# TABLE OF CONTENTS

**Abstract I**

**List of Figures II**

**List of Screens III**

|  |  |  |
| --- | --- | --- |
| **Chapter**  **No** | **CONTENT** | **PAGE NO.** |
| **1** | **INTRODUCTION**   * 1. **Motivation**   2. **Problem definition**   3. **Objective of Project**   4. **Limitations of Project**   5. **Organization of Documentation** | **1-3** |
| **2** | **LITERATURE SURVEY**   * 1. **Introduction**   2. **Existing System**   3. **Disadvantages of Existing system**   4. **Proposed System**   5. **Conclusion** | **4-6** |
| **3** | **SYSTEM ANALYSIS**   * 1. **Current environment**   **3.2 System architecture**  **3.2.1 Project architecture**  **3.3 Feasibility study**  **3.3.1 Economical feasibility**  **3.3.2 Technical feasibility**  **3.3.3 Social feasibility** | **7-10** |
| **4** | **System Requirements and Specifications**   * 1. **Requirement analysis**   2. **Functional requirements**   3. **Non-Functional requirements**   4. **Input and output design**   5. **System requirements** | **11-15** |
| **5** | **SYSTEM DESIGN**   * 1. **Introduction**   2. **Dataflow diagram**   3. **UML class diagram**   4. **UML diagram overview (usecase,sequence,collaboration,**   **activity)** | **16-36** |
| **6** | **IMPLEMENTATION**   * 1. **Modules**   2. **Technology description-python**   3. **Coding**   **6.4 Output screens** | **37-70** |
| **7** | **SYSTEM TESTING**  **7.1 Types of System testing** | **71-73** |
| **8**  **9** | **Conclusion**  **8.1 Future scope**  **8.2 Future enhancement**  **References** | **74-76**  **77** |

**LIST OF FIGURES**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Figure No** | **Description** | **Page No** |
| 1 | 3.1 | The Design of Project | 8 |
| 2 | 5.1-5.22 | UML Notations | 17-24 |
| 3 | 5.23-5.49 | UML Components & Relations | 25-36 |
| 4 | 6.1 | IBM Watson | 51 |
| 5 | 6.2 | Reinforcement Learning | 54 |
| 6 | 6.3 | Markov decision process | 55 |
| 7 | 6.4 | Layered Architecture | 56 |
| 8 | 6.5 | Django Architecture | 57 |

**LIST OF SCREENS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **SCREEN SHOT NO** | **Description** | **Page No** |
| 1 | 6.6 | Running code | 64 |
| 2 | 6.7 | Execution code | 65 |
| 3 | 6.8 | An input screen will be displayed | 66 |
| 4 | 6.9 | Cross check your personal details | 67 |
| 5 | 6.10 | Uploading the documents | 68 |
| 6 | 6.11 | Accessing the documents | 69 |
| 7 | 6.12 | Extracting the documents | 70 |

1. **INTRODUCTION**

The Existing works consider competitive insurance markets under compulsory insurance, and analyze the effect of insurance on agents’ security expenditures. The authors of consider a competitive market with homogeneous agents, and show that insurance often deteriorates the state of network security as compared to the no-insurance scenario.

The existing studies a network of heterogeneous agents and show that the introduction of insurance cannot improve the state of network security. Study the impact of the degree of agents’ interdependence, and show that agents’ investments decrease as the degree of interdependence increases. Study a competitive market under the assumption of voluntary participation by agents, with and without moral hazard. In the absence of moral hazard, the insurer can observe agents’ investments in security, and hence premium discriminates based on the observed investments. The existing studies a network of heterogeneous agents and show that the introduction of insurance cannot improve the state of network security.

Study the impact of the degree of agents’ interdependence, and show that agents’ investments decrease as the degree of interdependence increases. Study a competitive market under the assumption of voluntary participation by agents, with and without moral hazard. In the absence of moral hazard, the insurer can observe agents’ investments in security, and hence premium discriminates based on the observed investments.

They show that such a market can provide incentives for agents to increase their investments in self- protection. However, they show that under moral hazard, the market will not provide an incentive for improving agents’ investments. The impact of insurance on the state of network security in the presence of a monopolistic welfare maximizing insurer has been studied in existing system. In these models, as the insurer’s goal is to maximize social welfare, assuming compulsory insurance, agents are incentivized through premium discrimination, i.e., agents with higher investments in security pay lower premiums.

As a result, these studies show that insurance can lead to improvement of network security. An insurance market with a monopolistic profit maximizing insurer, under the assumption of voluntary participation, has been studied in existing work, which shows that in the presence of moral hazard, insurance cannot improve network security as compared to the no-insurance scenario.

## MOTIVATION

The motivation behind using pre-screening and security interdependence in designing cyber insurance policies is to ensure the effectiveness and sustainability of the insurance coverage. By conducting a thorough risk assessment, insurers can accurately evaluate the risk exposure of potential clients. This assessment allows them to differentiate between high-risk and low-risk organizations and adjust policy terms and premiums accordingly. The motivation behind this is to ensure that the insurance coverage adequately reflects the risk

profile of the insured organization. The establishment of underwriting criteria helps insurers maintain a balanced portfolio of insured organizations.

By setting specific standards, insurers can prevent adverse selection, where only high-risk organizations seek coverage.

The motivation behind this is to mitigate the concentration of risk and ensure the long-term financial sustainability of the insurance provider. By providing risk mitigation recommendations, insurers incentivize insured organizations to improve their cybersecurity practices. This motivation stems from the understanding that organizations with robust security measures are less likely to experience cyber incidents. Encouraging risk mitigation not only reduces the likelihood of claims but also fosters a culture of proactive cybersecurity, benefiting the insured organization and the insurance provider. Cyber insurance policies are a crucial component of risk management in today's digital landscape. With the rise of cyber threats and the increasing reliance on technology, organizations and individuals face significant financial and reputational risks in the event of a cyber incident.

The primary purpose of cyber insurance is to provide financial protection against the expenses incurred due to a cyber incident. This can include costs associated with incident response and recovery, such as forensic investigations, notifying affected individuals, credit monitoring services, and legal defense. By having cyber insurance, organizations can alleviate the financial burden and potential liabilities resulting from a cyberattack or data breach.

Cyber insurance policies are tailored to meet the specific needs of each insured party. The coverage and policy terms can vary, but generally, they address a wide range of cyber risks. Insured parties are expected to take proactive measures to minimize their exposure to cyber threats and maintain proper cybersecurity practices. This may include implementing security protocols, conducting regular risk assessments, and staying up to date with industry best practices.

Overall, cyber insurance policies play a crucial role in helping organizations and individuals navigate the complex landscape of cyber risks. They provide financial support, expertise, and resources in the event of a cyber incident, helping to mitigate the potential impact on business operations, reputation, and financial stability. As cyber threats continue to evolve, having cyber insurance is becoming increasingly important for businesses of all sizes, as well as individuals who handle sensitive data. This can include costs associated with incident response and recovery, such as forensic investigations, notifying affected individuals, credit monitoring services, and legal defense. By having cyber insurance, organizations can alleviate the financial burden and potential liabilities resulting from a cyberattack or data breach.

profile of the insured organization. The establishment of underwriting criteria helps insurers maintain a balanced portfolio of insured organizations.

By setting specific standards, insurers can prevent adverse selection, where only high-risk organizations seek coverage. The motivation behind this is to mitigate the concentration of risk and ensure the long-term financial sustainability of the insurance provider. By providing risk mitigation recommendations, insurers incentivize insured organizations to improve their cybersecurity practices. This motivation stems from the understanding that organizations with robust security measures are less likely to experience cyber incidents. Encouraging risk mitigation not only reduces the likelihood of claims but also fosters a culture of proactive cybersecurity, benefiting the insured organization and the insurance provider.

## PROBLEM DEFINITION

The problem definition of cyber insurance revolves around the challenges associated with providing insurance coverage for cyber risks. Insurers face the challenge of accurately assessing the cyber risk profile of organizations seeking coverage. Determining the likelihood and potential impact of cyber incidents requires evaluating various factors such as the organization's cybersecurity measures, IT infrastructure, data protection practices, and historical incident records. The problem lies in effectively quantifying and evaluating these risks to set appropriate policy terms and premiums. Cyber insurance policies need to define the scope of coverage and the specific cyber risks that are included or excluded. The problem is determining which types of cyber incidents and losses should be covered, as the cyber threat landscape is continually evolving. Balancing the need for comprehensive coverage while ensuring the policy remains financially viable for the insurer can be challenging.

## OBJECTIVE OF PROJECT

The primary objective of cyber insurance policies is to provide financial protection and risk transfer for organizations against the potential losses and liabilities resulting from cyber incidents. By offering coverage for various cyber risks, these policies aim to assist organizations in managing the financial consequences of cyber-attacks, data breaches, system failures, and other cyber-related events. Pre-screening also enables insurers to offer risk mitigation recommendations to organizations that may not meet the initial criteria. The objective is to incentivize insured organizations to enhance their cybersecurity practices and reduce their overall risk. By suggesting improvements and best practices, insurers promote proactive risk management among their clients. By recognizing that the actions of individual insured organizations can impact the overall security landscape, insurers aim to promote a collective approach to risk management. This encourages insured organizations to adopt robust cybersecurity practices, collaborate, and share information to enhance the overall security posture of the ecosystem.

## LIMITATION OF PROJECT

* Coverage Gaps: Cyber insurance policies may have coverage limitations and exclusions, leaving certain cyber risks or specific types of incidents uncovered. The evolving nature of cyber threats makes
* challenging to design policies that can comprehensively cover all possible scenarios.
* Policy Limits: Cyber insurance policies often have predefined policy limits that may not fully cover the financial losses incurred in the event of a severe cyber incident. Organizations need to carefully evaluate the adequacy of coverage limits based on their specific risk exposure and potential financial impact.
* Loss Quantification Challenges: Assessing the financial impact of a cyber incident can be complex. Determining the exact extent of the loss and valuing the damages can be subjective and challenging, leading to disputes during the claims process.
* Claims Process Complexities: The process of filing and resolving cyber insurance claims can be intricate, involving extensive documentation, investigations, and evaluations. The specialized nature of cyber incidents and the involvement of various stakeholders, such as forensic experts, can lead to delays and complexities in claim settlements.
* Incident response and recovery: Providing financial support for investigation, mitigation, and recovery efforts following a cyber incident.
* Notification and credit monitoring services: Covering the costs of notifying affected individuals and providing credit monitoring services to mitigate the potential impact of data breaches.
* Legal and forensic services: Assisting with legal expenses and forensic investigations to determine the cause and extent of a cyber incident.
* Business interruption coverage: Compensating for lost income and additional expenses incurred due to a cyber event disrupting normal business operations.
* Extortion and ransom payments: Reimbursing the insured party for ransom payments made to cybercriminals, subject to certain conditions.
* Loss Quantification Challenges: Assessing the financial impact of a cyber incident can be complex. Determining the exact extent of the loss and valuing the damages can be subjective and challenging, leading to disputes during the claims process.
* Claims Process Complexities: The process of filing and resolving cyber insurance claims can be intricate, involving extensive documentation, investigations, and evaluations. The specialized nature of cyber incidents and the involvement of various stakeholders, such as forensic experts, can lead to delays and complexities in claim settlements.

## ORGANIZATION OF DOCUMENT

* When organizing a document on cyber insurance policies, including the roles of pre-screening and security interdependence, it is beneficial to structure it in a logical and comprehensive manner. We briefly explain the purpose and importance of cyber insurance policies then highlighted the evolving cyber threat landscape and the need for effective risk management.
* Overview of Cyber Insurance Policies: we define cyber insurance and its role in mitigating financial losses from cyber incidents. Discussed the key components of a cyber insurance policy, including coverage scope, policy limits, and exclusions. Emphasized the importance of tailoring policies to the specific needs and risk profiles of insured organizations.
* Pre-Screening in Cyber Insurance Policies: We explained the role of pre-screening in the underwriting process of cyber insurance policies. Discussed the objectives of pre-screening, such as risk assessment, underwriting criteria, and risk mitigation recommendations. Described the methods and factors involved in pre-screening, including risk assessment techniques, audits, and data collection.
* Security Interdependence in Cyber Insurance Policies we introduced the concept of security interdependence and its relevance to cyber insurance policies. We explained how security interdependence acknowledges the shared responsibility of insured organizations in managing cyber risks. We Discussed the objectives of security interdependence, including collective risk management and incentivizing strong cybersecurity practices.
* Limitations and Challenges: Outlined the limitations and challenges associated with cyber insurance policies, pre-screening, and security interdependence. Discussed potential coverage gaps, policy limits, loss quantification complexities, and challenges in the claims process. Addressed the limitations of pre-screening, such as incomplete risk assessment and uncertainties in future risk exposure. Highlighted the limitations of security interdependence, including collaboration barriers and varying security postures.

# 2. LITERATURE SURVEY

## INTRODUCTION

A literature review is a comprehensive summary of previous research on a topic. The literature review surveys scholarly articles, books, and other sources relevant to a particular area of research**.** The review should enumerate, describe, summarize, objectively evaluate and clarify this previous research. It should give a theoretical base for the research and help you (the author) determine the nature of your research. The literature review acknowledges the work of previous researchers, and in so doing, assures the reader that your work has been well conceived. It is assumed that by mentioning a previous work in the field of study, that the author has read, evaluated, and assimilated that work into the work at hand.

## EXISTING SYSTEM

The Existing works consider competitive insurance markets under compulsory insurance, and analyze the effect of insurance on agents’ security expenditures. The authors of consider a competitive market with homogeneous agents, and show that insurance often deteriorates the state of network security as compared to the no-insurance scenario. The existing studies a network of heterogeneous agents and show that the introduction of insurance cannot improve the state of network security. Study the impact of the degree of agents’ interdependence, and show that agents’ investments decrease as the degree of interdependence increases. Study a competitive market under the assumption of voluntary participation by agents, with and without moral hazard. In the absence of moral hazard, the insurer can observe agents’ investments in security, and hence premium discriminates based on the observed investments. They show that such a market can provide incentives for agents to increase their investments in self-protection. However, they show that under moral hazard, the market will not provide an incentive for improving agents’ investments. The impact of insurance on the state of network security in the presence of a monopolistic welfare maximizing insurer has been studied in existing system. In these models, as the insurer’s goal is to maximize social welfare, assuming compulsory insurance, agents are incentivized through premium discrimination, i.e., agents with higher investments in security pay lower premiums. As a result, these studies show that insurance can lead to improvement of network security. An insurance market with a monopolistic profit maximizing insurer, under the assumption of voluntary participation, has been studied in existing work, which shows that in the presence of moral hazard, insurance cannot improve network security as compared to the scenario, Various insurance

companies and specialized cyber insurance providers offer cyber insurance policies. These insurers assess cyber risks, underwrite policies, and provide coverage for organizations.

## DISADVANTAGES

* Lack of Standardization: The cyber insurance industry lacks standardization in policy terms, coverage definitions, and assessment methodologies. This lack of uniformity makes it challenging for organizations to compare policies effectively and understand the specific coverage they are purchasing.
* Coverage Limitations and Exclusions: Cyber insurance policies often have coverage limitations and exclusions that may leave organizations vulnerable to certain types of cyber risks. Common exclusions include acts of war, nation-state attacks, and intentional acts by employees. Organizations need to carefully review policy terms to understand the extent of coverage and any gaps that may exist.
* Policy Complexity and Interpretation: Cyber insurance policies can be complex and difficult to interpret, particularly in terms of defining covered incidents and the valuation of losses. The policy language may be open to interpretation, leading to disputes during the claims process and delays in claim settlements.
* Premium Costs and Affordability: Cyber insurance premiums can be costly, especially for organizations with high-risk profiles or those seeking comprehensive coverage. Smaller organizations, in particular, may find it challenging to afford the premiums associated with robust cyber insurance coverage.
* Risk Assessments and Underwriting Challenges: Insurers face difficulties accurately assessing an organization's cyber risk profile during the underwriting process. The lack of standardized risk assessment methodologies and the evolving nature of cyber threats make it challenging to effectively price policies and avoid adverse selection.
* Insufficient Risk Mitigation Incentives: While cyber insurance policies may provide some incentives for risk mitigation, there can be limitations in encouraging insured organizations to adopt and maintain robust cybersecurity practices. In some cases, organizations may rely solely on insurance coverage without prioritizing proactive risk management.
* Data Collection and Privacy Concerns: Insurers typically require organizations to provide detailed information about their cybersecurity practices, incident history, and data protection measures. This data collection raises privacy concerns, as organizations may be hesitant to disclose sensitive information or worry about the potential misuse of data by insurers.

