Transcript:

[Slide 1: Introduction -> Words: 170 Minutes:1:40]

Hello and welcome, this is our team presentation about the practical development of intelligent agent for the purpose of file classification.

In the current world, computerised devices and digital data are integral to our everyday lives. This has led to significant challenges for businesses and individuals.

The first important challenge is organising and categorising vast quantities of files generated. This is to ensure efficient management and accessibility. Secondly, classifying suspicious files is crucial for identifying and mitigating security threats, preventing malware propagation and protecting both data and systems.

Not all suspicious files are necessarily malicious. But some files may trigger suspicion due to false positives or legitimate activities that may appear unusual. Proper classification helps differentiate between genuine threats and benign files, reducing the chances of false alarms and unnecessary disruptions.

This project aims to automate file classification using intelligent agents, by using the speed and efficiency of intelligent agents. The project intends to simplify the identification of ordinary and potentially dangerous file types, thereby enhancing file management and security.

[Slide 2:Background -> Words: 209 Minutes:1:52]

Now let’s look at the context of our project. An intelligent agent is basically a software program based system that has the ability to perceive its environment through sensors or input data, and use this to understand the environment, employ reasoning mechanisms to process and analyse the information they perceive, use

algorithms, logic or machine learning techniques to extract knowledge, make inferences and derive conclusions based on available data. Based on the reasoning process they can make autonomous decisions or provide recommendations

In our current work, we have utilised intelligent agent to perceive the system environment that awaits for user to input files, used rule based system to define normal and suspicious files. Here we have used reactive agent architecture. The main reason we used it is the requirement of real-time response for the user file input and complex reasoning was not required.

Agents communicate by communication protocols, languages, and frameworks to exchange information and coordinate actions in multiagent systems.

Agents learn using knowledge base to acquire knowledge, learn from experience, and adapt their behavior based on changing circumstances.

Multiple agents work together on specific tasks and cooperate, negotiate, and collaborate to achieve shared goals or solve complex problems. Agents can communicate with user as well to exchange information

[Slide 3:Objectives-> Words: 160; Minutes:1:26]

We formulated SMART objectives, so that our project team can have clear, specific and measurable targets that are achievable and time-bound, they provide guidance and focus throughout the project's development, testing, and validation phases

Our main objectives were to develop intelligent agents capable of accurately analysing file name extensions for classification of at least 10 different file types with an accuracy rate of 95% or higher, within a timeline of 10 days.

Also to create a user-friendly interface that allows users to review and validate the classification results, to provide feedback on any misclassifications and to complete this process within a timeframe of 10 days.

And finally to conduct rigorous testing and validation processes on the intelligent agents, including a dataset of 1,000 files representing a diverse range of file types, to

achieve an accuracy rate of 90% or higher and ensure that the classification process takes no more than 5 seconds per file, within a timeline of 5 days.

[Slide 4: Methodology -> Words: 236, Minutes: 2:02]

In this presentation, we will explore the benefits of utilising a multi-agent system and our rationale for choosing reactive agents in our file classification project. By establishing coordination and interaction among multiple agents, we leverage their distinct capabilities and knowledge to effectively tackle complex tasks. This approach enhances the efficiency of file classification, achieving accurate results through collaboration.

A notable study by Peng et al. (2001) compared single and multi-agent systems in file classification, evaluating criteria such as response time, classification quality, and economic and privacy considerations. The findings demonstrated that the collaborative multi-agent system outperformed the single-agent system in all aspects. These results highlight the promising potential of adopting a multi-agent system for file classification, enabling superior performance through agent collaboration.

Our deliberate decision to employ reactive agents further enhances our system's capabilities. These agents react and respond based on immediate input, ensuring speed and efficiency in file perception and classification. By incorporating reactive agents, we achieve dynamic and real-time responses to the classification requirements of encountered files. This choice aligns with our objective of delivering a fast, efficient system that adapts to changing classification needs.

To summarise, utilising a multi-agent system facilitates collaboration and interaction among agents, effectively addressing complex tasks in file classification. The comparative study by Peng et al. emphasises the superior performance of a collaborative multi-agent system. Additionally, employing reactive agents ensures a dynamic and real-time response to file classification needs.

