistics of the trend. Business and brands have realized that they need to do social and do it well. So they create a social media marketing strategy and then they start implementing the tactics to achieve the goals.

Problem
It takes a lot of resources. That means time, money and people. It needs creativity and inspiration. It requires a long term commitment. Succeeding with social media is not a get rich quick scheme. Creating quality content takes skill, experience and expertise. Building tribes, followers and fans takes focused attention and engagement. Obtaining 10,000 Twitter followers or 20,000 Facebook likes requires serious investment.

Background
Social media marketing is a consistent and continual treadmill. Creating content needs to be done every day. It then needs to be published. Then monitoring the engagement and responding to Facebook comments and Twitter streams needs to be attended to. Making sure that the person responsible for responding to social platform feedback does it with the right tone and voice is never ending. Often outsourcing it or asking the intern to take it on is not just going to cut it. The reality is that marketing is moving from "campaign marketing" to "continuous marketing". Marketing has now become a big daily commitment. That is a problem. because to do it properly and efficiently, you need the technology and tools to do it at scale.

Relevance
Social media is great news for businesses and brands and the best part is, both big and small brands can benefit. When you know the figures, it's easy to understand why business and consumer marketers almost unanimously believe that social media is crucial to building a brand. So, when it comes to your social media branding, it's important to invest some thought and resources into it.
According to a 2020 report, social media users have passed the 3.8 billion mark. Impressive, when you consider there are 4.5 billion internet users worldwide. 80% of consumers are more likely to evaluate solutions from the brands they follow on social channels. Brand awareness is cited as the top priority for marketers, and social media channels are a one-to-many solution for getting the word out about your products and services. By creating a strong brand presence on social media, you can reach a broader audience.

Branding is not just about the logo. Nor is it about being a multi-million dollar company. A brand is how people perceive your product, business, or even you as a person. The goal with branding is to make sure that perception is the one you want people to have. Even if you're not actively marketing your brand, that perception will still exist, or worse, won't exist at all, so it's up to you to shape how you want people to feel.

1.5 Scope Study

The research study analyses the global Ai In Social Media industry from 360-degree analysis of the market thoroughly delivering insights into the market for better business decisions, considering multiple aspects some of which are listed below as:

Recent Developments
Market Overview and growth analysis
Import and Export Overview
Volume Analysis
Current Market Trends and Future Outlook
Market Opportunistic and Attractive Investment Segment

Geographic Coverage
North America Market Size and/or Volume
Latin America Market Size and/or Volume

Europe Market Size and/or Volume
Asia-Pacific Market Size and/or Volume
Rest of the world Market Size and/or Volume

This project will be helpful to the firms, companies, government, political parties, social enthusiasts as well as to the common people as it will give a detailed analysis of each and every thread going on. Firms or Brands may get review about their new product on newly released hardware or softwares by using the analysis.

It's going to be a game changer for people in entertainment industry, as it will help them get analysis of GP in the rawest and most detailed way. People can use this to get an exact idea on how other people are thinking about a trend or event or issue. It will help in mass equalization. Elimination of Fake news is also a field we can approach through the results of our analyser. AI in social media market to grow from USD 0.6 billion in 2018 to USD 2.2 billion by 2025, at a Compound Annual Growth Rate (CAGR) of 28% during the forecast period.

It is expected that AI technology adoption for various applications in the social media sector will increase, and the use of AI-enabled smartphones will drive the growth of AI in the social media market. A limited number of AI experts and slow digitization rates are affecting the adoption of AI technologies in emerging economies, which is holding back the growth of the market. The major AI in social media vendors include Google (US), Facebook (US), AWS (US), IBM (US), Adobe Systems (US), Baidu (China), Salesforce (US), Twitter (US), Snap (US), Clara bridge (US), Conversion (US), Sprinklr (US), Unmetrical (US), Sentamu (US), Clue (US),

2- System Analysis

For brands that want to achieve and maintain their relevance on social channels—to lead and not follow markets—social media trend analysis is an essential practice. By leveraging real-time data from social networks, blogs, and forums, firms can collect primary market research data from billions of users around the world.

Using social listening and trend analysis tools together, organizations can monitor the conversations of real people about their brand, products, and services while obtaining actionable insights into the trends influencing their customers and their business. Through social media trend analysis, firms can take the pulse of the social web, using the knowledge they gain to more effectively inspire, engage, and delight online audiences.

Brands can use social media trend analysis to anticipate changes in consumer demographics, attitudes, values, and needs and identify developing local and global consumer trends. They can also detect the emergence of disruptive technologies and new competitors in their industry as well as identify the people and events their customers are interested in and the brands they respect.

Social media trend analysis is carried out easily using social media listening and analytics tools. These tools can be used to collect and analyse data from various social networks and other sources of publicly available user-generated content, transforming billions of mentions into practical insights that brands can use to inform their marketing, sales, customer support, and product development strategies.

More efficient and less costly than using conventional market research methods such as interviews and focus groups, social media trend analysis leverages the easily accessible, unbiased opinions of real people talking openly and honestly about the things that matter to them on the social web. Firms can use social listening tools to monitor these conversations and discover emerging trends, viral topics and videos, and other information useful for creating content and messaging that strikes a chord with consumers. Brands can monitor and track consumer perception of their brand and their competitors, obtain valuable feedback about their products and services, and become aware of important changes in consumer needs, habits, and expectations.

Another way firms can use social media trend analysis is to learn more about their consumers—their interests, preferences, and behaviours—the publications they read, the places they travel to, their favourite brands of beer or coffee. By using an audience insights tool, organizations can access detailed demographic, psychographic, and behavioural information about the groups they want to reach. They can explore how the likes and dislikes and values and habits of their audience change over time, using these insights to develop content that speaks directly to their target groups. By leveraging audience insights, firms can better engage consumers on social channels and optimize and boost the ROI of their marketing campaigns.

2.1 General

Sentiment Analysis is a technique widely used in text mining. Twitter Sentiment Analysis, therefore means, using advanced text mining techniques to analyse the sentiment of the text (here, tweet) in the form of positive, negative and neutral. It is also known as Opinion Mining, is primarily for analysing conversations, opinions, and sharing of views (all in the form of tweets) for deciding business strategy, political analysis, and also for assessing public actions. It may, therefore, be described as a text mining technique for analysing the underlying sentiment of a text message, i.e., a tweet. Twitter sentiment or opinion expressed through it may be positive, negative or neutral. However, no algorithm can give you 100% accuracy or prediction on sentiment analysis.

Ingenuity, Revealed Context, steam crab, Meaning Cloud, and Social Mention are some of the well-known tools used for the analysis of Twitter sentiment. R and Python are widely used for sentiment analysis dataset twitter. Sentiment Analysis of Twitter data is now much more than a college project or a certification program. A good number of Tutorials related to Twitter sentiment are available for educating students on the Twitter sentiment analysis project report and its usage with R and Python. You may also enrol for a python tutorial for the same program to get a promising career in sentiment analysis dataset twitter.

Our discussion will include, Twitter Sentiment Analysis in R and Python, and also throw light on its techniques and teach you how to generate the Twitter Sentiment Analysis project report, and the advantages of enrolling for its Tutorial.

As a part of Natural Language Processing, algorithms like SVM, Naive Bayes is used in predicting the polarity of the sentence. sentiment analysis of Twitter data may also depend upon sentence level and document level.

Methods like, positive and negative words to find on the sentence is however inappropriate, because the flavour of the text block depends a lot on the context. This may be done by looking at the POS (Part of Speech) Tagging.

2.2 Preliminary Investigation

It is important to note that some of the original changes to digital use in the public health field happened because of the rapid integration of SMS-enabled cellular devices — it is estimated there are approximately 3.6 billion people (of our roughly seven billion population globally) who have some unique access to a mobile device. However, only around 40 percent of that 3.6 billion have ever had access to the internet (via any device). Thus, it is important to note that SMS, especially in developing nations, is still the primary means of communicating between cellular devices. Public health campaigns have successfully integrated texting as part of their approach to communication, health education and response. Examples of these include Crisis Text Line, designed to support people in crisis; cellular service Orange's partnership with the Cameroon Ministry of Health to launch a preventive health texting system called My Healthline; and the integration of SMS reporting capabilities into crowdsourced platforms (like Ushahidi/Crowd Map) that lets users report how pollution affects their environment and health conditions like the witness Pollution Map.

Social media has influenced public and personal health spaces recently in two particular ways:

Social media can both help facilitate information sharing and be problematic in spreading rumours during normal (or seasonally expected) health events and health crises. Public health agencies and other organizations can use social media to disseminate time-sensitive health information, promote information sharing to encourage behavioural changes (including corrective changes during potential health crises), be a platform for conversation between agencies and constituents (rather than just as an information provider) and allow the public to provide useful information and feedback.

However, social media is a two-way street, and allows non-experts to share information just as rapidly as health agencies, if not more so. Managing misinformation during health crises is an important role that health agencies and other organizations have been forced to take on during events such as the 2014 Ebola crisis. Thus, it is crucial that public health agencies and organizations are equipped before a crisis with strategies, educational material, messages and an appropriate staff or volunteer social media management plan to counter misinformation.

Another consideration is the acceleration of available "mobile health (mHealth)" devices such as wearables, apps and social media platforms for tracking both personal health and allowing healthcare practitioners to track patients remotely. This ability to track personal health, check in remotely with your physician, and partake in competitive fitness and health regimes could lead to a re-imagination of how we experience our personal health daily (“have to get those 10,000 steps in!”), also creating predictive and preventative monitoring programs for outpatient care. In contrast, the open sharing of health data has prompted industry-wide questions about privacy, and begs for a deeper consideration of how both sensitive personal health information and patient information can be protected if a platform is hacked or a wearable device is lost.

While social media has manifold applications in public health, it has also been used by physicians and other health professionals to achieve important objectives in clinical practice, education and research. Examples of beneficial applications include:

● Improving practice efficiency by providing educational videos on general topics such as vaccine safety in advance of scheduled appointments, enabling deeper and more meaningful face-to-face discussions to address particular patient questions.

● Making subspecialty expertise on rare diseases and conditions available broadly via video, giving patients new access to information while also favourably positioning the expert.

● Facilitating patient-to-patient support groups that would be impractical without social networking platforms that overcome barriers of time and space.

● Engaging medical professionals in online discussions, providing evidence-based perspectives on current public health challenges.

● Recruiting subjects for clinical trials and improving the informed consent process.

● Providing continuing education asynchronously and on a global scale through online learning communities.

Key concerns relating to social media in healthcare include:

● Ensuring compliance with standards of online professionalism and patient privacy protection.

● Active dissemination of medical myths and misinformation by self-interested propagandists.

● The proliferation of physician and hospital rating and review sites, which often lack sufficient patient participation to provide an accurate reflection of satisfaction with services.

Thankfully, the opportunities for beneficial applications also provide the best ways to mitigate concerns. For example, hospital and physician participation in social media platforms helps them to manage their reputation by elevating their own content in search results, while also reducing the influence of misinformation. And training in digital professionalism is a prime candidate for delivery via online social collaboration platforms.

2.3 Feasibility Study

Feasibility Study is defined as the assessment of a project that has been proposed by someone. It is analysis and evaluation of a system and hence is also popularly known as feasibility analysis. The word feasibility study is used to describe the do-ability factor of a task and focuses on the all-important question of whether we should continue with it or not.