## PROPOSED SYSTEM

In this paper, we are interested in analyzing the possibility of using cyber-insurance as an incentive for improving network security. We adopt two model assumptions which we believe better capture the current state of cyber insurance markets but differ from the majority of the existing literature; we shall assume a profit maximizing cyber insurer, and voluntary participation, i.e., agents may opt out of purchasing a contract. Under this model, we focus on two features of cyber-insurance: (i) availability of risk assessment for mitigating moral hazard, and (ii) the interdependent nature of security. The first feature is since recent advances in Internet measurements combined with machine learning techniques now allow us to perform accurate, quantitative security posture assessments at a firm level.

pre-screening, of a prospective client to mitigate moral hazard by premium discrimination and the design of customized policies. The second distinct feature, the interdependent nature of security, refers to the observation that the security standing of an entity often depends not only on its own effort towards implementing security metrics, but also on the efforts of other entities interacting with it within the eco-system. Such interdependency is crucial for the insurer’s contract design problem, as the insurer will need to offer coverage to each insured for both its losses due to direct breaches, as well as indirect losses caused by breaches of other entities.

## ALGORITHM:

**REINFORCEMENT LEARNING ALGORITHM**

Reinforcement learning (RL) is an area of machine learning inspired by behaviorist psychology [citation needed], concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The problem, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. In the operations research and control literature, reinforcement learning is called approximate dynamic programming, or neuron-dynamic programming. The problems of interest in reinforcement learning have also been studied in the theory of optimal control, which is concerned mostly with the existence and characterization of optimal solutions, and algorithms for their exact computation, and less with learning or approximation, particularly in the absence of a mathematical model of the environment. In economics and game theory, reinforcement learning may be used to explain how equilibrium may arise under bounded rationality. In machine learning, the environment is typically formulated as a Markov decision process (MDP), as many reinforcement learning algorithms for this context utilize dynamic programming techniques. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

## CONCLUSION

We studied the problem of designing cyber insurance contracts by a single profit-maximizing insurer, for both risk-neutral and risk-averse agents. While the introduction of insurance worsens network security in a network of independent agents, we showed that the result could be different in a network of interdependent agents. Specifically, we showed that security interdependency leads to a profit opportunity for the insurer, created by the inefficient effort levels exerted by free-riding agents when insurance is not available but interdependency is present; this is in addition to risk transfer that an insurer typically profits from.

# SYSTEM ANALYSIS

## CURRENT ENVIRONMENT

The existing studies a network of heterogeneous agents and show that the introduction of insurance cannot improve the state of network security. Study the impact of the degree of agents’ interdependence, and show that agents’ investments decrease as the degree of interdependence increases. Study a competitive market under the assumption of voluntary participation by agents, with and without moral hazard. In the absence of moral hazard, the insurer can observe agents’ investments in security, and hence premium discriminates based on the observed investments. They show that such a market can provide incentives for agents to increase their investments in self-protection.

## SYSTEM ARCHITECTURE

The system architecture for cyber insurance policies, incorporating the roles of pre-screening and security interdependence, can be designed as follows:

* User Interface: A user-friendly interface allows insured organizations to interact with the system, access policy information, and initiate claims.
* Policy Management: Policy Creation: Insurers can create and customize cyber insurance policies based on the needs and risk profiles of insured organizations. Coverage Scope: Define the specific cyber risks covered, policy limits, deductibles, and exclusions. Premium Calculation: Calculate the premium based on factors such as the insured organization's risk profile, industry sector, and cybersecurity measures. Policy Documentation: Generate policy documents outlining coverage terms, conditions, and obligations.
* Pre-Screening Module: Risk Assessment: Conduct risk assessments of potential insured organizations to evaluate their cybersecurity measures, risk exposure, and incident history. Underwriting Criteria: Establish criteria to determine the eligibility and terms of coverage, taking into account the results of risk assessments. Risk Mitigation Recommendations: Provide risk mitigation recommendations to insured organizations based on the assessment results, promoting stronger cybersecurity practices.
* Data Collection and Analysis: Collect and analyze relevant data from insured organizations, such as cybersecurity policies, incident response plans, security controls, and incident history. Utilize advanced analytics and machine learning techniques to assess risk levels, identify emerging threats, and refine risk assessment methodologies.
* Security Interdependence: Foster collaboration and information sharing among insured organizations through a secure platform. Encourage collective risk management by facilitating discussions, sharing best practices, and promoting industry-wide cybersecurity initiatives.
* Incident Reporting and Claims Management: Provide a streamlined process for insured organizations to report cyber incidents and initiate the claims process. Enable real-time incident tracking, communication with insurers, and documentation submission for claims assessment and settlement.

Leverage incident response resources, including forensic experts, legal support, and breach response services, to assist insured organizations during the claims process.

* Monitoring and Compliance: Continuously monitor insured organizations' cybersecurity practices and risk posture to ensure compliance with underwriting criteria and policy terms. Conduct periodic assessments and audits to verify the implementation of recommended risk mitigation measures.
* Reporting and Analytics: Generate reports and analytics on cyber risk trends, claims experience, and policy performance to facilitate informed decision-making and continuous improvement.

## Project Architecture:

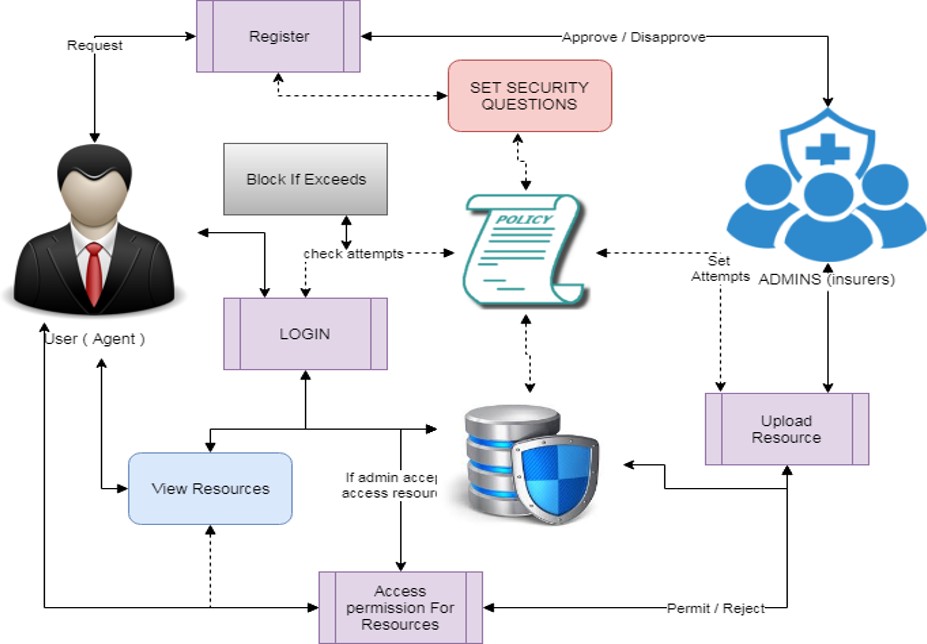


Fig: 2.2.1

Project architecture

Project architecture refers to the overall structure and organization of a software project, outlining how its components and modules interact with each other to achieve the desired functionality. It encompasses the high-level design decisions, such as the choice of programming languages, frameworks, and technologies, as well as the division of the system into different layers or modules. The project architecture aims to promote scalability, maintainability, and reusability of code, ensuring that the software is robust, efficient, and easy to manage.

## FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

**Three key considerations involved in the feasibility analysis are, ￼**

* ECONOMICAL FEASIBILITY
* TECHNICAL FEASIBILITY
* SOCIAL FEASIBILITY

## ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

## TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

## SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

A social feasibility study explores the impact of a project on society and of society on the project. For example, this type of study might look at how ambient social structure in the area will affect the number of qualified employees that may be available, or the compatibility of local residents with the project.

Social impacts on communities affected by the project include, for example, requirements for resettlement and the associated impact on quality of life and livelihoods, and impacts related to environmental alteration (e.g. on health and livelihoods)

Environmental impacts on the project location and in associated areas (eg downstream, ground water or ambient air) include effects on environmental resources due to alterations or pollutants

It will often be a mandatory regulatory requirement for assessments of social and environmental impacts to be carried out during infrastructure project development. The scope of social and environmental studies can cover:

* Quantifiable social and environmental costs and benefits
* Non quantifiable social and environmental costs and benefits
* Options for mitigating adverse impacts and the cost of mitigation.

The secondary effects should be included in the assessment. Public consultation is often a part of the social and environmental feasibility process.

The analysis should identify what type of social and environmental impact studies are needed, and the type of permits and licenses required, and should take into account health and safety standards. This information will assist the sponsor with the preparation of tender documents if the project is taken to market, and will assist bidders with the preparation of risk minimizing bids.

The final assessment of environmental and social costs and benefits is an input to the economic assessment of the project. Therefore, in addition to being a requirement from a legal and regulatory perspective, the social and environmental analysis is an important part of the assessment of the project’s overall welfare impact, as captured in the economic analysis. Social feasibility is a crucial aspect that needs to be addressed in the documentation of any project or policy. It involves assessing the potential impact on society and evaluating whether the proposed initiative aligns with social values and expectations. By incorporating social feasibility in the documentation, stakeholders can gain a better understanding of how the project will be received and accepted by the community. This includes considering the concerns and interests of various stakeholders, engaging in meaningful dialogue, and taking steps to address any potential social implications. The documentation should provide an overview of the social impact assessment conducted, stakeholder engagement strategies employed, and the measures taken to ensure that the project or policy meets social expectations and contributes positively to the well-being of the affected community.

# SYSTEM REQUIREMENTS SPECIFICATION

## REQUIREMENT ANALYSIS

Designing cyber insurance policies that incorporate pre-screening and interdependence requirement analysis can significantly enhance the effectiveness and coverage of the policies. Let's explore how these elements can be utilized in the policy design process:

Pre-Screening: Pre-screening helps insurers assess the risk profile of potential insured organizations before offering coverage. Here's how pre-screening can be integrated into the policy design:

1. Risk Assessment Questionnaire: Develop a comprehensive questionnaire that covers key cybersecurity aspects such as network security, data protection measures, incident response plans, employee training, and compliance with industry standards. Insured organizations would be required to complete this questionnaire during the application process.
2. Underwriting Criteria: Establish specific underwriting criteria based on the risk assessment questionnaire. Define thresholds for various cybersecurity indicators, such as the use of encryption, multi- factor authentication, and regular vulnerability assessments. Organizations that meet or exceed these criteria would be eligible for coverage, while those falling below the thresholds might require additional risk mitigation measures.
3. Documentation and Evidence: Require applicants to provide supporting documentation and evidence for their cybersecurity measures. This can include policies, procedures, audit reports, and certifications that demonstrate their commitment to cybersecurity best practices.
4. Risk Scoring and Premiums: Use the information collected from the risk assessment questionnaire and supporting documentation to assign a risk score to each applicant. Higher-risk organizations may be subject to higher premiums, while those with robust cybersecurity practices may receive lower premiums as an incentive.

Interdependence Requirement Analysis: Interdependence requirement analysis focuses on assessing the interconnectedness and potential cascading effects of cyber incidents across systems and networks. Here's how this analysis can be incorporated into policy design:

1. Dependency Mapping: Insurers should require insured organizations to conduct a thorough analysis of their interdependencies with critical third-party vendors, suppliers, partners, and cloud service providers. This mapping exercise should identify the potential impact of a cyber incident on these interconnected entities.
2. Incident Response Planning: Insured organizations should be mandated to have comprehensive incident response plans that include protocols for coordinating and communicating with interconnected entities during a cyber incident.

## FUNCTIONAL REQUIREMENTS

Functional requirement should include function performed by a specific screen outline work-flows performed by the system and other business or compliance requirement the system must meet. Functional requirements specify which output file should be produced from the given file they describe the relationship between the input and output of the system, for each functional requirement a detailed description of all data inputs and their source and the range of valid inputs must be specified.

specification was written, all requirements outlined in the user requirements specifications should be addressed in the functional requirements.

## NON-FUNCTIONAL REQUIREMENTS

In systems engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions. Non-functional requirements add tremendous value to business analysis. It is commonly misunderstood by a lot of people. It is important for business stakeholders, and Clients to clearly explain the requirements and their expectations in measurable terms. If the non-functional requirements are not measurable then they should be revised or rewritten to gain better clarity. For example, User stories help in mitigating the gap between developers and the user community in Agile Methodology.

## USABILITY:

Prioritize the important functions of the system based on usage patterns. Frequently used functions should be tested for usability, as should complex and critical functions. Be sure to create a requirement for this.

## RELIABILITY:

Reliability defines the trust in the system that is developed after using it for a period of time. It defines the likeability of the software to work without failure for a given time period.

The number of bugs in the code, hardware failures, and problems can reduce the reliability of the software. Your goal should be a long MTBF (mean time between failures). It is defined as the average period of time the system runs before failing.

Create a requirement that data created in the system will be retained for a number of years without the data being changed by the system.

It’s a good idea to also include requirements that make it easier to monitor system performance.

## PERFORMANCE:

What should system response times be, as measured from any point, under what circumstances? Are there specific peak times when the load on the system will be unusually high?

Think of stress periods, for example, at the end of the month or in conjunction with payroll disbursement.

## SUPPORTABILITY:

The system needs to be cost-effective to maintain.

Maintainability requirements may cover diverse levels of documentation, such as system documentation, as well as test documentation, e.g., which test cases and test plans will accompany the system.

## INPUT & OUTPUT DESIGN INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and

keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

* What data should be given as input?
* How the data should be arranged or coded?
* The dialog to guide the operating personnel in providing input.
* Methods for preparing input validations and steps to follow when error occur.

## OBJECTIVE

1. Input Design is the process of converting a user-oriented description of the input into a computer- based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow.

## OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system’s relationship to help user decision-making.

Design computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

The output form of an information system should accomplish one or more of the following objectives.

* Convey information about past activities, status or projections of the
* Future.
* Signal important events, opportunities, problems, or warnings.
* Trigger an action.
* Confirm an action.

## SYSTEMS REQUIREMENT AND SPECIFICATION

Designing a system for cyber insurance policies using pre-screening and interdependence would typically require certain hardware components to support its functionality. Here are some potential hardware requirements:

## SERVER INFRASTRUCTURE:

A robust server infrastructure is needed to host the system's software and databases.

This infrastructure should include servers capable of handling the expected workload, including processing and storing large amounts of data.

## NETWORKING:

Reliable networking equipment, such as routers, switches, and firewalls, is necessary to ensure secure and efficient data communication between the system and its users.

These components help maintain network connectivity, establish secure connections, and protect against unauthorized access.

## STORAGE:

Sufficient storage capacity is required to store and manage the data collected during the pre-screening process.

This may involve using storage servers or network-attached storage (NAS) systems to accommodate the growing volume of data.

## BACKUP:

Implementing a backup and disaster recovery system is essential to ensure data integrity and minimize downtime.

This may involve using redundant storage systems, backup servers, and offsite backup solutions to protect against data loss or system failures.

## SECURITY:

Strong security measures, such as intrusion detection and prevention systems (IDPS), antivirus software, and encryption technologies, should be implemented to protect the system from cyber threats.