[Slide 5: Project Design -> Words: 197, Minutes: 1:46]

The project's design is visually represented through two UML diagrams. The first diagram is the UML use case diagram, which illustrates the broad functionalities and interactions between the system and its actors. It shows the actors involved, such as the User, UploadAgent, ClassifierAgent, MoveFileAgent, and ViewFileAgent, along with their associated use cases. The relationships between the actors and use cases demonstrate their involvement in actions like file uploading, file viewing, receiving files for classification, passing files for classification, classifying files, moving safe and suspicious files to respective folders, and displaying safe and suspicious files.

The second diagram is the Sequence UML diagram, which highlights the sequential interactions between the actors involved in the use case. It shows the User initiating the process by uploading files, followed by interactions between the User, UploadAgent, ClassifierAgent, MoveFileAgent, and ViewFileAgent. The diagram provides a visual representation of the sequence of steps involved in the system, including receiving files for classification, passing files for classification, classifying files, moving files to appropriate folders, and displaying the files. These diagrams offer a comprehensive view of the system's key functionalities, interactions, and overall flow of actions, providing valuable insights into the project's design and functionality.

[Slide 6: Development -> Words: 205, Minutes: 1:57]

In this slide, we will focus on the Development stage of our app. Looking at the code on this slide, we start with the UploadAgent, which is like a secure mailbox. This piece of code allows users to upload files through a simple interface. Once a file is received, it's saved in a separate 'uploads' directory. Then, the ClassifierAgent steps in.

The ClassifierAgent is our detective, examining the uploaded files. It inspects file extensions to identify any that might be suspicious. If the file extension matches our list of suspicious file types, the file is marked as suspicious. Otherwise, it's considered safe.

Once files are classified, MoveFileAgent gets to work. It's the librarian in our system. This code moves files to either the 'Safe' or 'Suspicious' folders based on their classification. This way, our file library stays well-organised.

Finally, we have the ViewFileAgent. This is our display window. It shows you what's inside the 'Safe' and 'Suspicious' folders. With this code, you can see the list of all your files and their classifications.

In summary, each piece of code you see here represents an agent in our system. They work together to ensure files are uploaded, classified, organised, and viewable, providing a secure and user-friendly experience.

[Slide 7: Testing and Evaluation -> Words: 365, Minutes: 3:54]

This slide focuses on the Testing and Evaluation stages of our File Classifier application. The objective here is to ensure the application's functionality, identify potential issues, evaluate performance, and validate user satisfaction.

We have employed a variety of testing types to achieve these objectives. Starting with Unit Testing, we rigorously checked individual functions for expected outcomes using Python's unit test library.

Integration Testing followed, revealing any faults that might occur when these units interact with each other.

Then, we moved to System Testing, examining the application as a whole, and confirming that all parts coalesce into a compatible, smoothly functioning system.

Performance testing was also crucial, analysing the behaviour of the application under heavy load. This assessment ensures our application remains robust and reliable even when processing a large number of files.

Finally, we conducted User Acceptance Testing, or UAT, which played a key role in confirming user satisfaction and validating that the application effectively meets user needs.

For conducting these tests and measuring their coverage, we utilised Python's unit test, pytest-cov, and coverage libraries.

The screenshot labeled Figure 7.1, showcases the results of our unit tests. With a coverage of 89%, a majority of our code is tested and verified to function as expected.

Now, let's talk about Figure 7.2, our system process flow diagram. It starts with the Upload File stage. From there, the system determines the file type. If a file is deemed suspicious, it is moved to a suspicious folder. Safe files, on the other hand, are relocated to a safe folder. This flow diagram encapsulates the logic behind our application.

Turning to our User Interfaces, let's look at Figure 7.3 - the File Upload UI. This is the entry point for users, where they can upload multiple files for classification.

Next, we have Figure 7.4, the File Browser UI. Here, users can navigate through their file system, allowing for easy selection of files.