A feasibility study is a determining factor that evaluates whether the project is financially and technically sound and viable. Does it make a good business sense and is it within the feasibility of estimated cost is an important question that is answered by this concept. The initial designing stage of a task or a project that brings together all the elements of knowledge and estimate the level of expertise needed for it is conducted via a feasibility study. It also offers a quantitative and qualitative assessment of other vital resources, timetable, cost estimate, and identification of critical points.

The feasibility study is thus an analysis that is carried out by organizations to determine how successful a project is going to be. It is generally conducted when a large amount of money is at stake. A good feasibility study in its report mentions

- ✓ Legal requirements
- ✓ Financial data
- ✓ Policies and research data
- ✓ Tax implications and obligations
- ✓ Historical Background of the project
- ✓ A detailed description of the project
- ✓ Details of management
- ✓ Details of all the operations
- ✓ Accounting statement

2.3.1 Technical Feasibility

An appraisal exercise intimately connected with the evaluation of environmental feasibility is the assessment of the project’s impact on the lives of people that live and work in the project’s area of influence.

The social impact analysis (or social feasibility assessment) can be a very important part of the general appraisal of PPP projects, since many infrastructure initiatives cause severe adverse impacts on communities surrounding the site on which they are implemented.

Social impact analysis is an exercise aimed at identifying and analysing such impacts in order to understand the scale and reach of the project’s social impacts. It also ensures that these impacts are mitigated, to the extent possible, and fully considered in the green light decision.

Social impact analysis greatly reduces the overall risks of the project, as it helps to reduce resistance, strengthens general support, and allows for a more comprehensive understanding of the costs and benefits of the project.

However, social impact analysis can be expensive and time consuming, so the full analysis process cannot be justified for all projects. At a minimum, all projects demand a review of project data at the Appraisal Phase, so as to identify if material social impacts exist. If they do, a full social impact analysis should be conducted.

2.3.2 Economical Feasibility

Costs and benefits of the proposed computer system must always be considered together, because they are interrelated and often interdependent. Although the systems analyst is trying to propose a system that fulfills various information requirements, decisions to continue with the proposed system will be based on a cost-benefit analysis, not on information requirements. In many ways, benefits are measured by costs, as becomes apparent in the next section.

Systems analysts are required to predict certain key variables before the proposal is submitted to the client. To some degree, a systems analyst will rely on a what-if analysis, such as, “What if labor costs rise only 5 percent per year for the next three years, rather than 10 percent?” The systems analyst should realize, however, that he or she cannot rely on what-if analysis for everything if the proposal is to be credible, meaningful, and valuable.

The systems analyst has many forecasting models available. The main condition for choosing a model is the availability of historical data. If they are unavailable, the analyst must turn to one of the judgment methods: estimates from the sales force, surveys to estimate customer demand, Delphi studies (a consensus forecast developed independently by a group of experts through a series of iterations), creating scenarios, or drawing historical analogies.

If historical data are available, the next differentiation between classes of techniques involves whether the forecast is conditional or unconditional. Conditional implies that there is an association among variables in the model or that such a causal relationship exists. Common methods in this group include correlation, regression, leading indicators, econometrics, and input/output models.

Unconditional forecasting means the analyst isn’t required to find or identify any causal relationships. Consequently, systems analysts find that these methods are low-cost, easy-to-implement alternatives. Included in this group are graphical judgment, moving averages, and analysis of time-series data. Because these methods are simple, reliable, and cost effective, the remainder of the section focuses on them.

Trends can be estimated in a number of different ways. One way to estimate trends is to use a moving average. This method is useful because some seasonal, cyclical, or random patterns may be smoothed, leaving the trend pattern. The principle behind moving averages is to calculate the arithmetic mean of data from a fixed number of periods; a three-month moving average is simply the average of the last three months. For example, the average sales for

January, February, and March is used to predict the sales for April. Then the average sales for February, March, and April are used to predict the sales for May, and so on.

When the results are graphed, it is easily noticeable that the widely fluctuating data are smoothed. The moving average method is useful for its smoothing ability, but at the same time it has many disadvantages. Moving averages are more strongly affected by extreme values than by using graphical judgment or estimating using other methods such as least squares. The analyst should learn forecasting well, as it often provides information valuable in justifying the entire project.

Benefits and costs can be thought of as either tangible or intangible. Both tangible and intangible benefits and costs must be taken into account when systems are considered.

Tangible benefits are advantages measurable in dollars that accrue to the organization through the use of the information system. Examples of tangible benefits are an increase in the speed of processing, access to otherwise inaccessible information, access to information on a more timely basis than was possible before, the advantage of the computer's superior calculating power, and decreases in the amount of employee time needed to complete specific tasks. There are still others. Although measurement is not always easy, tangible benefits can actually be measured in terms of dollars,

Some benefits that accrue to the organization from the use of the information system are difficult to measure but are important nonetheless. They are known as intangible benefits.

Intangible benefits include improving the decision-making process, enhancing accuracy, becoming more competitive in customer service, maintaining a good business image, and increasing job satisfaction for employees by eliminating tedious tasks. As you can judge from the list given, intangible benefits are extremely important and can have far-reaching implications for the business as it relates to people both outside and within the organization.

Although intangible benefits of an information system are important factors that must be considered when deciding whether to proceed with a system, a system built solely for its intangible benefits will not be successful. You must discuss both tangible and intangible benefits in your proposal, because presenting both will allow decision makers in the business to make a well informed decision about the proposed system.

The concepts of tangible and intangible costs present a conceptual parallel to the tangible and intangible benefits discussed already. Tangible costs are those that can be accurately projected by the systems analyst and the business's accounting personnel.

Included in tangible costs are the cost of equipment such as computers and terminals, the cost of resources, the cost of systems analysts' time, the cost of programmers' time, and other employees
salaries. These costs are usually well established or can be discovered quite easily, and are the costs that will require a cash outlay of the business.

Intangible costs are difficult to estimate and may not be known. They include losing a competitive edge, losing the reputation for being first with an innovation or the leader in a field, declining company image due to increased customer dissatisfaction, and ineffective decision making due to untimely or inaccessible information. As you can imagine, it is next to impossible to project a dollar amount for intangible costs accurately. To aid decision makers who want to weigh the proposed system and all its implications, you must include intangible costs even though they are not quantifiable.

2.3.2 Operational Feasibility

Costs and benefits of the proposed computer system must always be considered together, because they are interrelated and often interdependent. Although the systems analyst is trying to propose a system that fulfills various information requirements, decisions to continue with the proposed system will be based on a cost-benefit analysis, not on information requirements. In many ways, benefits are measured by costs, as becomes apparent in the next section.

Systems analysts are required to predict certain key variables before the proposal is submitted to the client. To some degree, a systems analyst will rely on a what-if analysis, such as, “What if labor costs rise only 5 percent per year for the next three years, rather than 10 percent?” The systems analyst should realize, however, that he or she cannot rely on what-if analysis for everything if the proposal is to be credible, meaningful, and valuable.

The systems analyst has many forecasting models available. The main condition for choosing a model is the availability of historical data. If they are unavailable, the analyst must turn to one of the judgment methods: estimates from the sales force, surveys to estimate customer demand, Delphi studies (a consensus forecast developed independently by a group of experts through a series of iterations), creating scenarios, or drawing historical analogies.

If historical data are available, the next differentiation between classes of techniques involves whether the forecast is conditional or unconditional. Conditional implies that there is an association among variables in the model or that such a causal relationship exists. Common methods in this group include correlation, regression, leading indicators, econometrics, and input/output models.

Unconditional forecasting means the analyst isn’t required to find or identify any causal relationships. Consequently, systems analysts find that these methods are low-cost, easy-to-implement alternatives. Included in this group are graphical judgment, moving averages, and analysis of time-series data. Because these methods are simple, reliable, and cost effective, the remainder of the section focuses on them.

Trends can be estimated in a number of different ways. One way to estimate trends is to use a moving average. This method is useful because some seasonal, cyclical, or random patterns may be smoothed, leaving the trend pattern. The principle behind moving averages is to calculate the arithmetic mean of data from a fixed number of periods; a three-month moving average is simply the average of the last three months. For example, the average sales for January, February, and March is used to predict the sales for April. Then the average sales for February, March, and April are used to predict the sales for May, and so on.

When the results are graphed, it is easily noticeable that the widely fluctuating data are smoothed. The moving average method is useful for its smoothing ability, but at the same time it has many disadvantages. Moving averages are more strongly affected by extreme values than by using graphical judgment or estimating using other methods such as least squares. The analyst should learn forecasting well, as it often provides information valuable in justifying the entire project.

Benefits and costs can be thought of as either tangible or intangible. Both tangible and intangible benefits and costs must be taken into account when systems are considered.

Tangible benefits are advantages measurable in dollars that accrue to the organization through the use of the information system. Examples of tangible benefits are an increase in the speed of processing, access to otherwise inaccessible information, access to information on a more timely basis than was possible before, the advantage of the computer’s superior calculating power, and decreases in the amount of employee time needed to complete specific tasks. There are still others. Although measurement is not always easy, tangible benefits can actually be measured in terms of dollars,

Some benefits that accrue to the organization from the use of the information system are difficult to measure but are important nonetheless. They are known as intangible benefits.

Intangible benefits include improving the decision-making process, enhancing accuracy, becoming more competitive in customer service, maintaining a good business image, and increasing job satisfaction for employees by eliminating tedious tasks. As you can judge from the list given, intangible benefits are extremely important and can have far-reaching implications for the business as it relates to people both outside and within the organization.

Although intangible benefits of an information system are important factors that must be considered when deciding whether to proceed with a system, a system built solely for its intangible benefits will not be successful. You must discuss both tangible and intangible benefits in your proposal, because presenting both will allow decision makers in the business to make a well informed decision about the proposed system.

The concepts of tangible and intangible costs present a conceptual parallel to the tangible and intangible benefits discussed already. Tangible costs are those that can be accurately projected by the systems analyst and the business's accounting personnel.

Included in tangible costs are the cost of equipment such as computers and terminals, the cost of resources, the cost of systems analysts' time, the cost of programmers' time, and other employees
salaries. These costs are usually well established or can be discovered quite easily, and are the costs that will require a cash outlay of the business.

Intangible costs are difficult to estimate and may not be known. They include losing a competitive edge, losing the reputation for being first with an innovation or the leader in a field, declining company image due to increased customer dissatisfaction, and ineffective decision making due to untimely or inaccessible information. As you can imagine, it is next to impossible to project a dollar amount for intangible costs accurately. To aid decision makers who want to weigh the proposed system and all its implications, you must include intangible costs even though they are not quantifiable.

2.4 Software Specification

Python 3.6
Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.
Python was created in the late 1980s, and first released in 1991, by Guido van Rossum as a successor to the ABC programming language. Python 2.0, released in 2000, introduced new features, such as list comprehensions, and a garbage collection system with reference counting, and was discontinued with version 2.7 in 2020. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible and much Python 2 code does not run unmodified on Python 3. With Python 2's end-of-life, only Python 3.6.x and later are supported, with older versions still supporting e.g. Windows 7 (and old installers not restricted to 64-bit Windows).