Hardware security modules (HSM) can be used for secure key management and cryptographic operations, ensuring the confidentiality and integrity of sensitive data.

## SCALABILITY:

The hardware infrastructure should be designed with scalability in mind to accommodate potential growth in the number of policyholders and data volume.

This may involve using scalable cloud computing resources, load balancing techniques, and modular hardware configurations that can be easily expanded or upgraded.

It's important to note that the specific hardware requirements may vary depending on the scale and complexity of the system, as well as the expected user load. Organizations developing such systems should assess their needs and consult with IT professionals to determine the most suitable hardware setup.

## SOFTWARE REQUIREMENTS

Designing a software system for cyber insurance policies using pre-screening and interdependence involves several software requirements to support its functionality. Here are some potential software requirements:

Programming Languages: Depending on the chosen technology stack, programming languages such as Python, Java, C#, or JavaScript may be used for developing the software system.

Frameworks: Utilize web frameworks like Django, Ruby on Rails, or ASP.NET to streamline development and facilitate efficient software engineering practices.

Web Interface: Develop a user-friendly web-based interface that allows policyholders to complete the pre-screening process, submit necessary information, and review policy details.

Mobile Responsiveness: Design the user interface to be responsive, ensuring seamless accessibility from various devices, including desktops, tablets, and smartphones.

Database Management System: Utilize a reliable database management system (DBMS) like MySQL, PostgreSQL, or MongoDB to store and manage policyholder information securely.

Data Validation: Implement robust data validation techniques to ensure the accuracy and integrity of the data collected during the pre-screening process.

Data Encryption: Employ encryption algorithms and techniques to protect sensitive data, such as policyholder information and financial details.

Risk Assessment Algorithms: Develop algorithms that analyze the information provided by policyholders to assess their cyber risk exposure accurately.

Cyber Risk Modeling: Implement statistical and machine learning models to predict potential cyber risks and calculate risk scores based on various factors such as industry sector, security practices, and historical data. Authentication and Authorization: Implement secure authentication mechanisms, such as multi-factor authentication, to ensure that only authorized individuals can access the system.

# SYSTEM DESIGN

## 5.1. INTRODUCTION

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer’s goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement has been specified and analyzed, system design is the first of the three technical activities -design, code and test that is required to build and verify software.

The importance can be stated with a single word “Quality”. Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess for quality. Design is the only way that we can accurately translate a customer’s view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system – one that will be difficult to test, one whose quality cannot be assessed until the last stage. The purpose of the design phase is to plan a solution of the problem specified by the requirement document.

This phase is the first step in moving from the problem domain to the solution domain. In other words, starting with what is needed, design takes us toward how to satisfy the needs. The design of a system is perhaps the most critical factor affection the quality of the software; it has a major impact on the later phase, particularly testing, maintenance. The output of this phase is the design document. This document is like a blueprint for the solution and is used later during implementation, testing and maintenance. The design activity is often divided into two separate phases System Design and Detailed Design.

System Design, also called top-level design, aims to identify the modules that should be in the system, the specifications of these modules, and how they interact with each other to produce the desired results. At the end of the system design all the major data structures, file formats, output formats, and the major modules in the system and their specifications are decided.

During, Detailed Design, the internal logic of each of the modules specified in system design is decided. During this phase, the details of the data of a module are usually specified in a high-level design

description language, which is independent of the target language in which the software will eventually be implemented.

In system design the focus is on identifying the modules, whereas during detailed design the focus is on designing the logic for each of the modules. In other works, in system design the attention is on what components are needed, while in detailed design how the components can be implemented in software is the issue. Design is concerned with identifying software components specifying relationships among components. Specifying software structure and providing blue print for the document phase. Modularity is one of the desirable properties of large systems. It implies that the system is divided into several parts. In such a manner, the interaction between parts is minimal clearly specified.

During the system design activities, Developers bridge the gap between the requirements specification, produced during requirements elicitation and analysis, and the system that is delivered to the user.

Design is the place where the quality is fostered in development. Software design is a process through which requirements are translated into a representation of software.

## DFD DIAGRAMS:

A graphical tool used to describe and analyze the moment of data through a system manual or automated including the process, stores of data, and delays in the system. Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also known as a data flow graph or a bubble chart.

DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system’s structure charts. The Basic Notation used to create a DFD’s are as follows:

1. Dataflow: Data move in a specific direction from an origin to a destination.

Fig: 5.1 Arrow – Flow of information

1. Process: People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.
2. Source: External sources or destination of data, which may be People, programs, organizations or other entities.

Fig: 5.2 Rectangle – Source of information

1. Data Store: Here data are stored or referenced by a process in the System.

Fig: 5.3 semi-rectangle – Data storage

## UML CLASS DIAGRAM

Class diagrams are the backbone of almost every object-oriented method including UML. They describe the static structure of a system.

Basic Class Diagram Symbols and Notations. Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition, and write operations into the third.

A UML (Unified Modeling Language) class diagram is a visual representation of the structure and relationships of classes within a software system. It provides a high-level overview of the system's object-oriented design, showcasing the classes, their attributes, methods, and associations. UML class diagrams are widely used in software development as a communication and documentation tool among stakeholders, designers, and developers.

The main components of a UML class diagram include classes, attributes, methods, relationships, and multiplicity indicators. Classes are depicted as rectangles, with the class name in the top section, followed by attributes and methods in the middle and bottom sections, respectively. Attributes represent the properties or characteristics of a class, while methods define the behaviors or operations it can perform.

Relationships between classes are illustrated using various types of connectors, such as associations, aggregations, compositions, and inheritances. Associations depict the relationships between classes, indicating how they are connected and the cardinality of the relationship. Aggregations and compositions represent whole-part relationships, where one class contains or is composed of other classes. Inheritance, represented by arrows and solid lines, indicates the hierarchical relationships between classes, where a subclass inherits properties and behaviors from a superclass.

UML class diagrams aid in understanding the overall structure of a software system, identifying the key classes and their interactions. They help visualize the relationships between classes, facilitating the identification of potential design flaws or improvements. Class diagrams also serve as a reference for developers during the implementation phase, guiding the creation of class hierarchies, attributes, and methods.

Associations represent static relationships between classes. Place association names above, on, or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other. Note: It's uncommon to name both the association and the class roles.

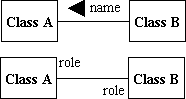
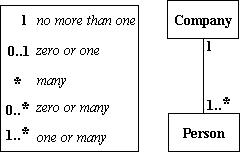


Fig: 5.7

Associations

## MULTIPLICITY

Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for one company only.

multiplicity describes the cardinality or number of instances that are allowed in a relationship between two classes. It specifies how many objects of one class can be associated with or connected to objects of another class.

In UML diagrams, multiplicity is represented using notations near the association lines or connectors between classes. The notation consists of two integers separated by a comma or a range of integers within square brackets. The first integer represents the minimum number of instances, and the second integer represents the maximum number of instances that can participate in the relationship.

Here are some common examples of multiplicity notations in UML:

* "1" indicates exactly one instance.
* "0..1" or "optional" means that zero or one instance can be associated.
* "0..*" or "*" denotes zero or more instances.
* "1..\*" or "1..n" signifies one or more instances.
* "n" represents a specific number of instances, such as "2" or "5".

For instance, consider a UML diagram representing a class diagram for a library system. If we have a class called "Book" and another class called "Library," the association between them could have a multiplicity notation of "0..\*" from the perspective of the "Library" class, indicating that a library can have zero or more books.

Multiplicity in UML is important for understanding the cardinality and constraints of relationships between classes, helping to define the structure and behaviour of a system in the modelling phase.

## CONSTRAINT

## In Unified Modeling Language (UML), constraints are used to specify and define various conditions, rules, or limits that must be satisfied within a system or model. Constraints provide additional information and restrictions on the elements of a UML model, helping to ensure correctness, consistency, and behavior adherence. They can be applied to different UML elements, including classes, attributes, operations, relationships, and more.

## Here are some common types of constraints used in UML:

## Structural Constraints: These constraints define the rules and limitations related to the structure of the system. For example, they can specify the range of valid values for an attribute, the cardinality of associations, or the inheritance hierarchy between classes.

## Behavioral Constraints: These constraints focus on the behavior and dynamics of the system. They can describe the ordering of events, the conditions under which an operation can be invoked, or the states and transitions of a state machine.

## OCL Constraints: OCL (Object Constraint Language) is a textual language used to express constraints in UML. OCL constraints provide a more expressive and formal way to specify conditions and rules. They can be applied to various UML elements using OCL expressions.

## Multiplicity Constraints: These constraints specify the allowed number of instances or occurrences in a relationship. They define the minimum and maximum cardinality of associations between classes.

## Invariant Constraints: Invariant constraints specify conditions that must always hold true for a particular class or system. They define invariants that should not be violated at any point during the system's execution.

## Pre- and Post-Conditions: These constraints define the conditions that must be true before and after an operation is executed. They specify the expected state of the system or object before and after a method invocation.

## Dependency Constraints: These constraints describe the dependencies between elements. They can express requirements, associations, or constraints related to the relationships between different parts of the system.

## Constraints play a crucial role in UML modelling by providing additional information, rules, and limitations that help in the analysis, design, and documentation of systems. They aid in validating the integrity and correctness of the model and guide developers in implementing the desired system behaviour.

Figure: 5.8 Cardinality

Place constraints inside curly braces {}.

http://wc1.smartdraw.com/resources/tutorials/images/uml_constraint.gif

Fig: 5.9

Simple Constraint

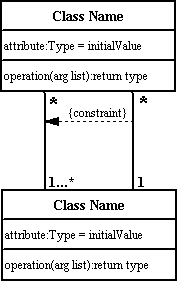
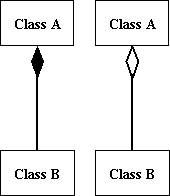


Fig: 5.10

Multi constraints

## COMPOSITION AND AGGREGATION

Composition is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part. Illustrate composition with a filled diamond. Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond end in both a composition and aggregation relationship points toward the "whole" class or the aggregate



## GENERALIZATION

## In Unified Modeling Language (UML), generalization is a fundamental concept used to represent inheritance and the relationship between classes. It allows one class, called the superclass or parent class, to be specialized into one or more subclasses or child classes. The subclasses inherit the characteristics (attributes, operations, relationships) of the superclass and can also add their own unique characteristics.

## In UML diagrams, generalization is represented using a solid line with an arrowhead pointing from the subclass to the superclass. The arrowhead indicates the direction of the inheritance relationship, from the specialized (subclass) to the generalized (superclass) class. The superclass is placed at the top, and the subclasses are placed below it.

## The generalization relationship is often referred to as an "is-a" relationship, as the subclasses are considered to be specialized versions of the superclass. For example, if we have a superclass called "Animal" and subclasses called "Cat" and "Dog," we can say that "Cat is an Animal" and "Dog is an Animal."

## Key points about generalization in UML include:

## Inheritance: Subclasses inherit the attributes, operations, and relationships defined in the superclass. They can access and use these inherited elements and may also add their own specific elements.

## Specialization: Subclasses specialize or refine the characteristics of the superclass. They provide more specific details or behavior.

## Code Reusability: Generalization promotes code reuse by allowing common attributes and behavior to be defined in the superclass and inherited by subclasses.

## Polymorphism: Polymorphism is supported through generalization. Objects of the subclass can be treated as objects of the superclass, allowing substitution and flexibility in usage.

## Inherited Relationships: Subclasses inherit the relationships of the superclass, including associations, aggregations, or dependencies. They can participate in these relationships along with any additional relationships specific to them.

## Generalization is a powerful mechanism in UML that promotes code reuse, abstraction, and hierarchy in object-oriented modeling. It helps to structure and organize classes into a hierarchical relationship, capturing the shared characteristics and variations among classes.

Fig: 5.11

Composition and aggregations

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So, the class Honda would have a generalization relationship with the class car.

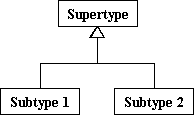


Fig: 5.12

generalizations

In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class. On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the super class.

## USE CASE UML:

Use case diagrams model the functionality of a system using actors and use cases. Use cases are services or functions provided by the system to its users.

Basic Use Case Diagram Symbols and Notations System

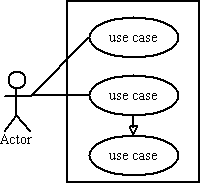
Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.

Fig: 5.13

Use case diagram

Draw use cases using ovals. Label with ovals with verbs that represent the system's functions.

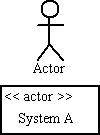
Use Case

Fig: 5.14

System functions

## ACTORS

Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.



## RELATIONSHIPS

Fig: 5.15

Actor entity

Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.

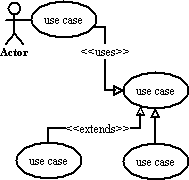


Fig: 5.16

Relations

## SEQUENCE DIAGRAM

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.

Basic Sequence Diagram Symbols and Notations

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

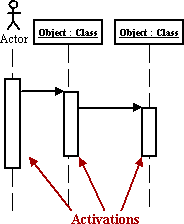
Class roles

Fig: 5.17

Class roles

## ACTIVATION

Activation boxes represent the time an object needs to complete a task.



## MESSAGES

Fig: 5.18

Activations

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from receiver.

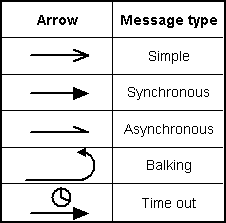
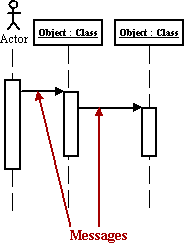


Fig: 5.19

Various message types for Sequence and Collaboration diagrams

## LIFE LINES

A lifeline diagram, also known as a sequence diagram, is a visual representation used in software engineering and system modelling to depict the interactions and order of events between various components or objects.

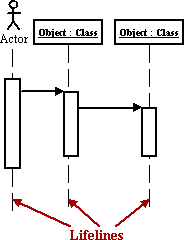


Fig: 5.20

Lifeline diagram

## LOOPS

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ]. In Unified Modeling Language (UML), loops can be represented in activity diagrams using control nodes and control flows. Activity diagrams are used to visualize the flow of activities or processes within a system or a specific scenario.

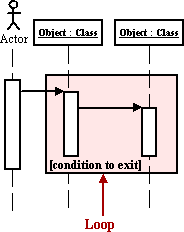


Fig: 5.21

Looping’s

## CLASS ROLES

Class roles describe how objects behave. Use the UML object symbol to illustrate class roles, but don't list object attributes.

Class roles

Fig: 5.22

Class roles

## ASSOCIATIONS

Association roles describe how an association will behave given a particular situation. You can draw association roles using simple lines labeled with stereotypes.

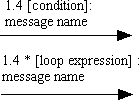
Association roles

## MESSAGES

Fig: 5.23

Association roles

Unlike sequence diagrams, collaboration diagrams do not have an explicit way to denote time and instead number messages in order of execution. Sequence numbering can become nested using the Dewey decimal system. For example, nested messages under the first message are labeled 1.1, 1.2, 1.3, and so on. The a condition for a message is usually placed in square brackets immediately following the sequence number.



## ACTIVITY DIAGRAM

Fig: 5.24

Message diagram

An activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a special kind of state chart diagram, it uses some of the same modeling conventions.

An activity diagram is a graphical representation used in software engineering to model the workflow or behaviour of a system, process, or use case. It is part of the Unified Modelling Language (UML) and provides a visual way to depict the sequence of activities, decisions, and flows of control within a system.

## ACTION STATES

Action states represent the non-interruptible actions of objects. You can draw an action state in Smart Draw using a rectangle with rounded corners.

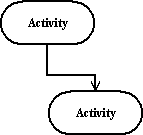
Action states

Fig: 5.25

Activity diagram

## ACTION FLOWS

Action flow arrows illustrate the relationships among action states.



## OBJECT FLOW

Fig: 5.26

Action flows

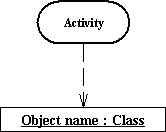
Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.