Finally, Figure 7.5 showcases the Result UI, where files are listed as either safe or suspicious, providing users with a clear understanding of the file classification results.

Through these robust testing methods, we have ensured functionality, performance, and user satisfaction with our application.

[Slide 8:Programming Approach -> Words: 274, Minutes: 3:13]

This slide provides a glimpse into our Programming Approach while developing the File Classifier application.

Let's begin by looking at our main server file, app.py, shown in the snippet on this slide. We've chosen Python as our programming language due to its readability, simplicity, and vast library support.

In this file, we make use of the Flask framework, which is lightweight and perfect for our web interface needs. We've also employed Werkzeug, a comprehensive WSGI web application library. It offers various utilities to ensure secure and efficient file handling and communication between our server and the clients.

We also utilised built-in Python modules such as os and shutil. The 'os' module handles operating system-dependent functionalities, such as path manipulation, whereas 'shutil' is responsible for high-level file operations like moving our files to the 'safe' or 'suspicious' folders.

Moving on to our front-end, we have the upload.html file. This HTML template forms the File Upload Page, a user-friendly interface where users can select and upload their files. It also includes a small JavaScript snippet to enhance user experience by enabling or disabling the 'Upload' button based on whether files are selected.

Finally, let's look at the view.html file. This HTML template acts as the File Viewing Page, where users can see the results of our file classifier. It lists files categorised as "safe" or "suspicious", providing a transparent overview of the classification results. We've utilised Flask's template syntax here to dynamically populate the file list based on the results from our server.

Together, these components form the basis of our programming approach, balancing user-friendly front-end interfaces with a secure, efficient back-end server.

[Slide 9: Challenges -> Words: 205, Minutes: 1:37]

One of the critical challenges we anticipate in the development process revolves around ensuring the precise classification of files. The conventional method of relying solely on file extensions can be misleading, particularly if a malicious file is disguised with a benign extension (Casey, 2011). Therefore, it becomes crucial for us to implement a robust mechanism that goes beyond file extensions to accurately identify and classify files.

Another critical concern is ensuring secure file uploads to mitigate the risk of directory traversal attacks. This type of attack occurs when an attacker manipulates file paths to access unauthorised directories and potentially compromise the system's security. To address this concern, we need to implement effective input validation techniques and conduct regular security testing (Stuttard & Pinto, 2011). By doing so, we can minimise the vulnerability to directory traversal attacks and safeguard the integrity and security of our file classification system.

In summary, guaranteeing accurate file classification and securing file uploads are paramount challenges in our development process. We need to overcome the limitations of file extensions by adopting a more comprehensive approach to classification. Additionally, implementing robust input validation and conducting regular security testing will fortify our system against directory traversal attacks and bolster its overall security posture.

[Slide 10: Conclusion Words: 124, Minutes: 1:11]

In conclusion, we successfully developed multiple agents to classify files based on their file name extension. User-friendly interface was created to review the classification results The performance of the agents were successfully tested and validated.

The potential drawbacks of using only filename extension as a basis for file classification were discussed. In future, this project could be further extended to classify files based on their content or other user defined attributes and could use machine learning methods and natural language processing to enhance agent performance.

Overall, our project with a multiagent system for file classification couples the power of multiple autonomous agents to achieve efficient and accurate file organisation. By utilising such systems can improve file management, retrieval, and decision-making processes in diverse domains

References

Bernardi, S., Donatelli, S. and Merseguer, J. (2002) 'From UML sequence diagrams and statecharts to analysable Petri Net Models', *Proceedings of the 3rd international workshop on Software and performance* [Preprint]. doi:10.1145/584369.584376.

IBM (no date) *Use-case diagrams*, *in UML modeling*. Available at: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case (Accessed: 02 July 2023).

Peng, S., Mukhopadhyay, S., Raje, R. & Palakal, M. (2001) A Comparison Between Single-agent and Multi-agent Classification of Documents. *Proceedings of the 10th Heterogeneous Computing Workshop* 2; 20090.2.

Wooldridge, M.J. (2009) An Introduction to Multiagent Systems. Chichester: John Wiley & Sons.