Python interpreters are supported for mainstream operating systems and available for a few more (and in the past supported many more). A global community of programmers develops and maintains CPython, a free and open-source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming[51] and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management.[56] It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter, map, and reduce functions; list comprehensions, dictionaries, sets, and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Readability counts.
Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it" design philosophy. Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture."

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonistas

Tweepy
Tweepy supports accessing Twitter via Basic Authentication and the newer method, OAuth. Twitter has stopped accepting Basic Authentication so OAuth is now the only way to use the Twitter API.

The main difference between Basic and OAuth authentication are the consumer and access keys. With Basic Authentication, it was possible to provide a username and password and access the API, but since 2010 when the Twitter started requiring OAuth, the process is a bit more complicated. An app has to be created at dev.twitter.com.

OAuth is a bit more complicated initially than Basic Auth, since it requires more effort, but the benefits it offers are very lucrative:

Tweets can be customized to have a string which identifies the app which was used.
It doesn't reveal user password, making it more secure.
It's easier to manage the permissions, for example a set of tokens and keys can be generated that only allows reading from the timelines, so in case someone obtains those credentials, he/she won't be able to write or send direct messages, minimizing the risk.
The application doesn't reply on a password, so even if the user changes it, the application will still work.

One of the main usage cases of tweepy is monitoring for tweets and doing actions when some event happens. Key component of that is the StreamListener object, which monitors tweets in real time and catches them.

Tweepy is a great open-source library which provides access to the Twitter API for Python. Although the documentation for tweepy is a bit scarce and doesn't have many examples, the fact that it heavily relies on the Twitter API, which has excellent documentation, makes it probably the best Twitter library for Python, especially when considering the Streaming API support, which is where tweepy excels. Other libraries like python-twitter provide many functions too, but the tweepy has most active community and most commits to the code in the last year.

Textblob
TextBlob is a Python (2 and 3) library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more. TextBlob is built upon NLTK and provides an easy to use interface to the NLTK library.

Twitter Apps
Twitter is an American microblogging and social networking service on which users post and interact with messages known as "tweets". Registered users can post, like and retweet tweets, but unregistered users can only read them. Users access Twitter through its website interface, through Short Message Service (SMS) or its mobile-device application software ("app"). Twitter, Inc. is based in San Francisco, California, and has more than 25 offices around the world. Tweets were originally restricted to 140 characters, but was doubled to 280 for non-CJK languages in November 2017.Audio and video tweets remain limited to 140 seconds for most accounts.

Twitter was created by Jack Dorsey, Noah Glass, Biz Stone, and Evan Williams in March 2006 and launched in July of that year. By 2012, more than 100 million users posted 340 million tweets a day, and the service handled an average of 1.6 billion search queries per day. In 2013, it was one of the ten most-visited websites and has been described as "the SMS of the Internet". As of 2018, Twitter had more than 321 million monthly active users.

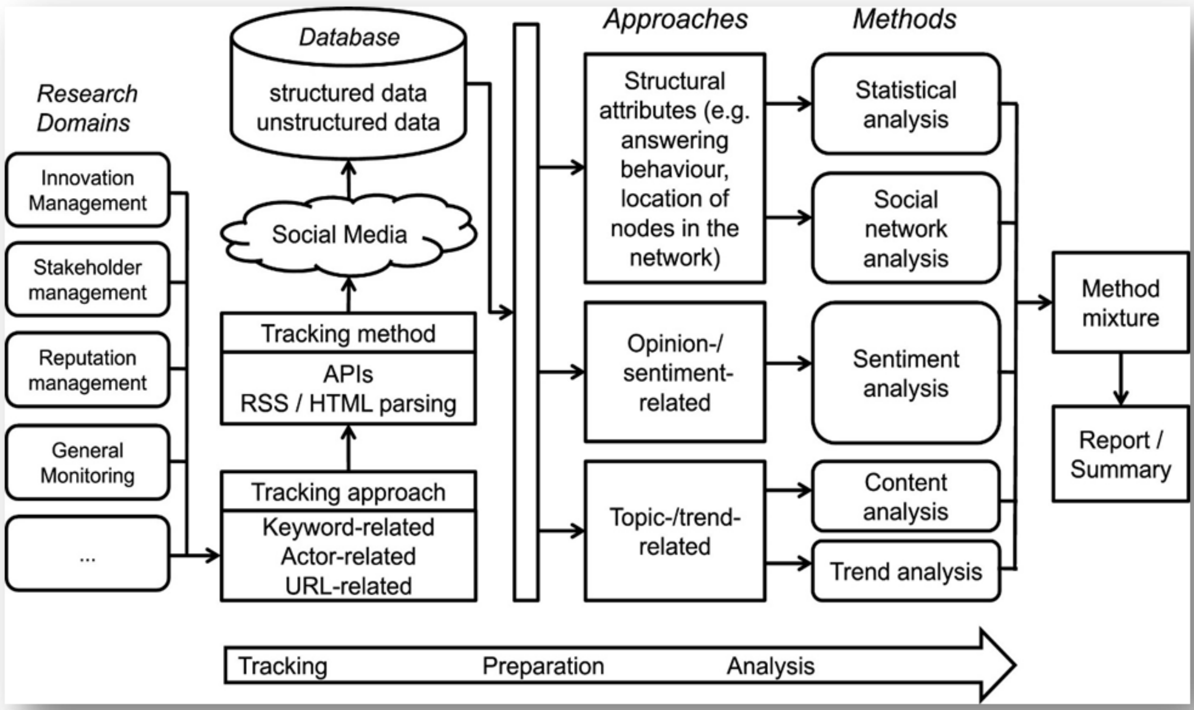
The first Twitter prototype, developed by Dorsey and contractor Florian Weber, was used as an internal service for Odeo employees. The full version was introduced publicly on July 15, 2006. In October 2006, Biz Stone, Evan Williams, Dorsey, and other members of Odeo formed Obvious Corporation and acquired Odeo, together with its assets — including Odeo.com and Twitter.com — from the investors and shareholders. Williams fired Glass, who was silent about his part in Twitter's startup until 2011. Twitter spun off into its own company in April 2007.

Tweets are publicly visible by default, but senders can restrict message delivery to only their followers. Users can mute users they do not wish to interact with and block accounts from viewing their tweets. Users can tweet via the Twitter website, compatible external applications (such as for smartphones), or by Short Message Service (SMS) available in certain countries. Users may subscribe to other users' tweets—this is known as "following" and subscribers are known as "followers" or "tweeps", a portmanteau of Twitter and peeps.[131] Individual tweets can be forwarded by other users to their own feed, a process known as a "retweet". Users can also "like" (formerly "favorite") individual tweets. Twitter allows users to update their profile via their mobile phone either by text messaging or by apps released for certain smartphones and tablets. Twitter has been compared to a web-based Internet Relay Chat (IRC) client.

According to research published in April 2014, around 44% of user accounts have never tweeted. San Antonio-based market-research firm Pear Analytics analyzed 2,000 tweets (originating from the United States and in English) over a two-week period in August 2009 from 11:00 am to 5:00 pm (CST) and separated them into six categories.[139] Pointless babble made up 40%, with 38% being conversational. Pass-along value had 9%, self-promotion 6% with spam and news each making 4%.

Despite Jack Dorsey's own open contention that a message on Twitter is "a short burst of inconsequential information", social networking researcher danah boyd responded to the Pear Analytics survey by arguing that what the Pear researchers labeled "pointless babble" is better characterized as "social grooming" and/or "peripheral awareness" (which she justifies as persons "want[ing] to know what the people around them are thinking and doing and feeling, even when co-presence isn't viable"). Similarly, a survey of Twitter users found that a more specific social role of passing along messages that include a hyperlink is an expectation of reciprocal linking by followers. The first tweet was posted by Jack Dorsey (creator) at 12:50 PM PST on March 21, 2006 and read "just setting up my twttr". In 2009, the first tweet was sent from space. US astronauts Nicola Stott and Jeff Williams took part in a live 'tweetup' from the International Space Station with around 35 members of the public at NASA Headquarters, Washington, DC.

2.5 Data Flow Diagram



3- System Design

Definition: Systems design is the process of defining elements of a system like modules, architecture, components and their interfaces and data for a system based on the specified requirements. It is the process of defining, developing and designing systems which satisfies the specific needs and requirements of a business or organization.

Description: A systemic approach is required for a coherent and well-running system. Bottom-Up or Top-Down approach is required to take into account all related variables of the system. A designer uses the modelling languages to express the information and knowledge in a structure of system that is defined by a consistent set of rules and definitions. The designs can be defined in graphical or textual modelling languages.

Some of the examples of graphical modelling languages are

a. Unified Modelling Language (UML): To describe software both structurally and behaviourally with graphical notation.

b. Flowchart: A schematic or stepwise representation of an algorithm.

c. Business Process Modelling Notation (BPMN): Used for Process Modelling language.

d. Systems Modelling Language (Sims): Used for systems engineering.

3.1 Design Methodology

Design methods:

1) Architectural design: To describes the views, models, behaviour, and structure of the system.

2) Logical design: To represent the data flow, inputs and outputs of the system. Example: ER Diagrams (Entity Relationship Diagrams).

3) Physical design: Defined as a) How users add information to the system and how the system represents information back to the user. b) How the data is modelled and stored within the system. c) How data moves through the system, how data is validated, secured and/or transformed as it flows through and out of the system.

The development of design methods has been closely associated with prescriptions for a systematic process of designing. These process models usually comprise a number of phases or stages, beginning with a statement or recognition of a problem or a need for a new design and culminating in a finalised solution proposal. In his 'Systematic Method for Designers produced a very elaborate, 229 step model of a systematic design process for industrial design, but also a summary model consisting of three phases: Analytical phase (programming and data collection, analysis), Creative phase (synthesis, development), and Executive phase (communication). The UK's design models the creative design process in four phases: Discover (insight into the problem), Define (the area to focus upon), Develop (potential solutions), Deliver (solutions that work). A systematic model for engineering design by Phal and Beit has phases of Clarification of the task, Conceptual design, Embodiment design, and Detail design. A less prescriptive approach to designing a basic design process for oneself has been outlined by

3.2 User Interface Design

User interfaces are the access points where users interact with designs. They come in three formats:

Graphical user interfaces (GUIs)—Users interact with visual representations on digital control panels. A computer's desktop is a GUI.

Voice-controlled interfaces (VUIs)—Users interact with these through their voices. Most smart assistants—e.g., Siri on iPhone and Alexa on Amazon devices—are VUIs.

Gesture-based interfaces—Users engage with 3D design spaces through bodily motions: e.g., in virtual reality (VR) games.

To design UIs best, you should consider:

Users judge designs quickly and care about usability and likeability.

They don't care about *your design*, but about *getting their tasks done* easily and with minimum effort.

Your design should therefore be "invisible": Users shouldn't focus on it but on completing tasks: e.g., ordering pizza on Domino's Zero Click app.

So, understand your users' contexts and task flows (which you can find from, e.g., customer journey maps), to fine-tune the best, most intuitive UIs that deliver seamless experiences.

UIs should also be enjoyable (or at least satisfying and frustration-free).

When your design predicts users' needs, they can enjoy more personalized and immersive experiences. Delight them, and they'll keep returning.

Where appropriate, elements of gamification can make your design more fun.

UIs should communicate brand values and reinforce users' trust.