Fig: 5.27

Flowing of information

## INITIAL STATE

A filled circle followed by an arrow represents the initial action state.

Initial State

Fig: 5.28

Initial states

## FINAL STATE

An arrow pointing to a filled circle nested inside another circle represents the final action state.

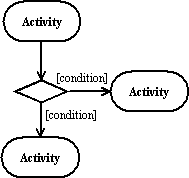
Final State

Fig: 5.29

Final states

## BRANCHING

A diamond represents a decision with alternate paths. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."

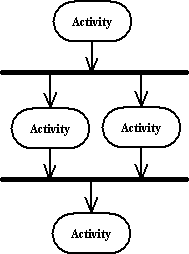


## SYNCHRONIZATION

Fig: 5.30

Branching

A synchronization bar helps illustrate parallel transitions. Synchronization is also called forking and joining. An activity diagram is a graphical representation used in software engineering to model the workflow or behaviours of a system, process, or use case. It is part of the Unified Modelling Language (UML) and provides a visual way to depict the sequence of activities, decisions, and flows of control within a system.



## SWIM LANES

Fig: 5.31

Synchronization

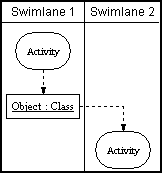


Fig: 5.32

### State chart diagram

A state chart diagram shows the behavior of classes in response to external stimuli. This diagram models the dynamic flow of control from state to state within a system.

Basic State chart Diagram Symbols and Notations

## STATES

States represent situations during the life of an object. You can easily illustrate a state in Smart Draw by using a rectangle with rounded corners.

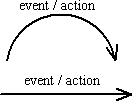
States

Fig: 5.33

State representation

## TRANSITIONS

A solid arrow represents the path between different states of an object. Label the transition with the event that triggered it and the action that results from it.



## INITIAL STATE

Fig: 5.34

Initial State

Fig: 5.35

Initial states

## FINAL STATE

An arrow pointing to a filled circle nested inside another circle represents the object's final state.

Final State

Fig: 5.36

Final states

## SPLITTING OF CONTROL

A short heavy bar with two transitions entering it represents a synchronization of control. A short heavy bar with two transitions leaving it represents a splitting of control that creates multiple states.

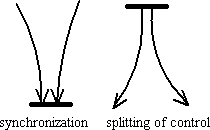


Fig: 5.37

Synchronous controls

A component diagram describes the organization of the physical components in a system.

Basic Component Diagram Symbols and Notations

## COMPONENT

A component is a physical building block of the system. It is represented as a rectangle with tabs.

[Learn how to resize grouped objects like components.](http://www.smartdraw.com/resources/tutorials/Objects)

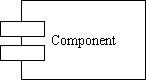


Fig: 5.38

Component diagram

## INTERFACES

An interface describes a group of operations used or created by components. In Unified Modeling Language (UML), an interface is a contract or specification that defines a set of operations, methods, and attributes that a class or component must implement. It represents a collection of behaviors that can be provided by multiple classes.

An interface in UML is depicted using a class-like notation with the stereotype "<<interface>>" placed above the name of the interface. The interface name is usually written in italics to differentiate it from classe

Fig: 5.39

Interface notation

## DEPENDENCIES

When it comes to software development or project management, dependencies refer to the relationships or connections between different tasks or components that determine their order of execution or completion. Dependencies help in understanding the order in which activities need to be performed, ensuring that certain prerequisites are met before proceeding with other

## COMPONENT

A component diagram is a type of UML (Unified Modelling Language) diagram that represents the structure and organization of software components in a system. It provides a visual representation of the high- level components and their relat

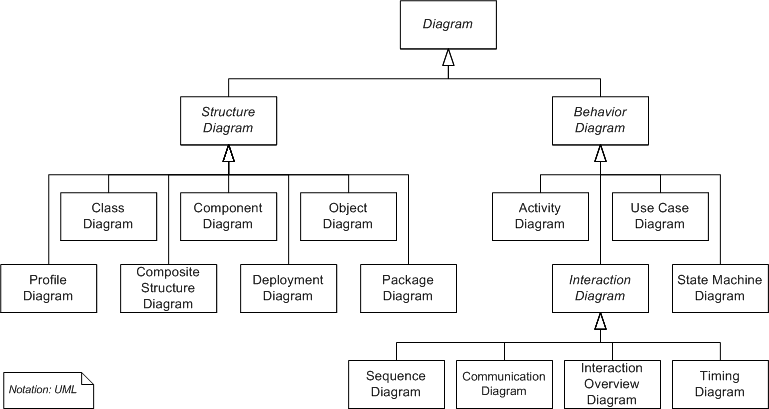


Fig: 5.44

Hierarchy of UML diagrams

UML combines best techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML has synthesized the notations of the Brooch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems.

## USECASE DAIGRAM:

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases.So we can say that uses cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system. The actors can be human user, some internal applications or may be some external applications. So in a brief when we are planning to draw an use case diagram we should have the following items identified.

* + - * Functionalities to be represented as an use case
      * Actors
      * Relationships among the use cases and actors.

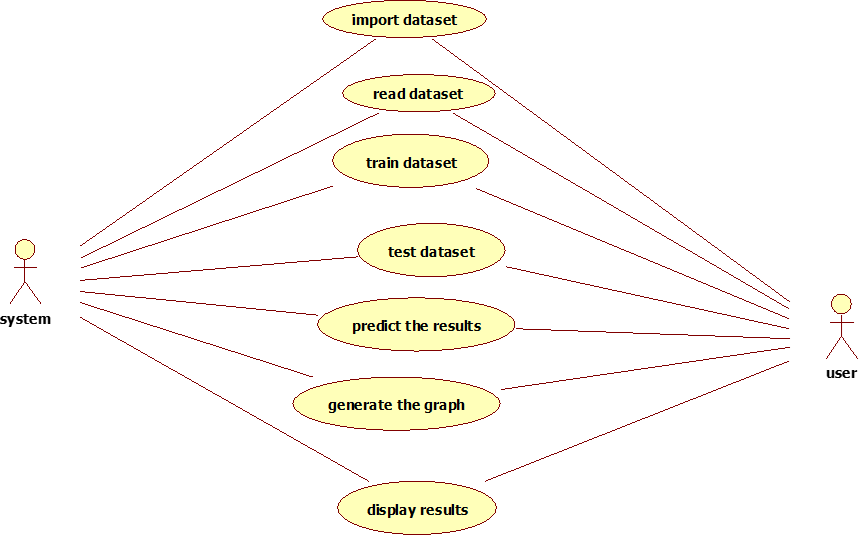


Fig: 5.45

Use case diagram for performing user actions

## SEQUENCE DAIGRAM

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines ("lifelines"), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

A sequence diagram is a visual representation of the interactions and order of messages exchanged between objects or components in a system. It provides a dynamic view of how different parts of the system collaborate to achieve a particular functionality or scenario. The diagram typically consists of lifelines representing the objects or components involved and messages exchanged between them. The purpose of a sequence diagram is to depict the flow of control and communication among these objects, showing the sequence in which messages are sent and received. It helps in understanding the behavior of the system, identifying potential bottlenecks or synchronization issues, and verifying the correctness of the system's logic.

In a sequence diagram, each lifeline represents an object or component involved in the interaction, and the vertical ordering of the lifelines represents their respective timelines. Messages between lifelines are represented by arrows, indicating the direction of communication. The sequence of messages determines the order of interactions, and the diagram illustrates how the objects collaborate to accomplish a specific task or scenario.

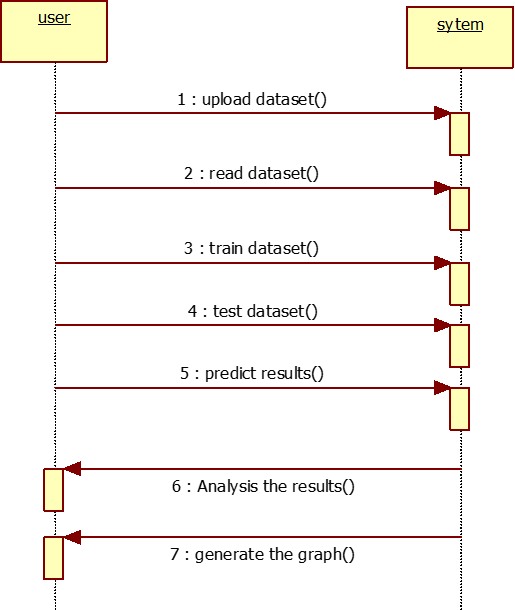


Fig: 5.46

Sequence diagram for Sequence diagram showing the process of execution

The above figure illustrates sequence diagram, each lifeline represents an object or component involved in the interaction, and the vertical ordering of the lifelines represents their respective timelines

## COLLEBARATION DAIGRAM:

A collaboration diagram, also called a communication diagram or interaction diagram. A Collaboration diagram is easily represented by modeling objects in a system and representing the associations between the objects as links. The interaction between the objects is denoted by arrows. To identify the sequence of invocation of these objects, a number is placed next to each of these arrows. A sophisticated modeling tool can easily convert a collaboration diagram into a sequence diagram and the vice versa. Hence, the elements of a Collaboration diagram are essentially the same as that of a Sequence diagram.

A communication diagram, also known as a collaboration diagram, is a type of UML diagram that depicts the interactions and relationships between objects or components within a system. It emphasizes the communication and message exchange between objects to achieve a specific goal or scenario.

Communication diagrams are useful for visualizing the dynamic behaviour and flow of information in a system, making them valuable for understanding the interactions among different components.

In a communication diagram, objects or components are represented as lifelines, which are vertical lines that extend across the diagram. Each lifeline represents an instance of a class or a component. Messages, represented by arrows, show the flow of communication between lifelines. The messages may be labelled to indicate the content or purpose of the communication, providing

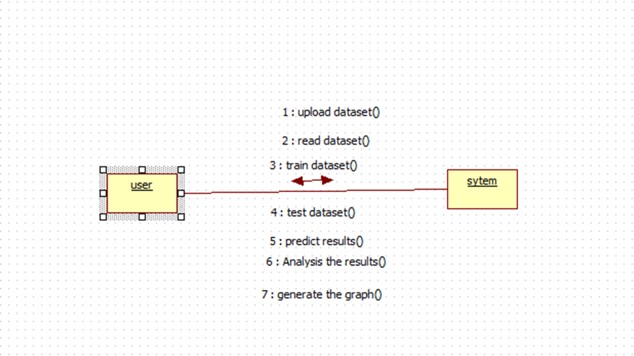


Fig: 5.47

Collaboration diagram for Flow of process

## CLASS DIAGRAM:

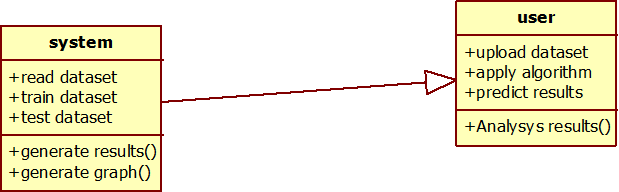


Fig: 5.48

Class diagram for Representation of class and its attributes

## ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control. An activity diagram consists of nodes, which represent the activities or actions performed within the system, and arrows, which illustrate the transitions or flow between these activities. The diagram begins with a start node, indicating the starting point of the process, and ends with an end node, indicating the completion or termination of the process.

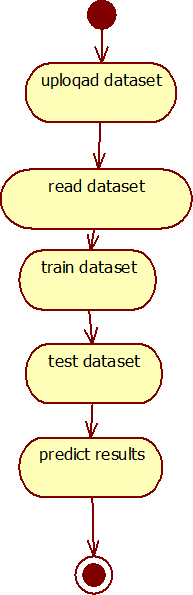


Fig: 5.49

Activity diagram

An activity diagram is a type of UML (Unified Modeling Language) diagram that visually represents the flow of activities, actions, and decisions within a system or process. It depicts the workflow from start to finish, showing the sequence of activities and their dependencies. Here is an explanation of an activity diagram:

An activity diagram consists of nodes, which represent the activities or actions performed within the system, and arrows, which illustrate the transitions or flow between these activities. The diagram begins with a start node, indicating the starting point of the process, and ends with an end node, indicating the completion or termination of the process. Activity diagrams are a type of UML (Unified Modeling Language) diagram that represents the flow of activities or processes within a system. They provide a visual representation of the sequential and parallel activities, decision points, and the flow of control from one activity to another.

# .6. IMPLEMENTATION

## MODULE & DESCRIPTION

* + 1. **MODULES**

Below are following modules offered in this project:

* Pre-screening
* Threat detection
* Limited resources
* Analysis

## MODULES DESCRIPTION

* + - 1. **PRESCREENING**
         * Normally the screening process of the system can be done by login system but with this system username and password alone not enough to authenticate the system. The security questions will be set to each user separately to make sure the correct user logged in or not. It sets the limit the access of users from threats. The class can be limited by admin while registering and admin alone approve the user’s entry to system.

## THREAT DETECTION

* + - * + The threat can be detected with the help of prescreening technique. Threats can be illegal access to system with more than five times trying to access the particular account with different act. The Insurance policies can be set to different users. According to policies users can be access. Within certain number of attempts goes wrong the user can be blocked and need to request admin to unblock again.

## LIMIT RESOURCE

* + - * + Admin is the authorized person to control polices and rules breaches. The wrong access of particular document more than certain number of time that is described in the policy can be blocked by admin and gets the intimation of breaches to admin. Then according to request by admin to user can be block or unblock the resources which are uploaded by admin/user.

## ANALYSIS

* + - * + The analysis of the system is done in this module. The proposed algorithm’s efficiency is calculated here. The comparison of various factors can be handy to calculate and visualize in the graphs such as pie chart, bar chart, line chart. The data to plot the graph is taken from the system which is done.

## TECHNOLOGY DESCRIPTION-PYTHON

* + 1. **INTRODUCTION TO PYTHON**

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit [code blocks](https://en.wikipedia.org/wiki/Code_block) rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer [lines of code](https://en.wikipedia.org/wiki/Source_lines_of_code) than might be used in languages such as [C++](https://en.wikipedia.org/wiki/C%2B%2B)or [Java.](https://en.wikipedia.org/wiki/Java_(programming_language)) It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many [operating systems](https://en.wikipedia.org/wiki/Operating_system). [CPython](https://en.wikipedia.org/wiki/CPython), the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python, is [open source](https://en.wikipedia.org/wiki/Open_source) software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation). Python features a [dynamic type](https://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](https://en.wikipedia.org/wiki/Memory_management). It supports multiple [programming paradigms,](https://en.wikipedia.org/wiki/Programming_paradigm) including [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), [imperative,](https://en.wikipedia.org/wiki/Imperative_programming) [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural,](https://en.wikipedia.org/wiki/Procedural_programming) and has a comprehensive [standard library.](https://en.wikipedia.org/wiki/Standard_library)

## PYTHON

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

## USED FOR:

* web development (server-side),
* software development,
* mathematics,
* system scripting.

## WHAT PYTHON CAN DO:

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

## WHY PYTHON:

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-orientated way or a functional way.

## GOOD TO KNOW:

* The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
* In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

## SYNTAX:

* Python was designed for readability, and has some similarities to the English language with influence from mathematics.
* Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
* Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

## INSTALLATION:

* Many PCs and Macs will have python already installed.
* To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):
* C:\Users\Your Name>python --version
* To check if you have python installed on a Linux or Mac, then on linux open the command line or on Mac open the Terminal and type:
* python --version
* If you find that you do not have python installed on your computer, then you can download it for free from the following website: https:/[/www](http://www.python.org/).[python.org/](http://www.python.org/)

## QUICKSTART:

* Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.
* The way to run a python file is like this on the command line:
* C:\Users\Your Name>python helloworld.py
* Where "helloworld.py" is the name of your python file.
* Let's write our first Python file, called helloworld.py, which can be done in any text editor.
* helloworld.py
* print("Hello, World!")
* The output should read:
* Hello, World!
* Congratulations, you have written and executed your first Python program.