Good design is emotional design. Users associate good feelings with brands that speak to them at all levels and keep the magic of pleasurable, seamless experiences alive.

How to make Great UIs

To deliver impressive GUIs, remember—**users are humans, with needs such as comfort and a limit on their mental capacities**. You should follow these guidelines:

1. Make buttons and other common elements perform predictably (including responses such as pinch-to-zoom) so users can unconsciously use them everywhere. Form should follow function.
2. Maintain high discoverability. Clearly label icons and include well-indicated affordances: e.g., shadows for buttons.
3. Keep interfaces simple (with only elements that help serve users' purposes) and create an "invisible" feel.
4. Respect the user's eye and attention regarding layout. Focus on hierarchy and readability:
5. Use proper alignment. Typically choose edge (over centre) alignment.
6. Draw attention to key features using:
7. Colour, brightness and contrast. Avoid including colours or buttons excessively.
8. Text via font sizes, bold type/weighting, italics, capitals and distance between letters. Users should pick up meanings just by scanning.
9. Minimize the number of actions for performing tasks but focus on one chief function per page. Guide users by indicating preferred actions. Ease complex tasks by using progressive disclosure.
10. Put controls near objects that users want to control. For example, a button to submit a form should be near the form.
11. Keep users informed regarding system responses/actions with feedback.
12. Use appropriate UI design patterns to help guide users and reduce burdens (e.g., pre-fill forms). Beware of using dark patterns, which include hard-to-see prefilled opt-in/opt-out checkboxes and sneaking items into users' carts.
13. Maintain brand consistency.
14. Always provide next steps which users can deduce naturally, whatever their context.

Requirements

The dynamic characteristics of a system are described in terms of the dialogue requirements contained in seven principles of part 10 of the ergonomics standard. This standard establishes a framework of ergonomic "principles" for the dialogue techniques with high-level definitions and illustrative applications and examples of the principles. The principles of the dialogue represent the dynamic aspects of the interface and can be mostly regarded as the "feel" of the interface. The seven dialogue principles are:

- Suitability for the task: the dialogue is suitable for a task when it supports the user in the effective and efficient completion of the task.
- Self-descriptiveness: the dialogue is self-descriptive when each dialogue step is immediately comprehensible through feedback from the system or is explained to the user on request.
- Controllability: the dialogue is controllable when the user is able to initiate and control the direction and pace of the interaction until the point at which the goal has been met.
- Conformity with user expectations: the dialogue conforms with user expectations when it is consistent and corresponds to the user characteristics, such as task knowledge, education, experience, and to commonly accepted conventions.

- Error tolerance: the dialogue is error-tolerant if, despite evident errors in input, the intended result may be achieved with either no or minimal action by the user.
- Suitability for individualization: the dialogue is capable of individualization when the interface software can be modified to suit the task needs, individual preferences, and skills of the user.
- Suitability for learning: the dialogue is suitable for learning when it supports and guides the user in learning to use the system.
- The concept of usability is defined of the ISO 9241 standard by effectiveness, efficiency, and satisfaction of the user. Part 11 gives the following definition of usability:
- Usability is measured by the extent to which the intended goals of use of the overall system are achieved (effectiveness).
- The resources that have to be expended to achieve the intended goals (efficiency).
- The extent to which the user finds the overall system acceptable (satisfaction).

Effectiveness, efficiency, and satisfaction can be seen as quality factors of usability. To evaluate these factors, they need to be decomposed into sub-factors, and finally, into usability measures.

The information presented is described in Part 12 of the ISO 9241 standard for the organization of information (arrangement, alignment, grouping, labels, location), for the display of graphical objects, and for the coding of information (abbreviation, colour, size, shape, visual cues) by seven attributes. The "attributes of presented information" represent the static aspects of the interface and can be generally regarded as the "look" of the interface. The attributes are detailed in the recommendations given in the standard. Each of the recommendations supports one or more of the seven attributes. The seven presentation attributes are:

- Clarity: the information content is conveyed quickly and accurately.
- Discriminability: the displayed information can be distinguished accurately.
- Conciseness: users are not overloaded with extraneous information.
- Consistency: a unique design, conformity with user's expectation.
- Detectability: the user's attention is directed towards information required.
- Legibility: information is easy to read.
- Comprehensibility: the meaning is clearly understandable, unambiguous, interpretable, and recognizable.
- The user guidance in Part 13 of the ISO 9241 standard describes that the user guidance information should be readily distinguishable from other displayed information and should be specific for the current context of use. User guidance can be given by the following five means:
- Prompts indicating explicitly (specific prompts) or implicitly (generic prompts) that the system is available for input.
- Feedback informing about the user's input timely, perceptible, and non-intrusive.
- Status information indicating the continuing state of the application, the system's hardware and software components, and the user's activities.
- Error management including error prevention, error correction, user support for error management, and error messages.
- On-line help for system-initiated and user-initiated requests with specific information for the current context of use.

Research

User interface design has been a topic of considerable research, including on its aesthetics.[7] Standards have been developed as far back as the 1980s for defining the usability of software products. One of the structural bases has become the IFIP user interface reference model. The model proposes four dimensions to structure the user interface:

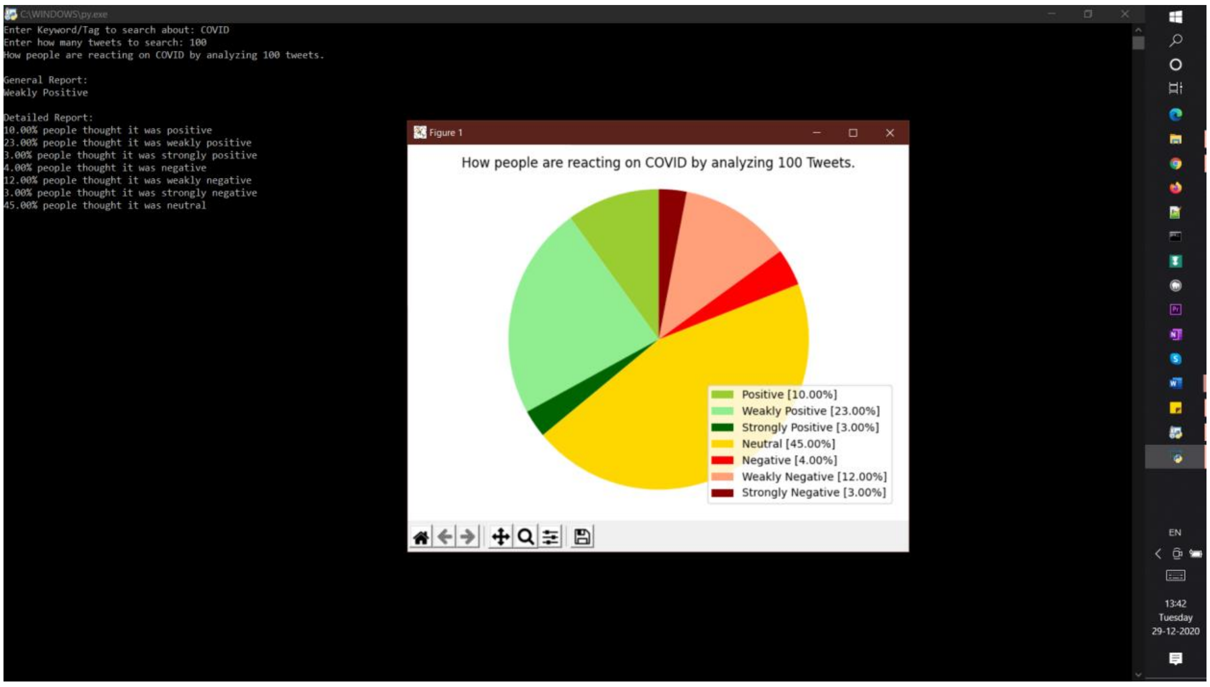
- The input/output dimension (the look)
- The dialogue dimension (the feel)
- The technical or functional dimension (the access to tools and services)
- The organizational dimension (the communication and co-operation support)

This model has greatly influenced the development of the international standard ISO 9241 describing the interface design requirements for usability. The desire to understand application-specific UI issues early in software development, even as an application was being developed, led to research on GUI rapid prototyping tools that might offer convincing simulations of how an actual application might behave in production use.[8] Some of this research has shown that a wide variety of programming tasks for GUI-based software can, in fact, be specified through means other than writing program code.[9]

Research in recent years is strongly motivated by the increasing variety of devices that can, by virtue of Moore's law, host very complex interfaces

4. Test Cases & Output Generated

4.1 COVID

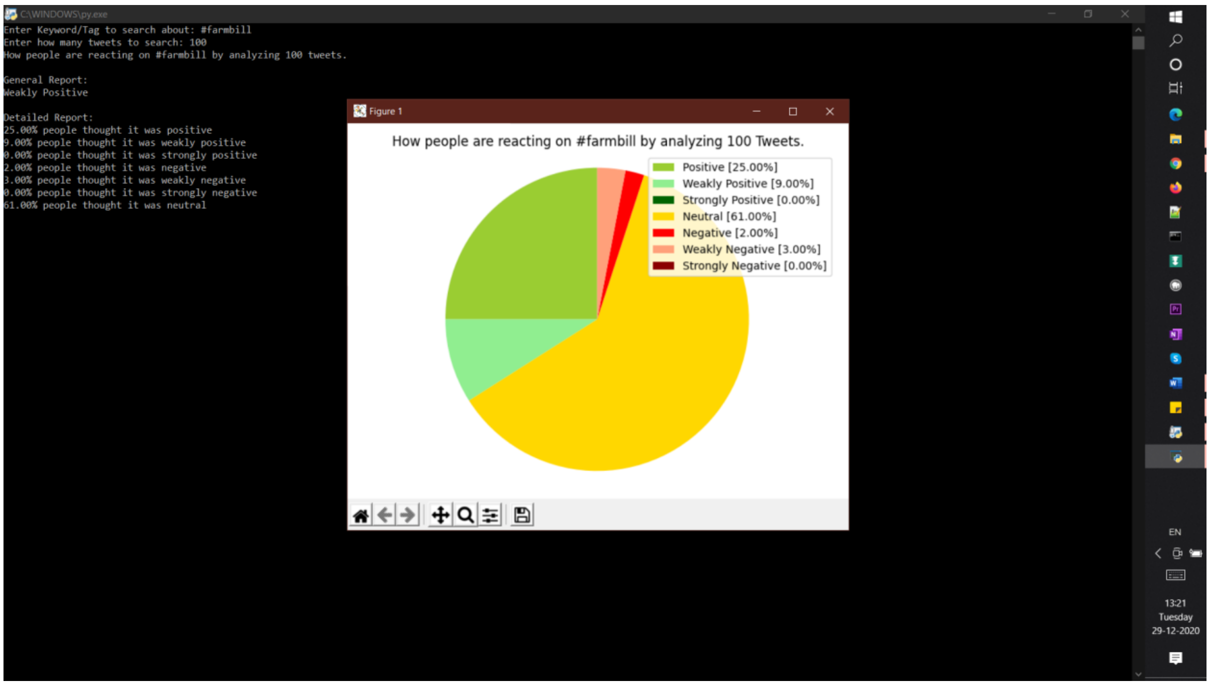


```
Enter Keyword/Tag to search about: COVID
Enter how many tweets to search: 100
How people are reacting on COVID by analyzing 100 tweets.

General Report:
Weakly Positive

Detailed Report:
10.00% people thought it was positive
23.00% people thought it was weakly positive
3.00% people thought it was strongly positive
4.00% people thought it was negative
12.00% people thought it was weakly negative
3.00% people thought it was strongly negative
45.00% people thought it was neutral
```