## THE PYTHON COMMAND LINE

* To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.
* Type the following on the Windows, Mac or Linux command line:
* C:\Users\Your Name>python
* Or, if the "python" command did not work, you can try "py":
* C:\Users\Your Name>py C:\Users\Your Name>python

Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32 Type "help", "copyright", "credits" or "license" for more information.

>>>print("Hello, World!")

Which will write "Hello, World!" in the command line:

C:\Users\Your Name>python

Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32 Type "help", "copyright", "credits" or "license" for more information.

>>>print("Hello, World!") Hello, World!

Whenever you are done in the python command line, you can simply type the following to quit the python command line interface:

exit()

Python applications will often use packages and modules that don’t come as part of the standard library. Applications will sometimes need a specific version of a library, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library’s interface.

This means it may not be possible for one Python installation to meet the requirements of every application. If application A needs version 1.0 of a particular module but application B needs version 2.0, then the requirements are in conflict and installing either version 1.0 or 2.0 will leave one application unable to run.

The solution for this problem is to create a virtual environment, a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.

Different applications can then use different virtual environments. To resolve the earlier example of conflicting requirements, application A can have its own virtual environment with version 1.0 installed while application

B has another virtual environment with version 2.0. If application B requires a library be upgraded to version 3.0, this will not affect application A’s environment.

## CREATING VIRTUAL ENVIRONMENTS

The module used to create and manage virtual environments is called venv. venv will usually install the most recent version of Python that you have available. If you have multiple versions of Python on your system, you can select a specific Python version by running python3 or whichever version you want.

To create a virtual environment, decide upon a directory where you want to place it, and run the venv module as a script with the directory path:

python3 -m venv tutorial-env

This will create the tutorial-env directory if it doesn’t exist, and also create directories inside it containing a copy of the Python interpreter, the standard library, and various supporting files.A common directory location for a virtual environment is .venv. This name keeps the directory typically hidden in your shell and thus out of the way while giving it a name that explains why the directory exists. It also prevents clashing with .env environment variable definition files that some tooling supports.

Once you’ve created a virtual environment, you may activate it.

On Windows, run:

tutorial-env\Scripts\activate.bat On Unix or MacOS, run:

source tutorial-env/bin/activate

(This script is written for the bash shell. If you use the csh or fish shells, there are alternate activate.csh and activate.fish scripts you should use instead.)

Activating the virtual environment will change your shell’s prompt to show what virtual environment you’re using, and modify the environment so that running python will get you that particular version and installation of Python. For example:

$ source ~/envs/tutorial-env/bin/activate (tutorial-env) $ python

Python 3.5.1 (default, May 6 2016, 10:59:36)

...

>>> import sys

>>>sys.path

You can install, upgrade, and remove packages using a program called pip. By default pip will install packages from the Python Package Index, <https://pypi.org>. You can browse the Python Package Index by going to it in your web browser, or you can use pip’s limited search feature:

(tutorial-env) $ pip search astronomy

skyfield - Elegant astronomy for Python

gary - Galactic astronomy and gravitational dynamics.

novas - The United States Naval Observatory NOVAS astronomy library astroobs - Provides astronomy ephemeris to plan telescope observations PyAstronomy - A collection of astronomy related tools for Python.

...

pip has a number of subcommands: “search”, “install”, “uninstall”, “freeze”, etc. (Consult the Installing Python Modules guide for complete documentation for pip.)

You can install the latest version of a package by specifying a package’s name: (tutorial-env) $ pip install novas

Collecting novas

Downloading novas-3.1.1.3.tar.gz (136kB) Installing collected packages: novas Running setup.py install for novas Successfully installed novas-3.1.1.3

You can also install a specific version of a package by giving the package name followed by == and the version number:

(tutorial-env) $ pip install requests==2.6.0 Collecting requests==2.6.0

Using cached requests-2.6.0-py2.py3-none-any.whl Installing collected packages: requests

Successfully installed requests-2.6.0

If you re-run this command, pip will notice that the requested version is already installed and do nothing. You can supply a different version number to get that version, or you can run pip install --upgrade to upgrade the package to the latest version:

(tutorial-env) $ pip install --upgrade requests Collecting requests

Installing collected packages: requests Found existing installation: requests 2.6.0 Uninstalling requests-2.6.0:

Successfully uninstalled requests-2.6.0 Successfully installed requests-2.7.0

pip uninstall followed by one or more package names will remove the packages from the virtual environment.

### pip show will display information about a particular package:

(tutorial-env) $ pip show requests

---

Metadata-Version: 2.0 Name: requests Version: 2.7.0

Summary: Python HTTP for Humans. Home-page: [http://python-requests.org](http://python-requests.org/) Author: Kenneth Reitz

Author-email: [me@kennethreitz.com](mailto:me@kennethreitz.com) License: Apache 2.0

Location: /Users/akuchling/envs/tutorial-env/lib/python3.4/site-packages Requires:

pip list will display all of the packages installed in the virtual environment:

(tutorial-env) $ pip list novas (3.1.1.3)

numpy (1.9.2)

pip (7.0.3)

requests (2.7.0)

setuptools (16.0)

pip freeze will produce a similar list of the installed packages, but the output uses the format that pip install expects. A common convention is to put this list in a requirements.txt file:

(tutorial-env) $ pip freeze >

requirements.txt (tutorial-env) $ cat requirements.txt novas==3.1.1.3

numpy==1.9.2 requests==2.7.0

The requirements.txt can then be committed to version control and shipped as part of an application.

Successfully installed novas-3.1.1.3 numpy-1.9.2 requests-2.7.0

pip has many more options. Consult the Installing Python Modules guide for complete documentation for pip. When you’ve written a package and want to make it available on the Python Package Index, consult the Distributing Python Modules guide.

## CROSS PLATFORM

Platform. Architecture (executable=sys.executable, bits='', linkage='')

Queries the given executable (defaults to the Python interpreter binary) for various architecture information. Returns a tuple (bits, linkage) which contain information about the bit architecture and the linkage format used for the executable. Both values are returned as strings.

Values that cannot be determined are returned as given by the parameter presets. If bits is given as '', the sizeof(pointer) (or sizeof(long) on Python version < 1.5.2) is used as indicator for the supported pointer size. The function relies on the system’s file command to do the actual work. This is available on most if not all Unix platforms and some non-Unix platforms and then only if the executable points to the Python interpreter. Reasonable defaults are used when the above needs are not met.

Note On Mac OS X (and perhaps other platforms), executable files may be universal files containing multiple architectures.

To get at the “64-bitness” of the current interpreter, it is more reliable to query the sys.maxsize attribute: is\_64bits = sys.maxsize> 2\*\*32

platform.machine ()

Returns the machine type, e.g. 'i386'. An empty string is returned if the value cannot be determined. platform.node ()

Returns the computer’s network name (may not be fully qualified!). An empty string is returned if the value cannot be determined.

platform. Platform(aliased=0, terse=0)

Returns a single string identifying the underlying platform with as much useful information as possible.

The output is intended to be human readable rather than machine parseable. It may look different on different platforms and this is intended.

If aliased is true, the function will use aliases for various platforms that report system names which differ from their common names, for example SunOS will be reported as Solaris. The system\_alias() function is used to implement this.

Setting terse to true causes the function to return only the absolute minimum information needed to identify the platform.

platform.processor()

Returns the (real) processor name, e.g. 'amdk6'.

An empty string is returned if the value cannot be determined. Note that many platforms do not provide this information or simply return the same value as for machine(). NetBSD does this.

platform.python\_build()

Returns a tuple (buildno, builddate) stating the Python build number and date as strings. platform.python\_compiler()

Returns a string identifying the compiler used for compiling Python. platform.python\_branch()

Returns a string identifying the Python implementation SCM branch. New in version 2.6.

platform.python\_implementation()

Returns a string identifying the Python implementation. Possible return values are: ‘CPython’, ‘IronPython’, ‘Jython’, ‘PyPy’.

New in version 2.6. platform.python\_revision()

Returns a string identifying the Python implementation SCM revision. New in version 2.6.

platform.python\_version()

Returns the Python version as string 'major.minor.patchlevel'.

Note that unlike the Python sys.version, the returned value will always include the patchlevel (it defaults to 0).

platform.python\_version\_tuple()

Returns the Python version as tuple (major, minor, patchlevel) of strings.

Note that unlike the Python sys.version, the returned value will always include the patchlevel (it defaults to '0').

platform.release()

Returns the system’s release, e.g. '2.2.0' or 'NT' An empty string is returned if the value cannot be determined. platform.system()

Returns the system/OS name, e.g. 'Linux', 'Windows', or 'Java'. An empty string is returned if the value cannot be determined.

platform.system\_alias(system, release, version)

Returns (system, release, version) aliased to common marketing names used for some systems. It also does some reordering of the information in some cases where it would otherwise cause confusion. platform.version()

Returns the system’s release version, e.g. '#3 on degas'. An empty string is returned if the value cannot be determined.

platform.uname()

Fairly portable uname interface. Returns a tuple of strings (system, node, release, version, machine, processor) identifying the underlying platform.

Note that unlike the os.uname() function this also returns possible processor information as additional tuple entry.

Entries which cannot be determined are set to ''.

Java Platform

platform.java\_ver(release='', vendor='', vminfo=('', '', ''), osinfo=('', '', '')) Version interface for Jython.

Returns a tuple (release, vendor, vminfo, osinfo) with vminfo being a tuple (vm\_name, vm\_release, vm\_vendor) and osinfo being a tuple (os\_name, os\_version, os\_arch). Values which cannot be determined are set to the defaults given as parameters (which all default to '').

Windows Platform

platform.win32\_ver(release='', version='', csd='', ptype='')

Get additional version information from the Windows Registry and return a tuple (release, version, csd, ptype) referring to OS release, version number, CSD level (service pack) and OS type (multi/single processor).

As a hint: ptype is 'Uniprocessor Free' on single processor NT machines and 'Multiprocessor Free' on multi processor machines. The ‘Free’ refers to the OS version being free of debugging code. It could also state ‘Checked’ which means the OS version uses debugging code, i.e. code that checks arguments, ranges, etc.

Note This function works best with Mark Hammond’s win32all package installed, but also on Python 2.3 and later (support for this was added in Python 2.6). It obviously only runs on Win32 compatible platforms.

Win95/98 specific

platform.popen(cmd, mode='r', bufsize=None)

Portable popen() interface. Find a working popen implementation preferring win32pipe.popen(). On Windows NT, win32pipe.popen() should work; on Windows 9x it hangs due to bugs in the MS C library.

Mac OS Platform

platform.mac\_ver(release='', versioninfo=('', '', ''), machine='')

Get Mac OS version information and return it as tuple (release, versioninfo, machine) with versioninfo being a tuple (version, dev\_stage, non\_release\_version).

Entries which cannot be determined are set to ''. All tuple entries are strings.

Unix Platforms

platform.dist(distname='', version='', id='', supported\_dists=('SuSE', 'debian', 'redhat', 'mandrake', ...))

This is an old version of the functionality now provided by linux\_distribution(). For new code, please use

platform.linux\_distribution(distname='', version='', id='', supported\_dists=('SuSE', 'debian', 'redhat', 'mandrake', ...), full\_distribution\_name=1)

Tries to determine the name of the Linux OS distribution name.

supported\_dists may be given to define the set of Linux distributions to look for. It defaults to a list of currently supported Linux distributions identified by their release file name.

If full\_distribution\_name is true (default), the full distribution read from the OS is returned. Otherwise the short name taken from supported\_dists is used.

Returns a tuple (distname,version,id) which defaults to the args given as parameters. id is the item in parentheses after the version number. It is usually the version codename.

Note This function is deprecated since Python 3.5 and removed in Python 3.8. See alternative like the distro package.

New in version 2.6.

platform.libc\_ver(executable=sys.executable, lib='', version='', chunksize=2048)

Tries to determine the libc version against which the file executable (defaults to the Python interpreter) is linked. Returns a tuple of strings (lib, version) which default to the given parameters in case the lookup fails. Note that this function has intimate knowledge of how different libc versions add symbols to the executable is probably only usable for executables compiled using gcc. The file is read and scanned in chunks of chunksize bytes.

## USING THE PYTHON INTERPRETER

The Python interpreter is usually installed as /usr/local/bin/python3.8 on those machines where it is available; putting /usr/local/bin in your Unix shell’s search path makes it possible to start it by typing the command.

**PYTHON 3.8**

Since the choice of the directory where the interpreter lives is an installation option, other places are possible; check with your local Python guru or system administrator. (E.g., /usr/local/python is a popular alternative location.)

On Windows machines where you have installed Python from the Microsoft Store, the python3.8 command will be available. If you have the py.exe launcher installed, you can use the py command. See Excursus: Setting environment variables for other ways to launch Python.

Typing an end-of-file character (Control-D on Unix, Control-Z on Windows) at the primary prompt causes the interpreter to exit with a zero-exit status. If that doesn’t work, you can exit the interpreter by typing the following command: quit ().

The interpreter’s line-editing features include interactive editing, history substitution and code completion on systems that support the GNU Readline library. Perhaps the quickest check to see whether command line editing is supported is typing Control-P to the first Python prompt you get. If it beeps, you have command line editing; see Appendix Interactive Input Editing and History Substitution for an introduction to the keys. If nothing appears to happen, or if ^P is echoed, command line editing isn’t available; you’ll only be able to use backspace to remove characters from the current line.

The interpreter operates somewhat like the Unix shell: when called with standard input connected to a tty device, it reads and executes commands interactively; when called with a file name argument or with a file as standard input, it reads and executes a script from that file.

A second way of starting the interpreter is python -c command [arg] ..., which executes the statement(s) in command, analogous to the shell’s -c option. Since Python statements often contain spaces or other characters that are special to the shell, it is usually advised to quote command in its entirety with single quotes. Some Python modules are also useful as scripts. These can be invoked using python -m module [arg] ..., which executes the source file for module as if you had spelled out its full name on the command line.

When a script file is used, it is sometimes useful to be able to run the script and enter interactive mode afterwards. This can be done by passing -i before the script.

### All command line options are described in Command line and environment.

When known to the interpreter, the script name and additional arguments thereafter are turned into a list of strings and assigned to the argv variable in the sys module. You can access this list by executing import sys. The length of the list is at least one; when no script and no arguments are given, sys.argv[0] is an empty string. When the script name is given as '-' (meaning standard input), sys.argv[0] is set to '-'. When -c command is used, sys.argv[0] is set to '-c'. When -m module is used, sys.argv[0] is set to the full name of the located module. Options found after -c command or -m module are not consumed by the Python interpreter’s option processing but left in sys.argv for the command or module to handle.

Interactive Mode

When commands are read from a tty, the interpreter is said to be in interactive mode. In this mode it prompts for the next command with the primary prompt, usually three greater-than signs (>>>); for continuation lines it prompts with the secondary prompt, by default three dots (...). The interpreter prints a welcome message stating its version number and a copyright notice before printing the first prompt:

$ python3.8

Python 3.8 (default, Sep 16 2015, 09:25:04) [GCC 4.8.2] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>>

>>>the\_world\_is\_flat = True

>>>ifthe\_world\_is\_flat:

... print ("Be careful not to fall off!")

...

Be careful not to fall off!

For more on interactive mode, see Interactive Mode.

## THE INTERPRETER AND ITS ENVIRONMENT

By default, Python source files are treated as encoded in UTF-8. In that encoding, characters of most languages in the world can be used simultaneously in string literals, identifiers and comments — although the standard library only uses ASCII characters for identifiers, a convention that any portable code should follow. To display all these characters properly, your editor must recognize that the file is UTF-8, and it must use a font that supports all the characters in the file.

To declare an encoding other than the default one, a special comment line should be added as the first line of the file. The syntax is as follows:

# -\*- coding: encoding -\*-

where encoding is one of the valid codecs supported by Python.