4.2 Farm Bill

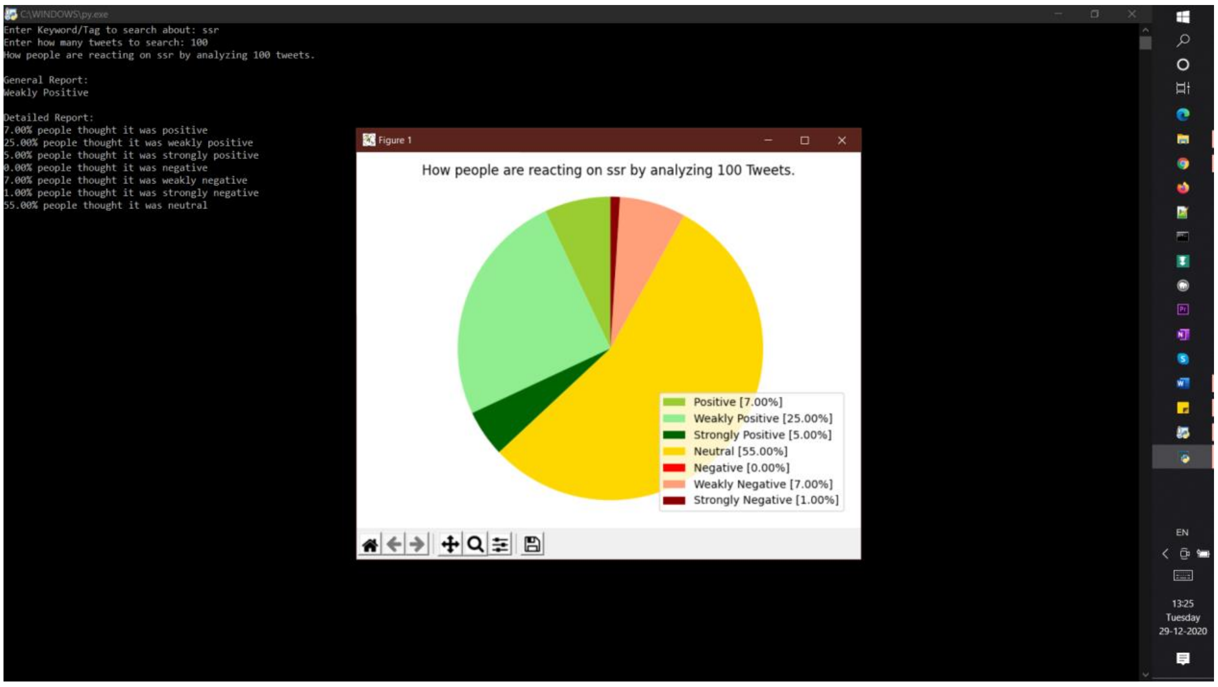


```
C:\WINDOWS\py.exe
Enter Keyword/Tag to search about: #farmbill
Enter how many tweets to search: 100
How people are reacting on #farmbill by analyzing 100 tweets.

General Report:
Weakly Positive

Detailed Report:
25.00% people thought it was positive
9.00% people thought it was weakly positive
0.00% people thought it was strongly positive
2.00% people thought it was negative
3.00% people thought it was weakly negative
0.00% people thought it was strongly negative
61.00% people thought it was neutral
```

4.3 SSR



```
C:\WINDOWS\py.exe
Enter Keyword/Tag to search about: ssr
Enter how many tweets to search: 100
How people are reacting on ssr by analyzing 100 tweets.

General Report:
Weakly Positive

Detailed Report:
7.00% people thought it was positive
25.00% people thought it was weakly positive
5.00% people thought it was strongly positive
0.00% people thought it was negative
7.00% people thought it was weakly negative
1.00% people thought it was strongly negative
55.00% people thought it was neutral
```

5. Testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test.[1] Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use.

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- is sufficiently usable,
- can be installed and run in its intended environments, and
- achieves the general result its stakeholder's desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects). The job of testing is an iterative process as when one bug is fixed, it can illuminate other, deeper bugs, or can even create new ones.

Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors.[1]

Software testing can be conducted as soon as executable software (even if partially complete) exists. The overall approach to software development often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an agile approach, requirements, programming, and testing are often done concurrently.

5.0.1 Overview

Although software testing can determine the correctness of software under the assumption of some specific hypotheses (see the hierarchy of testing difficulty below), testing cannot identify all the defects within the software.[2] Instead, it furnishes a *criticism* or *comparison* that compares the state and Behavior of the product against test oracles—principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, contracts,[3] comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions, but only that it does not function properly under specific conditions.[4] The scope of software testing often includes the examination of code as well as the execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.[5]:41–43

Every software product has a target audience. For example, the audience for video game software is completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it can assess whether the software product will be acceptable to its end users, its target audience, its purchasers and other stakeholders. Software testing assists in making this assessment.

5.0.2 Defects and failures

Not all software defects are caused by coding errors. One common source of expensive defects is requirement gaps, i.e., unrecognized requirements that result in errors of omission by the program designer.[5]:426 Requirement gaps can often be non-functional requirements such as testability, scalability, maintainability, performance, and security.

Software faults occur through the following processes. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure.[6] Not all defects will necessarily result in failures. For example, defects in the dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new computer hardware platform, alterations in source data, or interacting with different software.[6] A single defect may result in a wide range of failure symptoms.

5.0.3 Input combinations and preconditions

A fundamental problem with software testing is that testing under *all* combinations of inputs and preconditions (initial state) is not feasible, even with a simple product.[4]:17–18[7] This means that the number of defects in a software product can be very large and defects that occur infrequently are difficult to find in testing. More significantly, non-functional dimensions of quality (how it is supposed to *be* versus what it is supposed to *do*)—usability, scalability, performance, compatibility, reliability—can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another.

Software developers can't test everything, but they can use combinatorial test design to identify the minimum number of tests needed to get the coverage they want. Combinatorial test design enables users to get greater test coverage with fewer tests. Whether they are looking for speed or test depth, they can use combinatorial test design methods to build structured variation into their test cases.

5.1 Testing Techniques & Testing Strategies

There are many approaches available in software testing. Reviews, walkthroughs, or inspections are referred to as static testing, whereas executing programmed code with a given set of test cases is referred to as dynamic testing.[14][15]

Static testing is often implicit, like proofreading, plus when programming tools/text editors check source code structure or compilers (pre-compilers) check syntax and data flow as static program analysis. Dynamic testing takes place when the program itself is run. Dynamic testing may begin before the program is 100% complete in order to test particular sections of code and are applied to discrete functions or modules.[14][15] Typical techniques for these are either using stubs/drivers or execution from a debugger environment.[15]

Static testing involves verification, whereas dynamic testing also involves validation.[15]

Passive testing means verifying the system Behavior without any interaction with the software product. Contrary to active testing, testers do not provide any test data but look at system logs and traces. They mine for patterns and specific Behavior in order to make some kind of decisions.[16] This is related to offline runtime verification and log analysis.

5.1.2 Exploratory approach

Exploratory testing is an approach to software testing that is concisely described as simultaneous learning, test design and test execution. who coined the term in 1984 defines exploratory testing as "a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the quality of his/her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project?

5.1.3 The "box" approach

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that the tester takes when designing test cases. A hybrid approach called grey-box testing may also be applied to software testing methodology. With the concept of grey-box testing—which develops tests from specific design elements—gaining prominence, this "arbitrary distinction" between black- and white-box testing has faded somewhat.

5.1.4 White box testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) verifies the internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing, an internal perspective of the system (the source code), as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g., in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration, and system levels of the software testing process, it is usually done at the unit level.[21] It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

Techniques used in white-box testing include.

- API testing – testing of the application using public and private APIs (application programming interfaces)
- Code coverage – creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
- Fault injection methods – intentionally introducing faults to gauge the efficacy of testing strategies
- Mutation testing methods
- Static testing methods

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black-box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested. Code coverage as a software metric can be reported as a percentage for:

- *Function coverage*, which reports on functions executed
- *Statement coverage*, which reports on the number of lines executed to complete the test
- *Decision coverage*, which reports on whether both the True and the False branch of a given test has been executed

100% statement coverage ensures that all code paths or branches (in terms of control flow) are executed at least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly.[25] Pseudo-tested functions and methods are those that are covered but not specified (it is possible to remove their body without breaking any test case).

4.1.5 Black box testing

Black-box testing (also known as functional testing) treats the software as a "black box," examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it.[27] Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing, and specification-based testing.[19][20][24]

Specification-based testing aims to test the functionality of software according to the applicable requirements.[28] This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or Behavior), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional.

Specification-based testing may be necessary to assure correct functionality, but it is insufficient to guard against complex or high-risk situations.[29]

One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality. On the other hand, black-box testing has been said to be "like a walk in a dark labyrinth without a flashlight."[30] Because they do not examine the source code, there are situations when a tester writes many test cases to check something that could have been tested by only one test case or leaves some parts of the program untested.

This method of test can be applied to all levels of software testing: unit, integration, system and acceptance.[21] It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

4.1.6 Compound box testing

Component interface testing is a variation of black-box testing, with the focus on the data values beyond just the related actions of a subsystem component.[31] The practice of component interface testing can be used to check the handling of data passed between various units, or subsystem components, beyond full integration testing between those units.[32][33] The data being passed can be considered as "message packets" and the range or data types can be checked, for data generated from one unit, and tested for validity before being passed into another unit. One option for interface testing is to keep a separate log file of data items being passed, often with a timestamp logged to allow analysis of thousands of cases of data passed between units for days or weeks. Tests can include checking the handling of some extreme data values while other interface variables are passed as normal values.[32] Unusual data values in an interface can help explain unexpected performance in the next unit.

Visual testing

The aim of visual testing is to provide developers with the ability to examine what was happening at the point of software failure by presenting the data in such a way that the developer can easily find the information she or he requires, and the information is expressed clearly.[34][35]

At the core of visual testing is the idea that showing someone a problem (or a test failure), rather than just describing it, greatly increases clarity and understanding. Visual testing, therefore, requires the recording of the entire test process – capturing everything that occurs on the test system in video format. Output videos are supplemented by real-time tester input via picture-in-a-picture webcam and audio commentary from microphones.

Visual testing provides a number of advantages. The quality of communication is increased drastically because testers can show the problem (and the events leading up to it) to the developer as opposed to just describing it and the need to replicate test failures will cease to exist in many cases. The developer will have all the evidence she or he requires of a test failure and can instead focus on the cause of the fault and how it should be fixed.