For example, to declare that Windows-1252 encoding is to be used, the first line of your source code file should be:

# -\*- coding: cp1252 -\*-

One exception to the first line rule is when the source code starts with a UNIX “shebang” line. In this case, the encoding declaration should be added as the second line of the file. For example:

#!/usr/bin/env python3 # -\*- coding: cp1252 -\*-

## INTRODUCTION TO ARTIFICIAL INTELLIGENCE

“The science and engineering of making intelligent machines, especially intelligent computer programs”. -John McCarthy-

Artificial Intelligence is an approach to make a computer, a robot, or a product to think how smart human think. AI is a study of how human brain think, learn, decide and work, when it tries to solve problems. And finally this study outputs intelligent software systems.The aim of AI is to improve computer functions which are related to human knowledge, for example, reasoning, learning, and problem-solving.

The intelligence is intangible. It is composed of

* Reasoning
* Learning
* Problem Solving
* Perception
* Linguistic Intelligence

The objectives of AI research are reasoning, knowledge representation, planning, learning, natural language processing, realization, and ability to move and manipulate objects. There are long-term goals in the general intelligence sector.

Approaches include statistical methods, computational intelligence, and traditional coding AI. During the AI research related to search and mathematical optimization, artificial neural networks and methods based on statistics, probability, and economics, we use many tools. Computer science attracts AI in the field of science, mathematics, psychology, linguistics, philosophy and so on.

## TRENDING AI ARTICLES:

1. [Cheat Sheets for AI, Neural Networks, Machine Learning, Deep Learning & Big Data](https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463)
2. [Data Science Simplified Part 1: Principles and Process](https://becominghuman.ai/data-science-simplified-principles-and-process-b06304d63308)
3. [Getting Started with Building Realtime API Infrastructure](https://becominghuman.ai/getting-started-with-building-realtime-api-infrastructure-a19601fc794e) [AI & NLP Workshop](https://becominghuman.ai/ai-nlp-workshop-7bc121986d61)

Applications of AI

* + Gaming − AI plays important role for machine to think of large number of possible positions based on deep knowledge in strategic games. for example, chess,river crossing, N-queens problems and etc.

Natural Language Processing − Interact with the computer that understands natural language spoken by humans.

* + Expert Systems − Machine or software provide explanation and advice to the users.
  + Vision Systems − Systems understand, explain, and describe visual input on the computer.
  + Speech Recognition − There are some AI based speech recognition systems have ability to hear and express as sentences and understand their meanings while a person talks to it. For example Siri and Google assistant.
  + Handwriting Recognition − The handwriting recognition software reads the text written on paper and recognize the shapes of the letters and convert it into editable text.
  + Intelligent Robots − Robots are able to perform the instructions given by a human. Major Goals
* Knowledge reasoning
* Planning
* Machine Learning
* Natural Language Processing
* Computer Vision
* Robotics

## IBM WATSON

IBM Watson is a suite of AI (Artificial Intelligence) services, tools, and technologies developed by IBM. It offers a wide range of capabilities, including natural language processing, machine learning, data analysis, computer vision, and speech recognition. Watson leverages advanced algorithms and cognitive computing to understand, reason, and learn from vast amounts of data.

Fig: 6.1

IBM Watson

“Watson” is an IBM supercomputer that combines Artificial Intelligence (AI) and complex inquisitive programming for ideal execution as a “question answering” machine. The supercomputer is named for IBM’s founder, Thomas J. Watson.

IBM Watson is at the forefront of the new era of computing. At the point when IBM Watson made, IBM communicated that “more than 100 particular techniques are used to inspect perceive sources, find and make theories, find and score affirm, and combination and rank speculations.” recently, the Watson limits have been expanded and the way by which Watson works has been changed to abuse new sending models (Watson on IBM Cloud) and propelled machine learning capacities and upgraded hardware open to architects and authorities. It isn’t any longer completely a request answering figuring system arranged from Q&A joins yet can now ‘see’, ‘hear’, ‘read’, ‘talk’, ‘taste’, ‘translate’, ‘learn’ and ‘endorse’.

## MACHINE LEARNING

* + - 1. **INTRODUCTION**

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people. Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision- making processes based on data inputs.

Any technology user today has benefitted from machine learning. Facial recognition technology allows social media platforms to help users tag and share photos of friends. Optical character recognition (OCR) technology converts images of text into movable type. Recommendation engines, powered by machine

learning, suggest what movies or television shows to watch next based on user preferences. Self-driving cars that rely on machine learning to navigate may soon be available to consumers.

Machine learning is a continuously developing field. Because of this, there are some considerations to keep in mind as you work with machine learning methodologies, or analyze the impact of machine learning processes. In this tutorial, we’ll look into the common machine learning methods of supervised and unsupervised learning, and common algorithmic approaches in machine learning, including the k-nearest neighbor algorithm, decision tree learning, and deep learning. We’ll explore which programming languages are most used in machine learning, providing you with some of the positive and negative attributes of each.

Additionally, we’ll discuss biases that are perpetuated by machine learning algorithms, and consider what can be kept in mind to prevent these biases when building algorithms.

## MACHINE LEARNING METHODS

In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed.

Two of the most widely adopted machine learning methods are supervised learning which trains algorithms based on example input and output data that is labeled by humans, and unsupervised learning which provides the algorithm with no labeled data in order to allow it to find structure within its input data. Let’s explore these methods in more detail.

## SUPERVISED LEARNING

In supervised learning, the computer is provided with example inputs that are labeled with their desired outputs. The purpose of this method is for the algorithm to be able to “learn” by comparing its actual output with the “taught” outputs to find errors, and modify the model accordingly. Supervised learning therefore uses patterns to predict label values on additional unlabeled data.

For example, with supervised learning, an algorithm may be fed data with images of sharks labeled as fish and images of oceans labeled as water. By being trained on this data, the supervised learning algorithm should be able to later identify unlabeled shark images as fish and unlabeled ocean images as water.

A common use case of supervised learning is to use historical data to predict statistically likely future events. It may use historical stock market information to anticipate upcoming fluctuations, or be employed to filter out spam emails. In supervised learning, tagged photos of dogs can be used as input data to classify untagged photos of dogs.

## UNSUPERVISED LEARNING

In unsupervised learning, data is unlabeled, so the learning algorithm is left to find commonalities among its input data. As unlabeled data are more abundant than labeled data, machine learning methods that facilitate unsupervised learning are particularly valuable.

The goal of unsupervised learning may be as straightforward as discovering hidden patterns within a dataset, but it may also have a goal of feature learning, which allows the computational machine to automatically discover the representations that are needed to classify raw data.

Unsupervised learning is commonly used for transactional data. You may have a large dataset of customers and their purchases, but as a human you will likely not be able to make sense of what similar attributes can be drawn from customer profiles and their types of purchases. With this data fed into an unsupervised learning algorithm, it may be determined that women of a certain age range who buy unscented soaps are likely to be pregnant, and therefore a marketing campaign related to pregnancy and baby products can be targeted to this audience in order to increase their number of purchases.

Without being told a “correct” answer, unsupervised learning methods can look at complex data that is more expansive and seemingly unrelated in order to organize it in potentially meaningful ways. Unsupervised learning is often used for anomaly detection including for fraudulent credit card purchases, and recommender systems that recommend what products to buy next. In unsupervised learning, untagged photos of dogs can be used as input data for the algorithm to find likenesses and classify dog photos together.

## APPROACHES

As a field, machine learning is closely related to computational statistics, so having a background knowledge in statistics is useful for understanding and leveraging machine learning algorithms.

For those who may not have studied statistics, it can be helpful to first define correlation and regression, as they are commonly used techniques for investigating the relationship among quantitative variables. Correlation is a measure of association between two variables that are not designated as either dependent or independent. Regression at a basic level is used to examine the relationship between one dependent and one independent variable. Because regression statistics can be used to anticipate the dependent variable when the independent variable is known, regression enables prediction capabilities.

Approaches to machine learning are continuously being developed. For our purposes, we’ll go through a few of the popular approaches that are being used in machine learning at the time of writing.

## REINFORCEMENT LEARNING

Reinforcement learning (RL) is a branch of machine learning that deals with the development of algorithms and models that enable an agent to learn and make decisions in an environment to maximize a cumulative reward. RL is inspired by how humans and animals learn from experience through trial and error.

In reinforcement learning, an agent interacts with an environment and learns from the feedback it receives in the form of rewards or punishments. The agent's goal is to learn the optimal actions or policies that maximize the long-term cumulative reward.

Here are the key components of reinforcement learning:

* + - * + Environment: The external system or problem that the agent interacts with and learns from. It provides feedback in the form of rewards or punishments.
        + State: The current representation of the environment at a given time, which the agent uses to make decisions.
        + Action: The specific choices or decisions made by the agent at each time step. The agent selects actions based on the observed state.
        + Reward: A scalar value that indicates the desirability or quality of the agent's actions. The agent's objective is to maximize the cumulative reward over time.
        + Policy: The strategy or rule that the agent follows to select actions based on the observed state. It maps states to actions.
        + Value Function: An estimate of the expected cumulative reward an agent can achieve from a given state or state-action pair. It is used to evaluate the quality of different states or actions.
        + Q-Learning: A popular algorithm used in reinforcement learning that employs the concept of a Q- value, which represents the expected cumulative reward of taking a particular action in a given state. Q-Learning updates the Q-values iteratively based on observed rewards.
        + Exploration vs. Exploitation: A trade-off in RL between exploring new actions or states to gain more information and exploiting the already learned knowledge to maximize rewards.

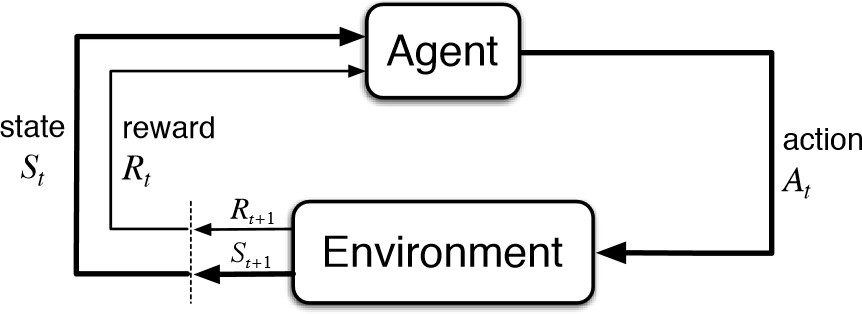


Fig: 6.2

Reinforcement Learning

## MARKOV DECISION

Markov Decision Processes (MDPs) are mathematical models used in the field of reinforcement learning to formalize decision-making problems in an uncertain environment. MDPs are an extension of Markov chains and incorporate the concept of decision-making and rewards.

In a Markov Decision Process, we have the following elements:

* + - * + States (S): A set of possible states in the environment. The system being modeled is assumed to be in one of these states at any given time.
        + Actions (A): A set of possible actions that the agent can take. The agent selects an action based on the current state.
        + Transition Probabilities (T): The probabilities associated with transitioning from one state to another when an action is taken. These probabilities define the dynamics of the system.
        + Rewards (R): The immediate rewards or penalties associated with transitioning from one state to another. Rewards can be positive, negative, or zero and provide feedback to the agent about the desirability of a particular state-action pair.
        + Policy (π): The strategy or rule that the agent follows to select actions based on the current state. It defines the agent's behavior in the MDP.

The dynamics of an MDP can be represented using the state transition probabilities, rewards, and the policy. The goal in an MDP is to find an optimal policy that maximizes the expected cumulative reward over time.

Several algorithms can be used to solve MDPs and find optimal policies, such as Value Iteration, Policy Iteration, and Q-Learning. These algorithms aim to estimate the value function (expected cumulative reward) for different states or state-action pairs and use it to guide the agent's decision-making process.

MDPs have applications in various fields, including robotics, finance, healthcare, and control systems, where decision-making in uncertain environments is required. They provide a principled framework for modeling and solving sequential decision-making problems and have been instrumental in advancing the field of reinforcement learning.

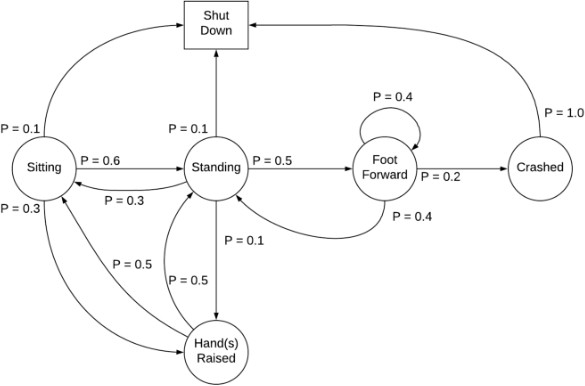


Fig: 6.3

Markov decision process

## A FORMAL DEFINITION OF DEEP LEARNING IS- NEURONS

Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

In human brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbors.

The question here is how we recreate these neurons in a computer. So, we create an artificial structure called an artificial neural net where we have nodes or neurons. We have some neurons for input value and some for-output value and in between, there may be lots of neurons interconnected in the hidden layer.

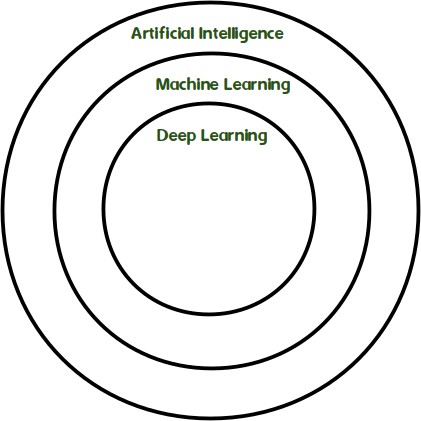


Figure 6.4 Layered architecture of AI

## PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An [interpreted language,](https://en.wikipedia.org/wiki/Interpreted_language) Python has a design philosophy that emphasizes code [readability](https://en.wikipedia.org/wiki/Readability) (notably using [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation to delimit [code blocks](https://en.wikipedia.org/wiki/Code_block) rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer [lines of code](https://en.wikipedia.org/wiki/Source_lines_of_code) than might be used in languages such as [C++](https://en.wikipedia.org/wiki/C%2B%2B)or [Java.](https://en.wikipedia.org/wiki/Java_(programming_language)) It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many [operating systems](https://en.wikipedia.org/wiki/Operating_system). [CPython](https://en.wikipedia.org/wiki/CPython), the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python, is [open source](https://en.wikipedia.org/wiki/Open_source) software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation). Python features a [dynamic type](https://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](https://en.wikipedia.org/wiki/Memory_management). It supports multiple [programming paradigms,](https://en.wikipedia.org/wiki/Programming_paradigm) including [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), [imperative,](https://en.wikipedia.org/wiki/Imperative_programming) [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural,](https://en.wikipedia.org/wiki/Procedural_programming) and has a large and comprehensive [standard](https://en.wikipedia.org/wiki/Standard_library) [library](https://en.wikipedia.org/wiki/Standard_library)

## DJANGO

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of [don't repeat](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself) [yourself.](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself) Python is used throughout, even for settings files and data models.