Ad hoc testing and exploratory testing are important methodologies for checking software integrity, because they require less preparation time to implement, while the important bugs can be found quickly. In ad hoc testing, where testing takes place in an improvised, impromptu way, the ability of the tester(s) to base testing off documented methods and then improvise variations of those tests can result in more rigorous examination of defect fixes. However, unless strict documentation of the procedures is maintained, one of the limits of ad hoc testing is lack of repeatability.[36]

Grey-box testing

Grey-box testing (American spelling: Gray-box testing) involves having knowledge of internal data structures and algorithms for purposes of designing tests while executing those tests at the user, or black-box level. The tester will often have access to both "the source code and the executable binary."[37] Grey-box testing may also include reverse engineering (using dynamic code analysis) to determine, for instance, boundary values or error messages.[37] Manipulating input data and formatting output do not qualify as grey-box, as the input and output are clearly outside of the "black box" that we are calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for the test.

By knowing the underlying concepts of how the software works, the tester makes better-informed testing choices while testing the software from outside. Typically, a grey-box tester will be permitted to set up an isolated testing environment with activities such as seeding a database. The tester can observe the state of the product being tested after performing certain actions such as executing SQL statements against the database and then executing queries to ensure that the expected changes have been reflected. Grey-box testing implements intelligent test scenarios, based on limited information. This will particularly apply to data type handling, exception handling, and so on.

Testing levels

Broadly speaking, there are at least three levels of testing: unit testing, integration testing, and system testing.[39][40][41][42] However, a fourth level, acceptance testing, may be included by developers. This may be in the form of operational acceptance testing or be simple end-user (beta) testing, testing to ensure the software meets functional expectations.[43][44][45] Based on the ISTQB Certified Test Foundation Level syllabus, test levels includes those four levels, and the fourth level is named acceptance testing.[46] Tests are frequently grouped into one of these levels by where they are added in the software development process, or by the level of specificity of the test.

Unit testing

Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.[47]

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to ensure that the building blocks of the software work independently from each other.

Unit testing is a software development process that involves a synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development life cycle. Unit testing aims to eliminate construction errors before code is promoted to additional testing; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development process.

Depending on the organization's expectations for software development, unit testing might include static code analysis, data-flow analysis, metrics analysis, peer code reviews, code coverage analysis and other software testing practices.

Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is co considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.[48]

Integration tests usually involve a lot of code, and produce traces that are larger than those produced by unit tests. This has an impact on the ease of localizing the fault when an integration test fails. To overcome this issue, it has been proposed to automatically cut the large tests in smaller pieces to improve fault localization.[49]

System testing

System testing tests a completely integrated system to verify that the system meets its requirements.[50][obsolete source] For example, a system test might involve testing a login interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff.

Acceptance testing

Commonly this level of Acceptance testing includes the following four types[46]:

- User acceptance testing
- Operational acceptance testing
- Contractual and regulatory acceptance testing
- Alpha and beta testing

User acceptance testing and Alpha and beta testing are described in the next #Testing types section.

Operational acceptance is used to conduct operational readiness (pre-release) of a product, service or system as part of a quality management system. OAT is a common type of non-functional software testing, used mainly in software development and software maintenance projects. This type of testing focuses on the operational readiness of the system to be supported, or to become part of the production environment. Hence, it is also known as operational readiness testing (ORT) or Operations readiness and assurance (OR&A) testing. Functional testing within OAT is limited to those tests that are required to verify the *non-functional* aspects of the system.

In addition, the software testing should ensure that the portability of the system, as well as working as expected, does not also damage or partially corrupt its operating environment or cause other processes within that environment to become inoperative.[51]

Contractual acceptance testing is performed based on the contract's acceptance criteria defined during the agreement of the contract, while regulatory acceptance testing is performed based on the relevant regulations to the software product. Both of these two testings' can be performed by users or independent testers. Regulation acceptance testing sometimes involves the regulatory agencies auditing the test results.

5.2 Debugging & Code Improvement

Installation testing

Most software systems have installation procedures that are needed before they can be used for their main purpose. Testing these procedures to achieve an installed software system that may be used is known as installation testing.

Compatibility testing

A common cause of software failure (real or perceived) is a lack of its compatibility with other application software, operating systems (or operating system versions, old or new), or target environments that differ greatly from the original (such as a terminal or GUI application intended to be run on the desktop now being required to become a Web application, which must render in a Web browser). For example, in the case of a lack of backward compatibility, this can occur because the programmers develop and test software only on the latest version of the target environment, which not all users may be running. This results in the unintended consequence that the latest work may not function on earlier versions of the target environment, or on older hardware that earlier versions of the target environment were capable of using. Sometimes such issues can be fixed by proactively abstracting operating system functionality into a separate program module or library.

Smoke and sanity testing

Sanity testing determines whether it is reasonable to proceed with further testing.

Smoke testing consists of minimal attempts to operate the software, designed to determine whether there are any basic problems that will prevent it from working at all. Such tests can be used as build verification test.

Regression testing

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, as degraded or lost features, including old bugs that have come back. Such regressions occur whenever software functionality that was previously working correctly, stops working as intended. Typically, regressions occur as an unintended consequence of program changes, when the newly developed part of the software collides with the previously existing code. Regression testing is typically the largest test effort in commercial software development,[53] due to checking numerous details in prior software features, and even new software can be developed while using some old test cases to test parts of the new design to ensure prior functionality is still supported.

Common methods of regression testing include re-running previous sets of test cases and checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release process and the risk of the added features. They can either be complete, for changes added late in the release or deemed to be risky, or be very shallow, consisting of positive tests on each feature, if the changes are early in the release or deemed to be of low risk. In regression testing, it is important to have strong assertions on the existing

Behavior. For this, it is possible to generate and add new assertions in existing test cases,[54] this is known as automatic test amplification.

Acceptance testing

Acceptance testing can mean one of two things:

1. A smoke test is used as a build acceptance test prior to further testing, e.g., before integration or regression.
2. Acceptance testing performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.

Alpha testing

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing before the software goes to beta testing.

Beta testing

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team known as beta testers. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Beta versions can be made available to the open public to increase the feedback field to a maximal number of future users and to deliver value earlier, for an extended or even indefinite period of time (perpetual beta).

Functional vs non-functional testing

Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work."

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or other performance, Behavior under certain constraints, or security. Testing will determine the breaking point, the point at which extremes of scalability or performance leads to unstable execution. Non-functional requirements tend to be those that reflect the quality of the product, particularly in the context of the suitability perspective of its users.

Continuous testing

Continuous testing is the process of executing automated tests as part of the software delivery pipeline to obtain immediate feedback on the business risks associated with a software release candidate.[58][59] Continuous testing includes the validation of both functional requirements and non-functional requirements; the scope of testing extends from validating bottom-up requirements or user stories to assessing the system requirements associated with overarching business goals.

Destructive testing

Destructive testing attempts to cause the software or a sub-system to fail. It verifies that the software functions properly even when it receives invalid or unexpected inputs, thereby establishing the robustness of input validation and error-management routines.[citation needed] Software fault injection, in the form of fuzzing, is an example of failure testing. Various commercial non-functional testing tools are linked from the software fault injection page; there are also numerous open-source and free software tools available that perform destructive testing.

Software performance testing

Performance testing is generally executed to determine how a system or sub-system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Load testing is primarily concerned with testing that the system can continue to operate under a specific load, whether that be large quantities of data or a large number of users. This is generally referred to as software scalability. The related load testing activity of when performed as a non-functional activity is often referred to as *endurance testing*. Volume testing is a way to test software functions even when certain components (for example a file or database) increase radically in size. Stress testing is a way to test reliability under unexpected or rare workloads. *Stability testing* (often referred to as load or endurance testing) checks to see if the software can continuously function well in or above an acceptable period.

There is little agreement on what the specific goals of performance testing are. The terms load testing, performance testing, scalability testing, and volume testing, are often used interchangeably.

Real-time software systems have strict timing constraints. To test if timing constraints are met, real-time testing is used.

Usability testing

Usability testing is to check if the user interface is easy to use and understand. It is concerned mainly with the use of the application. This is not a kind of testing that can be automated; actual human users are needed, being monitored by skilled UI designers.

Accessibility testing

Accessibility testing may include compliance with standards such as:

- Americans with Disabilities Act of 1990
- Section 508 Amendment to the Rehabilitation Act of 1973
- Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C)
-

Security testing

Security testing is essential for software that processes confidential data to prevent system intrusion by hackers.

The International Organization for Standardization (ISO) defines this as a "type of testing conducted to evaluate the degree to which a test item, and associated data and information, are protected so that unauthorised persons or systems cannot use, read or modify them, and authorized persons or systems are not denied access to them.

Internationalization and localization

Testing for internationalization and localization validates that the software can be used with different languages and geographic regions. The process of pseudo localization is used to test the ability of an application to be translated to another language, and make it easier to identify when the localization process may introduce new bugs into the product.

Globalization testing verifies that the software is adapted for a new culture (such as different currencies or time zones).[64]

Actual translation to human languages must be tested, too. Possible localization and globalization failures include:

- Software is often localized by translating a list of strings out of context, and the translator may choose the wrong translation for an ambiguous source string.
- Technical terminology may become inconsistent, if the project is translated by several people without proper coordination or if the translator is imprudent.
- Literal word-for-word translations may sound inappropriate, artificial or too technical in the target language.
- Untranslated messages in the original language may be left hard coded in the source code.
- Some messages may be created automatically at run time and the resulting string may be ungrammatical, functionally incorrect, misleading or confusing.
- Software may use a keyboard shortcut that has no function on the source language's keyboard layout, but is used for typing characters in the layout of the target language.
- Software may lack support for the character encoding of the target language.
- Fonts and font sizes that are appropriate in the source language may be inappropriate in the target language; for example, CJK characters may become unreadable, if the font is too small.
- A string in the target language may be longer than the software can handle. This may make the string partly invisible to the user or cause the software to crash or malfunction.
- Software may lack proper support for reading or writing bi-directional text.
- Software may display images with text that was not localized.
- Localized operating systems may have differently named system configuration files and environment variables and different formats for date and currency.

Development testing

Main article: Development testing

Development Testing is a software development process that involves the synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle.

Development Testing aims to eliminate construction errors before code is promoted to other testing; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development process.

Depending on the organization's expectations for software development, Development Testing might include static code analysis, data flow analysis, metrics analysis, peer code reviews, unit testing, code coverage analysis, traceability, and other software testing practices.

A/B testing

Main article: A/B testing

A/B testing is a method of running a controlled experiment to determine if a proposed change is more effective than the current approach. Customers are routed to either a current version (control) of a feature, or to a modified version (treatment) and data is collected to determine which version is better at achieving the desired outcome.

Concurrent testing

Main article: Concurrent testing

Concurrent or concurrency testing assesses the behaviour and performance of software and systems that use concurrent computing, generally under normal usage conditions. Typical problems this type of testing will expose are deadlocks, race conditions and problems with shared memory/resource handling.

Conformance testing or type testing

Main article: Conformance testing

In software testing, conformance testing verifies that a product performs according to its specified standards. Compilers, for instance, are extensively tested to determine whether they meet the recognized standard for that language.

Output comparison testing

Creating a display expected output, whether as data comparison of text or screenshots of the UI,[65]:195 is sometimes called snapshot testing or Golden Master Testing unlike many other forms of testing, this cannot detect failures automatically and instead requires that a human evaluate the output for inconsistencies.

Testing process

Traditional waterfall development model

A common practice in waterfall development is that testing is performed by an independent group of testers. This can happen:

- after the functionality is developed, but before it is shipped to the customer.[66] This practice often results in the testing phase being used as a project buffer to compensate for project delays, thereby compromising the time devoted to testing.[13]:145–146
- at the same moment the development project starts, as a continuous process until the project finishes.[67]

However, even in the waterfall development model, unit testing is often done by the software development team even when further testing is done by a separate team.[68]

Further information: Capability Maturity Model Integration *and* Waterfall model

Agile or XP development model

In contrast, some emerging software disciplines such as extreme programming and the agile software development movement, adhere to a "test-driven software development" model. In this process, unit tests are written first, by the software engineers (often with pair programming in the extreme programming methodology). The tests are expected to fail initially. Each failing test is followed by writing just enough code to make it pass.[69] This means the test suites are continuously updated as new failure conditions and corner cases are discovered, and they are integrated with any regression tests that are developed. Unit tests are maintained along with the rest of the software source code and generally integrated into the build process (with inherently interactive tests being relegated to a partially manual build acceptance process).

The ultimate goals of this test process are to support continuous integration and to reduce defect rates.[70][69]

This methodology increases the testing effort done by development, before reaching any formal testing team. In some other development models, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

A sample testing cycle

Although variations exist between organizations, there is a typical cycle for testing.[2] The sample below is common among organizations employing the Waterfall development model. The same practices are commonly found in other development models, but might not be as clear or explicit.

- Requirements analysis: Testing should begin in the requirements phase of the software development life cycle. During the design phase, testers work to determine what aspects of a design are testable and with what parameters those tests work.
- Test planning: Test strategy, test plan, testbed creation. Since many activities will be carried out during testing, a plan is needed.
- Test development: Test procedures, test scenarios, test cases, test datasets, test scripts to use in testing software.
- Test execution: Testers execute the software based on the plans and test documents then report any errors found to the development team. This part could be complex when running tests with a lack of programming knowledge.
- Test reporting: Once testing is completed; testers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.
- Test result analysis: Or Defect Analysis, is done by the development team usually along with the client, in order to decide what defects should be assigned, fixed, rejected (i.e., found software working properly) or deferred to be dealt with later.
- Defect Retesting: Once a defect has been dealt with by the development team, it is retested by the testing team.
- Regression testing: It is common to have a small test program built of a subset of tests, for each integration of new, modified, or fixed software, in order to ensure that the latest delivery has not ruined anything and that the software product as a whole is still working correctly.

- Test Closure: Once the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects.

Automated testing

Many programming groups are relying more and more on automated testing, especially groups that use test-driven development. There are many frameworks to write tests in, and continuous integration software will run tests automatically every time code is checked into a version control system.

While automation cannot reproduce everything that a human can do (and all the ways they think of doing it), it can be very useful for regression testing. However, it does require a well-developed test suite of testing scripts in order to be truly useful.

Testing tools

Program testing and fault detection can be aided significantly by testing tools and debuggers. Testing/debug tools include features such as:

- Program monitors, permitting full or partial monitoring of program code, including:
- Instruction set simulator, permitting complete instruction level monitoring and trace facilities
- Hypervisor, permitting complete control of the execution of program code including: -
- Program animation, permitting step-by-step execution and conditional breakpoint at source level or in machine code
- Code coverage reports
- Formatted dump or symbolic debugging, tools allowing inspection of program variables on error or at chosen points
- Automated functional Graphical User Interface (GUI) testing tools are used to repeat system-level tests through the GUI
- Benchmarks, allowing run-time performance comparisons to be made
- Performance analysis (or profiling tools) that can help to highlight hot spots and resource usage

Some of these features may be incorporated into a single composite tool or an Integrated Development Environment (IDE).

Measurement in software testing

Quality measures include such topics as correctness, completeness, security and ISO/IEC 9126 requirements such as capability, reliability, efficiency, portability, maintainability, compatibility, and usability.

There are a number of frequently used software metrics, or measures, which are used to assist in determining the state of the software or the adequacy of the testing.

Hierarchy of testing difficulty

Based on the amount of test cases required to construct a complete test suite in each context (i.e. a test suite such that, if it is applied to the implementation under test, then we collect enough information to precisely determine whether the system is correct or incorrect according to some specification), a hierarchy of testing difficulty has been proposed.[71] [72] It includes the following testability classes:

- Class I: there exists a finite complete test suite.
- Class II: any partial distinguishing rate (i.e., any incomplete capability to distinguish correct systems from incorrect systems) can be reached with a finite test suite.
- Class III: there exists a countable complete test suite.
- Class IV: there exists a complete test suite.
- Class V: all cases.

It has been proved that each class is strictly included in the next. For instance, testing when we assume that the Behavior of the implementation under test can be denoted by a deterministic finite-state machine for some known finite sets of inputs and outputs and with some known number of states belongs to Class I (and all subsequent classes). However, if the number of states is not known, then it only belongs to all classes from Class II on. If the implementation under test must be a deterministic finite-state machine failing the specification for a single trace (and its continuations), and its number of states is unknown, then it only belongs to classes from Class III on. Testing temporal machines where transitions are triggered if inputs are produced within some real-bounded interval only belongs to classes from Class IV on, whereas testing many non-deterministic systems only belongs to Class V (but not all, and some even belong to Class I). The inclusion into Class I does not require the simplicity of the assumed computation model, as some testing cases involving implementations written in any programming language, and testing implementations defined as machines depending on continuous magnitudes, have been proved to be in Class I. Other elaborated cases, such as the testing framework by Matthew Hennessy under must semantics, and temporal machines with rational timeouts, belong to Class II.

Testing artifacts

A software testing process can produce several artifacts. The actual artifacts produced are a factor of the software development model used, stakeholder and organisational needs.

Test plan

A test plan is a document detailing the approach that will be taken for intended test activities. The plan may include aspects such as objectives, scope, processes and procedures, personnel requirements, and contingency plans.[43] The test plan could come in the form of a single plan that includes all test types (like an acceptance or system test plan) and planning considerations, or it may be issued as a master test plan that provides an overview of more than one detailed test plan (a plan of a plan).[43] A test plan can be, in some cases, part of a wide "test strategy" which documents overall testing approaches, which may itself be a master test plan or even a separate artifact.

Traceability matrix

A traceability matrix is a table that correlates requirements or design documents to test documents. It is used to change tests when related source documents are changed, to select test cases for execution when planning for regression tests by considering requirement coverage.

Test case

A test case normally consists of a unique identifier, requirement references from a design specification, preconditions, events, a series of steps (also known as actions) to follow, input, output, expected result, and the actual result. Clinically defined, a test case is an input and an expected result.[73] This can be as terse as 'for condition x your derived result is y', although normally test cases describe in more detail the input scenario and what results might be expected. It can occasionally be a series of steps (but often steps are contained in a separate test procedure that can be exercised against multiple test cases, as a matter of economy) but with one expected result or expected outcome. The optional fields are a test case ID, test step, or order of execution number, related requirement(s), depth, test category, author, and check boxes for whether the test is automatable and has been automated. Larger test cases may also contain prerequisite states or steps, and descriptions. A test case should also contain a place for the actual result. These steps can be stored in a word processor document, spreadsheet, database, or other common repositories. In a database system, you may also be able to see past test results, who generated the results, and what system configuration was used to generate those results. These past results would usually be stored in a separate table.

Test script

A test script is a procedure or programming code that replicates user actions. Initially, the term was derived from the product of work created by automated regression test tools. A test case will be a baseline to create test scripts using a tool or a program.

Test suite

The most common term for a collection of test cases is a test suite. The test suite often also contains more detailed instructions or goals for each collection of test cases. It definitely contains a section where the tester identifies the system configuration used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests.

Test fixture or test data

In most cases, multiple sets of values or data are used to test the same functionality of a particular feature. All the test values and changeable environmental components are collected in separate files and stored as test data. It is also useful to provide this data to the client and with the product or a project. There are techniques to generate test data.

Test harness

The software, tools, samples of data input and output, and configurations are all referred to collectively as a test harness.

6 Implementation

Implementation is the process that actually yields the lowest-level system elements in the system hierarchy (system breakdown structure). System elements are made, bought, or reused. Production involves the hardware fabrication processes of forming, removing, joining, and finishing, the software realization processes of coding and testing, or the operational procedures development processes for operators' roles. If implementation involves a production process, a manufacturing system which uses the established technical and management processes may be required.

The purpose of the implementation process is to design and create (or fabricate) a system element conforming to that element's design properties and/or requirements. The element is constructed employing appropriate technologies and industry practices. This process bridges the system definition processes and the integration process.

6.1 System Implementation

During the implementation process, engineers apply the design properties and/or requirements allocated to a system element to design and produce a detailed description. They then fabricate, code, or build each individual element using specified materials, processes, physical or logical arrangements, standards, technologies, and/or information flows outlined in detailed descriptions (drawings or other design documentation). A system element will be verified against the detailed description of properties and validated against its requirements.

Such figures provide a useful overview of the **systems engineering** (SE) community's perspectives on what is required for implementation and what the general results of implementation may be. These are further supported by the discussion of implementation inputs, outputs, and activities found in the National Aeronautics and Space Association's (NASA's) *Systems Engineering Handbook* (NASA 2007). It is important to understand that these views are process -oriented. While this is a useful model, examining implementation only in terms of process can be limiting.

Depending on the technologies and systems chosen when a decision is made to produce a system element, the implementation process outcomes may generate constraints to be applied on the architecture of the higher-level system; those constraints are normally identified as derived system requirements and added to the set of system requirements applicable to this higher-level system. The architectural design has to must take those constraints into account.

If the decision is made to purchase or reuse an existing system element, it has to must be identified as a constraint or system requirement applicable to the architecture of the higher-level system. Conversely, the implementation process may involve some adaptation or adjustments to the system requirement in order to be integrated into a higher-level system or aggregate.

Implementation also involves packaging, handling, and storage, depending on the concerned technologies and where or when the system requirement needs to be integrated into a higher-level aggregate. Developing the supporting documentation for a system requirement, such as the manuals for operation, maintenance, and/or installation, is also a part of the implementation process; these artifacts are utilized in the system deployment and use phase. The system element requirements and the associated verification and validation criteria are inputs to this process; these inputs come from the **architectural design** process detailed outputs.

Execution of the implementation process is governed by both industrial and government standards and the terms of all applicable agreements. This may include conditions for packaging and storage, as well as preparation for use activities, such as operator training. In addition, packaging, handling, storage, and transportation (PHS&T) considerations will constrain the implementation activities. For more information, refer to the discussion of PHS&T in the System Deployment and Use article. The developing or integrating organization will likely have enterprise-level safety practices and guidelines that must also be considered.

Activities of the Process

The following major activities and tasks are performed during this process:

- **Define the implementation strategy** - Implementation process activities begin with detailed design and include developing an implementation strategy that defines fabrication and coding procedures, tools and equipment to be used, implementation tolerances, and the means and criteria for auditing configuration of resulting elements to the detailed design documentation. In the case of repeated system element implementations (such as for mass manufacturing or replacement elements), the implementation strategy is defined and refined to achieve consistent and repeatable element production; it is retained in the project decision database for future use. The implementation strategy contains the arrangements for packing, storing, and supplying the implemented element.
- **Realize the system element** - Realize or adapt and produce the concerned system element using the implementation strategy items as defined above. Realization or adaptation is conducted with regard to standards that govern applicable safety, security, privacy, and environmental guidelines or legislation and the practices of the relevant implementation technology. This requires the fabrication of hardware elements, development of software elements, definition of training capabilities, drafting of training documentation, and the training of initial operators and maintainers.
- **Provide evidence of compliance** - Record evidence that the system element meets its requirements and the associated verification and validation criteria as well as the legislation policy. This requires the conduction of peer reviews and unit testing, as well as inspection of operation and maintenance manuals. Acquire measured properties that characterize the implemented element (weight, capacities, effectiveness, level of performance, reliability, availability, etc.).
- **Package, store, and supply the implemented element** - This should be defined in the implementation strategy.