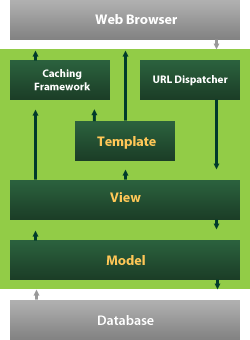


Fig: 6.5

Django architecture

Django also provides an optional administrative [create, read, update and delete](https://en.wikipedia.org/wiki/Create%2C_read%2C_update_and_delete) interface that is generated dynamically through [introspection](https://en.wikipedia.org/wiki/Introspection_(computer_science)) and configured via admin models

## CODING

from django.db.models import Count

from django.shortcuts import render, redirect, get\_object\_or\_404 # Create your views here.

from user.models import RegisterModel, UploadModel, RequestModel, FeedbackModel def login(request):

if request.method == "POST":

if request.method == "POST":

usid = request.POST.get('username') pswd = request.POST.get('password') if usid == 'admin' and pswd == 'admin':

return render(request,'admins/login.html') def admin\_page(request):

return render(request,'admins/admin\_page.html') def view\_userdetails(request):

obj = RegisterModel.objects.all()

return render(request,'admins/view\_userdetails.html',{'objects':obj}) def view\_uploadfile(request):

obj = UploadModel.objects.all()

return render(request,'admins/view\_uploadfile.html',{'object':obj}) def view\_user\_request(request):

obj=RequestModel.objects.all()

return render(request,'admins/view\_user\_request.html',{'obj':obj}) def admin\_update(request,pk):

obj = RequestModel.objects.get(id=pk) a=obj.accesstwo.id jki=UploadModel.objects.filter(id=a) if request.method == "POST":

add\_count = request.POST.get('add\_count', '') obj = get\_object\_or\_404(UploadModel, id=a) obj.add\_count = add\_count obj.save(update\_fields=["add\_count",]) return redirect('view\_user\_request')

return render(request,'admins/admin\_update.html',{'a':jki}) def admin\_graphicalanalysis(request):

chart = RequestModel.objects.values('cate').annotate(dcount=Count('cate')) return render(request,'admins/admin\_graphicalanalysis.html',{'objects':chart})

def view\_userquery(request):

obj = FeedbackModel.objects.all()

return render(request,'admins/view\_userquery.html',{'object':obj}) """Designing\_Cyber\_Insurance\_Policies URL Configuration

Class-based views

1. Add an import: from other\_app.views import Home
2. Add a URL to urlpatterns: url(r'^$', Home.as\_view(), name='home') Including another URLconf
3. Import the include() function: from django.conf.urls import url, include
4. Add a URL to urlpatterns: url(r'^blog/', include('blog.urls')) """

from django.conf.urls import url

from django.conf.urls.static import static from django.contrib import admin

from Designing\_Cyber\_Insurance\_Policies import settings from user import views as user\_views

from admins import views as admin\_views urlpatterns = [

url(r'^admin/', admin.site.urls), url('^$',user\_views.index,name="index"), url('user/register', user\_views.register, name="register"),

url('user\_page', user\_views.user\_page, name="user\_page"), url('upload\_fileview', user\_views.upload\_fileview, name="upload\_fileview"), url('view\_file', user\_views.view\_file, name="view\_file"), url('^user/otppage/(?P<pk>\d+)/$',user\_views.otppage,name="otppage"), url('download\_page',user\_views.download\_page,name="download\_page"), url('send\_feedback',user\_views.send\_feedback,name="send\_feedback"), url('mydetails',user\_views.mydetails,name="mydetails"), url('user/request/(?P<pk>\d+)/$',user\_views.request, name="request"),

WSGI config for Designing\_Cyber\_Insurance\_Policies project.

It exposes the WSGI callable as a module-level variable named ``application``. For more information on this file, see https://docs.djangoproject.com/en/1.11/howto/deployment/wsgi/

"""

import os

from django.core.wsgi import get\_wsgi\_application

application = get\_wsgi\_application() import os

# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)

from Designing\_Cyber\_Insurance\_Policies.emailsetting import SET\_EMAIL\_USE\_TLS, SET\_EMAIL\_HOST, SET\_EMAIL\_HOST\_USER, \

SET\_EMAIL\_HOST\_PASSWORD, SET\_EMAIL\_PORT, SET\_EMAIL\_BACKEND, SET\_DEFAULT\_FROM\_EMAIL

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath( file ))) # Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/1.11/howto/deployment/checklist/ # SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'goj924+523x(a01ws98%wt&cbbj+pd^-jl&fi!!cx2v@v\_rjc(' # SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True ALLOWED\_HOSTS = []

# Application definition INSTALLED\_APPS = [

'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles',

'django.middleware.security.SecurityMiddleware', 'django.contrib.sessions.middleware.SessionMiddleware', 'django.middleware.common.CommonMiddleware', 'django.middleware.csrf.CsrfViewMiddleware', 'django.contrib.auth.middleware.AuthenticationMiddleware', 'django.contrib.messages.middleware.MessageMiddleware', 'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'Designing\_Cyber\_Insurance\_Policies.urls' TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates', 'DIRS': [(os.path.join(BASE\_DIR,'assets/templates'))], 'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [ 'django.template.context\_processors.debug', 'django.template.context\_processors.request', 'django.contrib.auth.context\_processors.auth', 'django.contrib.messages.context\_processors.messages',

],

},

},

]

WSGI\_APPLICATION = 'Designing\_Cyber\_Insurance\_Policies.wsgi.application' # Database

# https://docs.djangoproject.com/en/1.11/ref/settings/#databases DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql', 'NAME': 'cyber\_insurance',

'USER': 'root',

'PASSWORD': '', 'HOST': '127.0.0.1',

'PORT': '3306',

}

}

#Password validation

# https://docs.djangoproject.com/en/1.11/ref/settings/#auth-password-validators AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

# Internationalization

# https://docs.djangoproject.com/en/1.11/topics/i18n/ LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True USE\_L10N = True USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/1.11/howto/static-files/ STATIC\_URL = '/static/' STATICFILES\_DIRS=[os.path.join(BASE\_DIR,'assets/static')] MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'assets/media') #EMAIL SETTING

EMAIL\_USE\_TLS = SET\_EMAIL\_USE\_TLS EMAIL\_HOST = SET\_EMAIL\_HOST EMAIL\_HOST\_USER = SET\_EMAIL\_HOST\_USER

EMAIL\_HOST\_PASSWORD = SET\_EMAIL\_HOST\_PASSWORD EMAIL\_PORT = SET\_EMAIL\_PORT

EMAIL\_BACKEND = SET\_EMAIL\_BACKEND DEFAULT\_FROM\_EMAIL = SET\_DEFAULT\_FROM\_EMAIL

## OUTPUT SCREENS

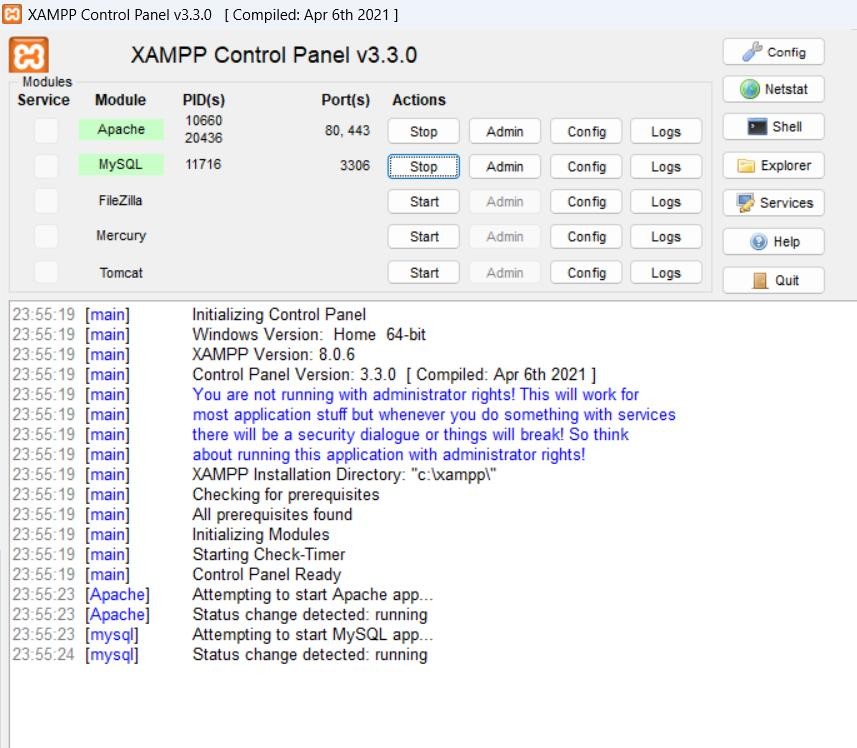


Fig: 6.6 Running the code

The above figure illustrates how we run the code. We need to install XAMPP control panel and start Apache and MySQL module. We usually use command -> python main.py run server which invokes python interpreter. On Windows machines where you have installed Python from the [Microsoft Store](https://docs.python.org/3/using/windows.html#windows-store), the python3.11 command will be available. If you have the [py.exe launcher](https://docs.python.org/3/using/windows.html#launcher) installed, you can use the py command. See [Excursus: Setting environment variables](https://docs.python.org/3/using/windows.html#setting-envvars) for other ways to launch Python.

XAMPP (pronounced "cross-amp") is a popular software package that provides an easy way to install and configure a local server environment on your computer. It stands for "X" (cross-platform), Apache, MySQL, PHP, and Perl, which are the components included in the package.

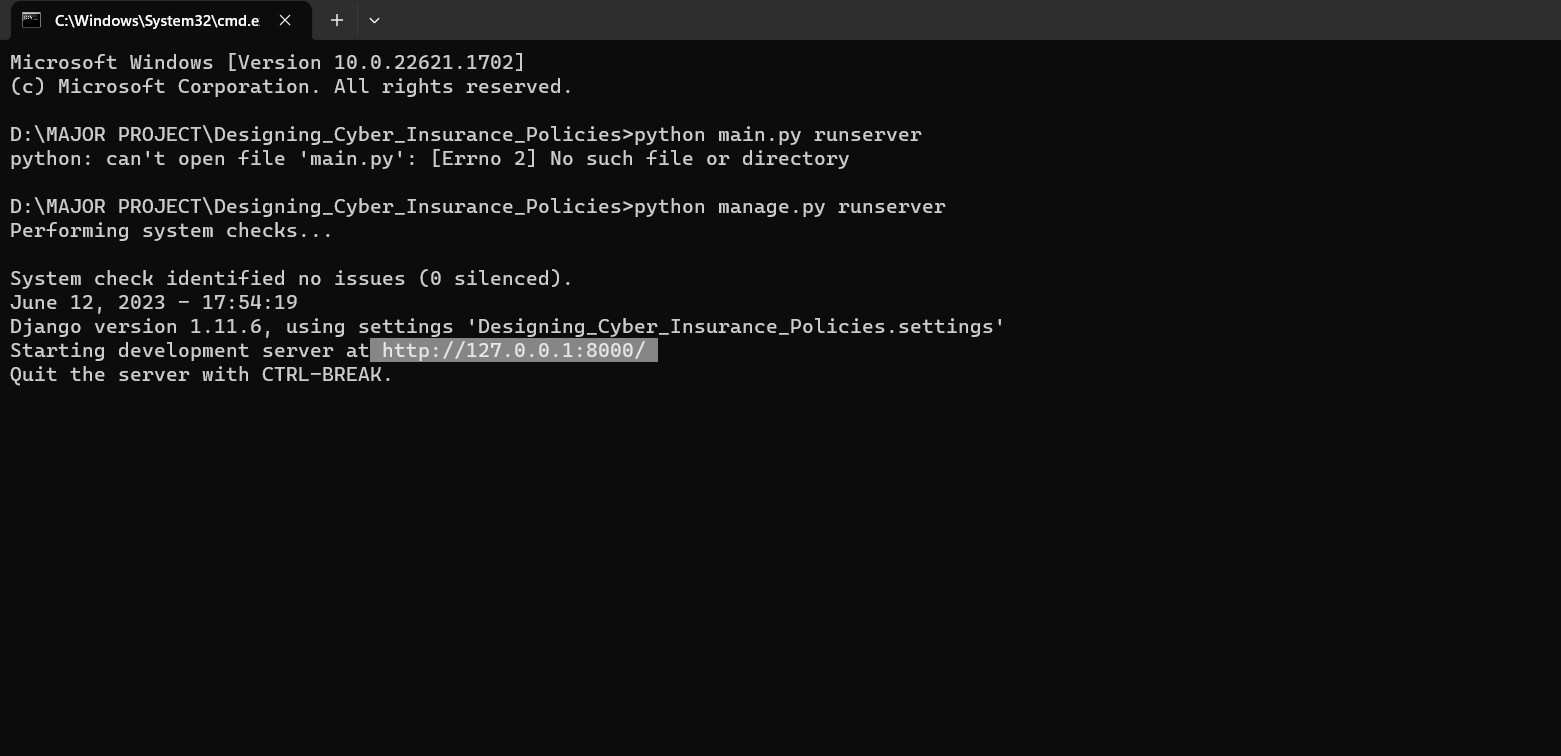


Fig 6.7 execution

To run the Django development server, type the command python manage.py run server in your terminal of the Django project and hit enter. If everything is okay with your project, Django will start running the server at localhost port 8000 (127.0.0.1:8000) and then you have to navigate to that link in your browser. While the server is running, Django will print out in real-time all the actions that are taking place



Fig 6.8

An input screen will be displayed.

The page asks you to sign in if you already have an account or sign up if you are new. Once logged in you can gain access to the facility to store your documents on the given server. The login portal of our cyber insurance website serves as a gateway for policyholders and authorized users to access their accounts and manage their cyber insurance policies. With a focus on security and convenience, our login portal offers a seamless and user-friendly experience.

Upon accessing the login portal, users are presented with a clean and intuitive interface. The portal incorporates modern design elements that prioritize simplicity and ease of use, ensuring that users can navigate the login process effortlessly. The login page features a prominently displayed login form, inviting users to input their credentials to proceed.

To enhance security, our login portal employs robust authentication measures. Users are required to enter their unique username or email address, followed by a strong and confidential password. To further fortify account protection, we implement additional security layers such as two-factor authentication (2FA). This multi-step verification process adds an extra level of security by requiring users to confirm their identity through a secondary method, such as a verification code sent to their registered mobile device.



Fig 6.9

cross check your personal details

In the above screen we can check our personal details which we have provided during signing up such as name, email address, mobile number etc.

On our cyber insurance website, you have the ability to easily access and review your personal details. By logging into your account, you can view and manage your personal information securely. We understand the importance of maintaining accurate and up-to-date details, so our platform provides a user-friendly interface for checking and editing your personal information. Simply log in to your account, navigate to the relevant section, and review or update your personal details as needed. We prioritize the security and privacy of your information, implementing industry-standard encryption protocols and robust security measures to ensure the confidentiality of your personal data. If you require any assistance or have questions regarding your personal details, our dedicated support team is available to provide guidance and support.



Fig: 6.10

uploading the documents

In above screen we enter the documents that we need to upload to the server. We can even name the uploaded document and even provide it with a location.

Our cyber insurance website offers a convenient feature that allows policyholders to securely upload their documents. We understand the importance of providing a streamlined process for submitting and managing necessary paperwork. By accessing your account on our platform, you can easily upload your documents to ensure a smooth and efficient claims or policy management experience. We prioritize the security and confidentiality of your information, employing industry-standard encryption and data protection measures to safeguard your documents during transmission and storage.