Artifacts and Ontology Elements

This process may create several artifacts such as:

- an implemented system
- implementation tools
- implementation procedures
- an implementation plan or strategy
- verification reports

- issue, anomaly, or trouble reports
- change requests (about design)

6.2 Software Implementation

Structured Programming

The codes lead to enlarge the software size as the codes multiply thus making it a difficult task to connect with the program flow. It becomes very hard for the program to be shared or modifies, as the files, programs, procedures and the manner in which the program is constructed is not remembered. This drawback of coding is overcome by structured programming. Some of the structures such as subroutines and loops are used by the developers. These subroutines and loops facilitate in improving the efficiency and the coding time is decreased and also the coding is organized.

The coding of a particular programming is described by structured programming. There are three basic concepts upon which the concept of structured programming revolves. They are -

- **Top-down analysis** – The most important part of any program is problem solving. Problem can be easily solved only when the problem is understood. The problem is divided into several parts. Each sub-problem is solved individually and hence the problem solving is simplified.
- **Modular Programming** – Programming can also be done by simplifying the code into groups of instructions. These groups of instructions are known as modules or sub-programs. Modular programming is done by following the Top-down analysis. As the jump method makes it difficult to locate the program. Structured programming always prefers modular format that Jumps.
- **Structured Coding** – Under this concept, the modules are broken down further thus simplifying the execution. The flow of the program is controlled as control structure is chosen by structured programming. The coding instructions are enabled to organize by structured coding.

Functional Programming

It is a programming that uses mathematical functions. When a particular argument is received by a mathematical function, result produced by that function will be the same. In some of the procedural languages, procedures take over the control on the flow of the program. There is a possibility of the state of the program getting changed in the process of shifting the control flow from one procedure to another.

When a particular argument is received by the procedural programming, result produced by that program will be different as the state of the program keeps on changing. This calls for taking more consideration about some of the aspects of programming such as sequence of the program and the timing of the code.

Mathematical functions are used by functional programming and which enable to produce the result without considering the state of the program. This enables to forecast the program behaviour.

- There are different concepts upon which the concept of functional programming revolves. They are
- **First class and High-order functions** – Some other function can be accepted as an argument or the result may be produced on some other function by these High-order functions.
- **Pure functions** – These functions are not considered to be destructive as the memory, Input or Output is not influenced by these functions. They can be easily deleted, if not required and hence the program is not hampered.
- **Recursion** – The program code is repeated by the function as the function calls itself. The repetition of the program code stops when some pre-defined conditions match. The repetition of the code results in development of loops. Here the input and the output are based on functional programming.
- **Strict evaluation** – The argument of the function is evaluated either by strict evaluation or non-strict evaluation. Before a function is called, the expression is evaluated by strict evaluation. Unless required the expression is not evaluated by non-strict evaluation.
- **λ-calculus** – λ-calculus is chosen by the function programming. Only when they occur, they are evaluated.
- Examples of the function programming are - Common Lisp, Scala, Haskell, Erlang and F#

Programming style

The code is written in accordance with some of the predefined coding rules. Programming style refers to the set of such coding rules. For the program, the program code may be written by on developer and the program is worked on by other developer. Thus, making it a big confusion. This confusion is avoided by setting and following some standard programming styles to write the program code.

Relevant function and variable names are included by the programming style for a particular task. This facilitates in letting the indentation to be well-placed, at the reader convenience the code can be commented and the presentation of the code. Thus, the program code is easily understood. Here solving of the errors can be made easy. It also simplifies the documentation of the program code and updating.

Coding Guidelines

Different organization has different language of coding and different styles for coding.

Each of the organization coding guidelines has to consider some of the coding elements in general, such as -

- **Naming conventions** – Defines the way in which the functions, variables, constants and variables need to be named.
- **Indenting** – Indenting is the space that is left blank at the beginning of line, 2-8 whitespace or single tab is used for indenting.
- **Whitespace** - It is generally omitted at the end of line.

- **Operators** – The different operators – assignment. Mathematical and logical are defined. For example, space should be there before and after the assignment operator '=' as in "x = 2".
- **Control Structures** – Some of the clauses such as if-then-else, case-switch, while-until are to be written by following some set of rules which are defined by Control Structures.
- **Line length and wrapping** – The length of the line in terms of number of characters is defined. For the long sentences the lines should be wrapped as per the specified rules of wrapping.
- **Functions** – The declaration and the calling of the function is defined in two ways – with parameters and without parameters.
- **Variables** – The definitions and the declarations of different variables are mentioned.
- **Comments** – The functioning of the code is described as comments. The predefined comments help in creating documentations and the associated descriptions.

How the documentation of the Software is done?

A software document is considered as a repository of information related to the complete process of the software. The information about the usage of the product is also provided by the software documentation. Well-structured and well-organized software documentation involves maintenance of certain documents. They are –

Requirement documentation

As the name implies, all the functional and non-functional requirements and the description of the desired software are documented in requirement documentation. Since this includes the collection of requirements, it is considered as an important tool for the complete team of developers, designers and testers.

This document is prepared on the basis of previous data of the running software, which is running at the Client; It is also based on the research, questionnaires submitted by the clients and by interviewing the clients. The software management team stores and maintains this document and is stored in either spreadsheet or word processing document.

It is the basic foundation to develop any software. Verification and validation of the software is also done on the basis of this document. Requirement documentation also facilitates in preparing the test-cases.

Software Design documentation

The necessary information that is required for developing the software is provided by this software design documentation. The document provides information related to High-level software architecture, Software design, Data flow diagrams, and Database design

The software implementation is mainly based on this document. All the relevant and important information regarding coding and implementation is provided. Though not provide the details of exactly how the program is coded.

Technical documentation

The information about the code is documented by technical documentation, which can only be prepared by the developers. The code is written along with some additional information such as the code objective, writer of the code, the usage of the code, resources required by the code etc.

When the same code is worked on by different programmers, this documentation enables the programmers to better understand and interact with each other. The ability of re-usability of the code is improved by this documentation.

Technical documentation can be prepared by using different available tools and some of such tools are made available with the programming language. For example, java comes with Javadoc tool, which facilitate in developing the technical documentation of the code.

User documentation

The explanation about the working of the software and best usage of the software for obtaining the desired results is provided by User documentation. User documentation varies from other three types of documentation in the aspect that they are more information oriented for software development, while user documents provides explanation.

The guidelines related to how to install a software, user-guidelines, methods for uninstallation, information about the license updating is provided by these user documentations.

What are the different challenges faced by Software Implementation?

- The implementation of the software is not left without the challenges for the developers. Some of the major challenges are -
- **Code-reuse** – The code that is earlier created and meant for some other software is preferred to be re-used by the management in order to reduce the cost of the end product. This leave the programmers with bunch of issues as they will be in a dilemma on how much of the code need to be re-used and it is troublesome to do the compatibility checks for the code for re-using.
- **Version Management** – For all the new software, the version of the software and the configuration of the software has to be documented and maintained by the developers. This should me made easily available and handy whenever required.
- **Target-Host** – The software program needs to be designed with respect to the host machines at the end of the user. Sometimes it may be feasible to do so but at some instances, it becomes very difficult and challenging to design the software with respect to the host machines.

06.CONCLUSION & FUTURE SCOPE

Approaches that consider only the historical Behavior of the analysed object have been widely employed for trend prediction. However, the contents generated by people are clearly influenced by their connections. How information spreads are an important factor that can be considered in prediction. The sheer amount and the different types of data on twitter and the public nature of tweets have allowed exploiting twitter information in data analysis. The proposed approach achieved better results than the standard time series-based models. In addition to simple prediction techniques, such as linear regression, we applied more robust techniques that resulted in even more accurate models. In conclusion, we found out that looking at the social structure of data sources alongside the main analysed data can help better understanding the information temporal Behavior.

07. REFRENCES

1. Opinion Mining and Trend Analysis on Twitter Data. (2020). *International Journal of Recent Technology and Engineering Regular Issue, 9*(1), 1650-1653. doi:10.35940/ijrte.a1547.059120
2. Aghababaei, S., & Makrehchi, M. (2018). Mining Twitter data for crime trend prediction. *Intelligent Data Analysis, 22*(1), 117-141. doi:10.3233/ida-163183
3. Lambrecht, A., Tucker, C., & Wiertz, C. (2014). Should You Target Early Trend Propagators? Evidence from Twitter. *SSRN Electronic Journal*. doi:10.2139/ssrn.2419743
4. Sharma, P., Agarwal, A., & Sardana, N. (2018). Extraction of Influencers Across Twitter Using Credibility and Trend Analysis. *2018 Eleventh International Conference on Contemporary Computing (IC3)*. doi:10.1109/ic3.2018.8530462
5. Bizhanova, A., & Uchida, O. (2014). Product Reputation Trend Extraction from Twitter. *Social Networking, 03*(04), 196-202. doi:10.4236/sn.2014.34024
6. Rathod, T., & Barot, M. (2018). Trend Analysis on Twitter for Predicting Public Opinion on Ongoing Events. *International Journal of Computer Applications, 180*(26), 13-17. doi:10.5120/ijca2018916596
7. Pawar, M. P. (2019). Real Time Twitter Trend Mining System. *International Journal for Research in Applied Science and Engineering Technology, 7*(11), 70-71. doi:10.22214/ijraset.2019.11012
8. Lu, R., & Yang, Q. (2012). Trend Analysis of News Topics on Twitter. *International Journal of Machine Learning and Computing,* 327-332. doi:10.7763/ijmlc.2012.v2.139

9. Song, M., & Kim, M. C. (2013). RT^2M: Real-Time Twitter Trend Mining System. *2013 International Conference on Social Intelligence and Technology*. doi:10.1109/society.2013.19
10. Doshi, Z., Nadkarni, S., Ajmera, K., & Shah, N. (2017). TweerAnalyzer: Twitter Trend Detection and Visualization. *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*. doi:10.1109/iccubea.2017.8463951
11. Doshi, Z., Nadkarni, S., Ajmera, K., & Shah, N. (2017). TweerAnalyzer: Twitter Trend Detection and Visualization. *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*. doi:10.1109/iccubea.2017.8463951
12. Mathioudakis, M., & Koudas, N. (2010). TwitterMonitor. *Proceedings of the 2010 International Conference on Management of Data - SIGMOD '10*. doi:10.1145/1807167.1807306
13. Gukanesh, A. V., Kumar, G. K., & K. Karthik Raja Kumar | N. Saranya. (2018). Twitter Data Analytics – Sentiment Analysis of An Election. *International Journal of Trend in Scientific Research and Development, Volume-2*(Issue-3), 1600-1603. doi:10.31142/ijtsrd11457