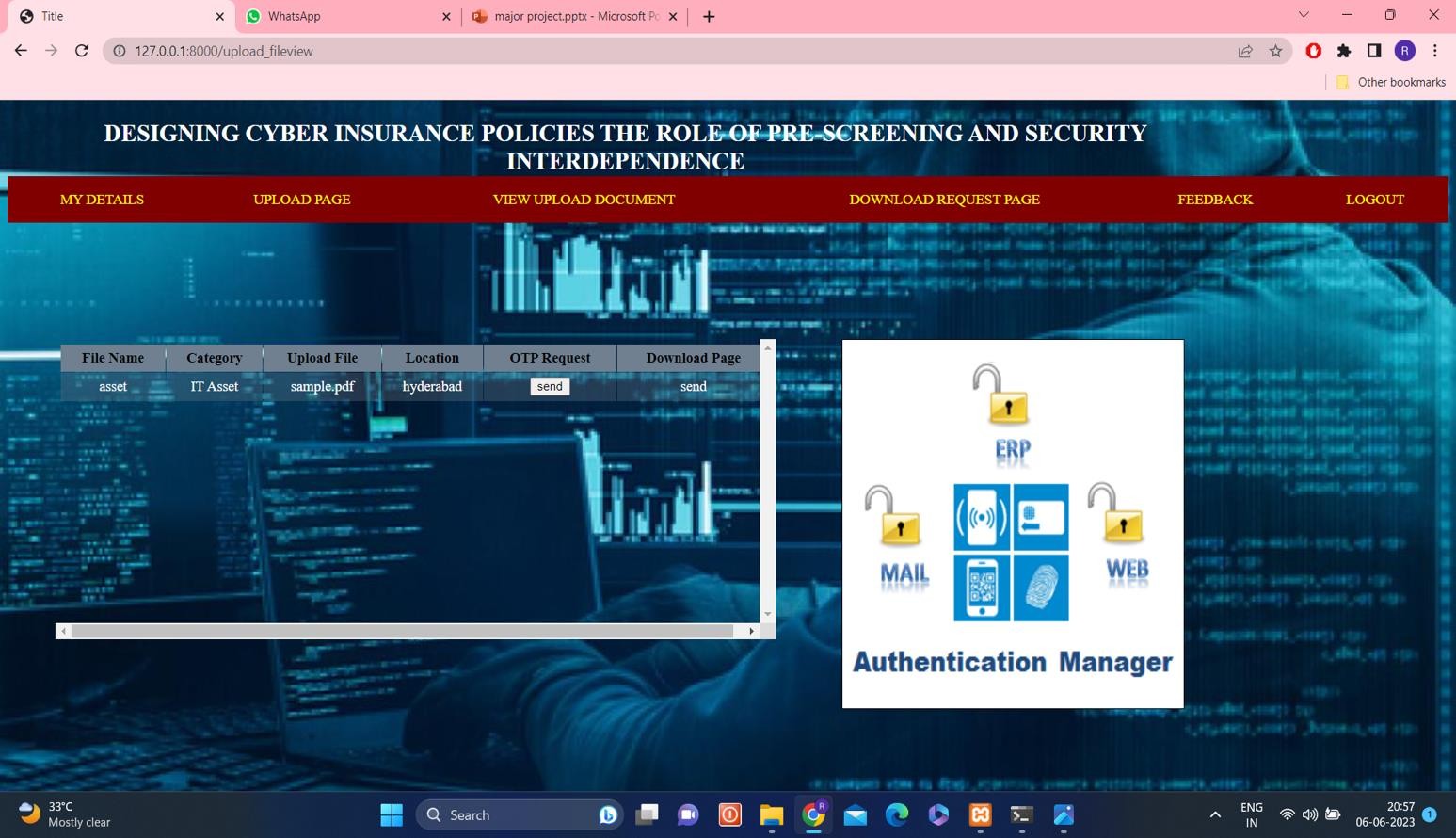


Fig 6.11

Accessing the documents

To access the document, we select the view upload document tab and click on the send button that lies below the OTP request section. Once we do that an email consisting of the OTP is sent to the user via the admin mail automatically.

On our cyber insurance website, you have the capability to conveniently view and access your documents. We understand the importance of having quick and easy access to your policy-related paperwork, and our platform is designed to provide a seamless document viewing experience. By logging into your account and navigating to the appropriate section, you can effortlessly access a comprehensive list of your documents. Our user-friendly interface ensures that you can easily browse, search, and open your documents with just a few clicks. We prioritize the security and privacy of your documents, implementing robust data protection measures and encryption protocols to ensure the confidentiality and integrity of your information.

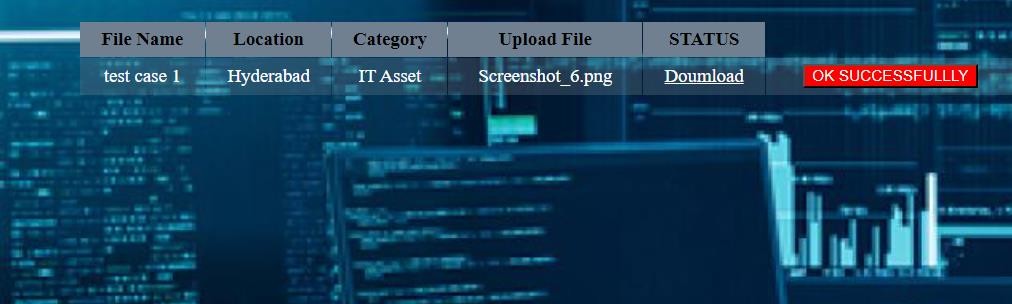


Fig 6.12

Extracting the documents

After entering the OTP, the document is ready to be downloaded from the download request page. When documenting a software system or application, it is common to include screenshots or output screens to provide visual representations of the user interface or the expected results of certain operations. Including screenshots in documentation can help users better understand the system's functionality and how to interact with it effectively.

Here are a few ways to incorporate screenshots or output screens in documentation:

1. Capture Screenshots: Take screenshots of the relevant screens or outputs from the system. This can be done using built-in operating system tools or specialized screen capture software.
2. Label Screenshots: Provide clear labels or captions for each screenshot to indicate the purpose or context of the captured screen. It helps users easily identify what they are looking at.
3. Include Annotations: Add annotations or callouts to highlight specific elements or provide additional explanations within the screenshot itself.

# SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

## TYPES OF TESTS

* + 1. **UNIT TESTING**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

## INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

## FUNCTIONAL TEST

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

* + - * Valid Input: identified classes of valid input must be accepted.
      * Invalid Input: identified classes of invalid input must be rejected.
      * Functions: identified functions must be exercised.
      * Output: identified classes of application outputs must be exercised.
      * Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing.

## SYSTEM TEST

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre- driven process links and integration points.

## WHITE BOX TESTING

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

## BLACK BOX TESTING

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

## TEST STRATEGY AND APPROACH

Field testing will be performed manually and functional tests will be written in detail.

## TEST OBJECTIVES

* + - * All field entries must work properly.
      * Pages must be activated from the identified link.
      * The entry screen, messages and responses must not be delayed.

## FEATURES TO BE TESTED

* + - * Verify that the entries are of the correct format
      * No duplicate entries should be allowed
      * All links should take the user to the correct page.

## INTEGRATION TESTING

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g., components in a software system or – one step up – software applications at the company level – interact without error.

**TEST RESULTS:** All the test cases mentioned above passed successfully. No defects encountered.

## ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements. Acceptance testing is a crucial phase in software development that focuses on evaluating the system's compliance with business requirements and ensuring that it meets the needs of end-users or stakeholders. It involves executing test cases or scenarios to validate the software's functionality, usability, reliability, and performance, among other aspects.

During acceptance testing, the software is tested in a realistic environment that closely resembles the production environment. The goal is to determine if the software meets the acceptance criteria defined by the stakeholders and if it is ready for deployment. This type of testing is typically performed by end-users, clients, or independent testers who are representative of the target audience.

Acceptance testing can be broadly categorized into two types:

1. User Acceptance Testing (UAT): User Acceptance Testing involves testing the software from the end-user's perspective. Users or business representatives perform tests to verify if the software meets their specific requirements, business processes, and workflow. UAT focuses on validating the overall usability, functionality, and suitability of the software for the intended users.
2. Operational Acceptance Testing: Operational Acceptance Testing focuses on assessing the software's ability to function in the operational or production environment. It involves testing non-functional aspects such as performance, security, reliability, and scalability. The objective is to ensure that the software is capable of performing optimally and meets the operational requirements of the organization.

The goal is to determine if the software meets the acceptance criteria defined by the stakeholders and if it is ready for deployment. This type of testing is typically performed by end-users, clients, or independent testers who are representative of the target audience. Acceptance testing is a crucial phase in software development that focuses on verifying whether a system meets the requirements and specifications outlined by the stakeholders. It is performed to ensure that the software is ready for deployment and meets the needs and expectations of the end users.

**TEST RESULTS:** All the test cases mentioned above passed successfully. No defects encountered.

1. **CONCLUSION**

## CONCLUSION

We studied the problem of designing cyber insurance contracts by a single profit-maximizing insurer, for both risk-neutral and risk-averse agents. While the introduction of insurance worsens network security in a network of independent agents, we showed that the result could be different in a network of interdependent agents. Specifically, we showed that security interdependency leads to a profit opportunity for the insurer, created by the inefficient effort levels exerted by free-riding agents when insurance is not available but interdependency is present, this is in addition to risk transfer that an insurer typically profits from. We showed that security prescreening then allows the insurer to take advantage of this additional profit opportunity by designing the right contracts to incentivize the agents to increase their effort levels and essentially selling commitment to interdependent agents. We show under what conditions this type of contracts leads to not only increased profit for the principal and utility for the agents, but also improved state of network security.

## FUTURE SCOPE

In conclusion, cyber insurance plays a vital role in managing and mitigating the risks associated with cyber threats and attacks. It provides financial protection and support to organizations in the event of a data breach, cyberattack, or other cyber incidents. By obtaining cyber insurance, businesses can transfer some of the financial burdens and liabilities that may arise from cyber incidents to insurance providers.

Cyber insurance policies typically cover various aspects, including data breach response and investigation costs, legal and regulatory expenses, business interruption losses, ransomware payments, and third-party liabilities. The coverage can vary depending on the specific policy and the needs of the organization.

Having cyber insurance encourages organizations to invest in robust cybersecurity measures, as it promotes risk awareness and risk management practices. It also provides assurance to customers, partners, and stakeholders that the organization takes cybersecurity seriously and has measures in place to address potential incidents.

In the future, the scope of designing cyber insurance policies using pre-screening and interdependence can be expanded to further enhance the effectiveness and coverage of policies. Here are some potential future developments:

1. **ADVANCED RISK ASSESSMENT:** a. Cyber Risk Modeling: Utilize advanced data analytics and machine learning techniques to develop predictive models that assess cyber risks more accurately. These models can incorporate a wide range of data sources, such as threat intelligence feeds, historical breach data, and vulnerability assessments, to provide a comprehensive risk profile.

posture. This can involve automated scans of their networks, systems, and applications, as well as analysis of threat intelligence data to identify emerging risks.

1. **CYBERSECURITY COLLABORATION:** a. Cybersecurity Partnerships: Foster partnerships with cybersecurity service providers to offer policyholders access to discounted or specialized security services. This collaboration can help policyholders improve their security posture, leading to reduced risks and potential premium discounts. b. Incident Response Support: Establish partnerships with incident response firms to provide policyholders with immediate assistance in case of a cyber incident. This can include rapid response teams, forensics analysis, and legal support to help mitigate the impact of an attack.
2. **INTERDEPENDENCE CONSIDERATIONS**: a. Supply Chain Risk Management: Recognize the interdependencies within supply chains and extend coverage to include potential risks originating from third-party vendors and suppliers. This can involve evaluating the security practices of vendors and mandating certain security standards as a condition for coverage. b. Cloud Service Provider (CSP) Coverage: Develop specific policies tailored to cover risks associated with cloud service usage. This can address potential data breaches, service interruptions, and liabilities arising from the use of cloud infrastructure.
3. **DYNAMIC POLICY ADJUSTMENTS:** a. Flexible Coverage: Explore the possibility of dynamic policy adjustments based on changes in the policyholder's cyber risk profile. This can involve premium adjustments or coverage modifications to align with evolving cybersecurity needs. b. Cybersecurity Incentives: Offer incentives, such as premium reductions or additional coverage benefits, to policyholders who actively invest in improving their cybersecurity posture. This approach encourages continuous risk mitigation and rewards responsible cybersecurity practices.
4. **CYBER INSURANCE STANDARDS AND REGULATION:** a. Standardization:

Collaborate with industry stakeholders to develop standardized frameworks for cyber insurance policies, covering risk assessment methodologies, coverage terms, and claims processes. This can enhance transparency, comparability, and overall policy effectiveness. b. Regulatory Frameworks: Work closely with regulatory bodies to establish guidelines and regulations around cyber insurance, ensuring fair and consistent practices across the industry. This can promote consumer protection and provide a clear legal framework for policyholders and insurers.

As the cybersecurity landscape evolves, leveraging pre-screening and interdependence considerations will continue to be crucial for designing effective cyber insurance policies that align with emerging risks and technological advancements.

## DISADVANTAGES:

**LIMITED AVAILABILITY**: Prescreening and security interdependence may result in limited availability of cyber insurance policies. Insurers may choose to provide coverage only to organizations that meet specific security criteria or have a predetermined level of cybersecurity measures in place. This can make it difficult for organizations with less mature security practices to obtain coverage.

**HIGHER PREMIUMS**: Policies designed with prescreening and security interdependence may come with higher premiums. Insurers may charge higher rates for organizations that do not meet the strict security requirements or have dependencies on interconnected systems. This can place a financial burden on businesses, especially smaller organizations with limited resources.

**COMPLEXITY AND ADMINISTRATIVE BURDEN**: Implementing prescreening and security interdependence requires detailed assessments and ongoing monitoring of policyholders' security practices. This can result in increased complexity and administrative burden for both insurers and policyholders. Insurers may need to invest in additional resources and expertise to conduct thorough security evaluations, while policyholders may face additional reporting and compliance requirements.

**POTENTIAL COVERAGE GAPS**: Depending on the specific prescreening criteria and security interdependencies considered, there is a risk of coverage gaps. If a policyholder experiences a cyber incident that falls outside the predefined scope or dependencies, the policy may not provide adequate coverage. This can leave organizations exposed to financial losses that are not covered by their insurance policy.

**INFLEXIBILITY FOR CHANGING SECURITY LANDSCAPE**: Prescreening and security interdependence rely on specific security criteria and assumptions about interconnected systems. However, the cybersecurity landscape is constantly evolving. New threats and vulnerabilities emerge, and security best practices evolve over time. Policies based on static criteria may struggle to adapt to these changes, potentially leaving organizations underinsured or with outdated coverage.

It's essential to carefully consider these disadvantages and strike a balance between risk mitigation, affordability, and the dynamic nature of cybersecurity when designing cyber insurance policies using prescreening and security interdependence.

1. **REFERENCES**
2. M. M. Khalili, P. Naghizadeh, and M. Liu, “Designing cyber insurance policies: Mitigating moral hazard through security prescreening,” in The 7th International EAI Conference on Game Theory fo Networks (Gamenets), 2017.
3. M. M. Khalili, P. Naghizadeh, and M. Liu, “Designing cyber insurance policies in the presence of security interdependence,” in The 12th Workshop on the Economics of Networks, Systems and Computation (NetEcon), 2017.
4. C. Hemenwa, “ABI Research: Cyber insurance market to reach $10B by 2020,” [www.advisenltd.com/2015/07/30/abi-researchcyber-insurance-market-to-reach-10b-by-2020/,](http://www.advisenltd.com/2015/07/30/abi-researchcyber-insurance-market-to-reach-10b-by-2020/) 2015.
5. Insurance Information Institute, “U.S. cyber insurance market demonstrates growth, innovation in wake of high profile data breaches,” [www.iii.org/pressrelease/us-cyber-insurance-marketdemonstrates-](http://www.iii.org/pressrelease/us-cyber-insurance-marketdemonstrates-) growthinnovation-in-wake-of-high-profile-databreaches-102015, 2015.
6. N. Shetty, G. Schwartz, and J. Walrand, “Can competitive insurers improve network security?,” in The Third International Conference on Trust and Trustworthy Computing (TRUST), 2010.
7. N. Shetty, G. Schwartz, M. Felegyhazi, and J. Walrand, “Competitive cyber-insurance and internet security,” Economics of Information Security and Privacy, pp. 229–247, 2010.
8. G. Schwartz, N. Shetty, and J. Walrand, “Cyber-insurance: Missing market driven by user heterogeneity,” 2010.
9. G. A. Schwartz and S. S. Sastry, “Cyber-insurance framework for large scale interdependent networks,” in The Third International Conference on High Confidence Networked Systems, 2014.
10. H. Ogut, N. Menon, and S. Raghunathan, “Cyber insurance and it security investment: Impact of interdependence risk,” in The Workshop on the Economics of Information Security, 2005.
11. Z. Yang and J. C. S. Lui, “Security adoption and influence of cyber-insurance markets in heterogeneous networks,” Performance Evaluation, vol. 74, pp. 1–17, Apr. 2014.