DECISION MAKING ON A SOFTWARE UPGRADE OR DECOMMISSION WITH DATA MINING AND MACHINE LEARNING TECHNIQUES

IN INFORMATION TECHNOLOGY INDUSTRY

BY

RAVIKANTH KOWDEED

DISSERTATION

Presented to the Swiss School of Business and Management Geneva

In fulfillment of the requirements

for the Degree

GLOBAL DOCTOR OF BUSINESS ADMINISTRATION

December 2024

Approved by

[illegible]

Prof.dr.sc. Saša Petar, Ph.D., dissertation Chair

Received/Approved By:

____

Admissions Director

Dedication

I dedicate this idea, proposal and research work to my family who supported me in this journey.

Acknowledgements

I acknowledge and thank my mentor Dr Monika Singh for valuable review inputs and suggestions in my entire journey of this research work. As part of this research, I tried to come up with a process and framework for the complex problem of decision making on software upgrades and decommissions in terms of providing a decision tree, defining and integrating the input data sets, providing a prototype solution to derive the recommendations.

In this process, I have also filed a patent application for the proposal of recommendation tool that performs the above-mentioned functionality. I dedicate this to my family who has supported me in thick and thin times of my career so far.

ABSTRACT

Decision Making on a Software Upgrade or Decommission with Data Mining and Machine Learning Techniques in Information

Technology Industry

RAVIKANTH KOWDEED
2024

Dissertation Chair: <Chair’s Name>
Co-Chair: <If applicable. Co-Chair’s Name>

Background:

As part of Digital Transformation needs, the Organizations are investing more in Technology and Infrastructure like software upgrades, software renewals, software replacements, Cloud migrations etc., apart from investment in Business, People, and Processes. In this context, it is not an easy task for stakeholders to decide whether to go for a software upgrade or to replace it with another software. There is no unified approach or solution available today which proactively integrates key data such as Software Versions, Platform Compatibility, Dependent Software versions, Investment and Operational Costs, Open defects and fixes, Software Performance Metrics and Service level objectives. Due to this, the so-called decision making is a manual and tedious process taking time and effort.

Research Method:

This research is quantitative and experimental, tries to simplify the decision-making process by conceptualizing and prototyping a recommendation system that is proactive and data driven in nature.

This gathers information from the Software Engineering Life Cycle stages and apply Pareto law on the metrics - 80% of consequences come from 20% of causes - after establishing relationship between the data sets, executing Machine learning models on this big data.

This research proposes relational data modelling of input data, store the input data in database tables, apply Data Mining and Machine Learning techniques on the aggregated data to derive recommendation insights on a regular basis.

- Software assets versions (source: Software release documentation)
- Platform compatible versions (source: Software feature documentation)
- Dependencies with other software (source: Software release documentation, Tools and Frameworks to manage Compile and Run time dependencies)
- Operational service level agreement needs (source: Business requirements)
- SLA and SLO requirements (source: Organizational Operational metrics)
- Quality assurance and Systems performance metrics (source: Organizational Operational metrics)
- Cyber Security vulnerability fixes (NVD reported issues and resolutions)
- Number of defects and fixes in timely manner (Defects and resolution as tracked at software level and as provided in Software release documentation)
- Investment Cost (Software cost)
- Operational Cost (Software cost for renewals/patches/maintenance)
- Estimated cost for replacement (Cost of new software adoption and decommissioning the current software)

Limitations of this Research:

Every Organization will have own challenges and learnings in modernizing their software systems. While the software vendor release notes are publicly available, the release documentation is precise, and system dependencies are complex. All this data needs to be collated and analyzed to define data relationships and variables. While a prototype process and framework are proposed, the actual derivations and recommendations on this time series data in organizations depends on lot of other factors including constant reviews and uploading them back to the public domain for reuse, which remains out of the scope topic.

Opportunities for future:

With evolution and adoption of Generative Artificial Intelligence, Organizations may leverage their own data in conjunction with other publicly available Organization case studies. This research can be further expanded to build recommendation systems using private Large Language Models which provides capabilities of having Chat bots on the Organizational data considering data privacy. The overall idea and concept remain the same i.e., have a user interface that feeds the input data set, have a background process that performs data mining and provides graphical representation of data variables and outliers, have a presentation layer that chooses the machine learning model that triggers process of generating recommendation if software needs upgrade or replacement thereby decommissioning the existing software.

TABLE OF CONTENTS

CHAPTER 1:

INTRODUCTION

This research tries to simplify the decision-making process by conceptualizing and prototyping a recommendation system that is data driven and customizable in nature. This research is about collecting critical data needed, integrating it, building a relational data model, storing data in database tables, applying Data Mining and Machine Learning techniques to derive recommendation insights on a regular basis.

All the input data is fed into the proposed recommendation system that could provide insights helping decision making of software upgrade and decommissions in the Digital Transformation exercise. This research also proposes unified solution for Software Development Upgrades and Decommissions Life Cycle as shown in picture below.

FIG. 1

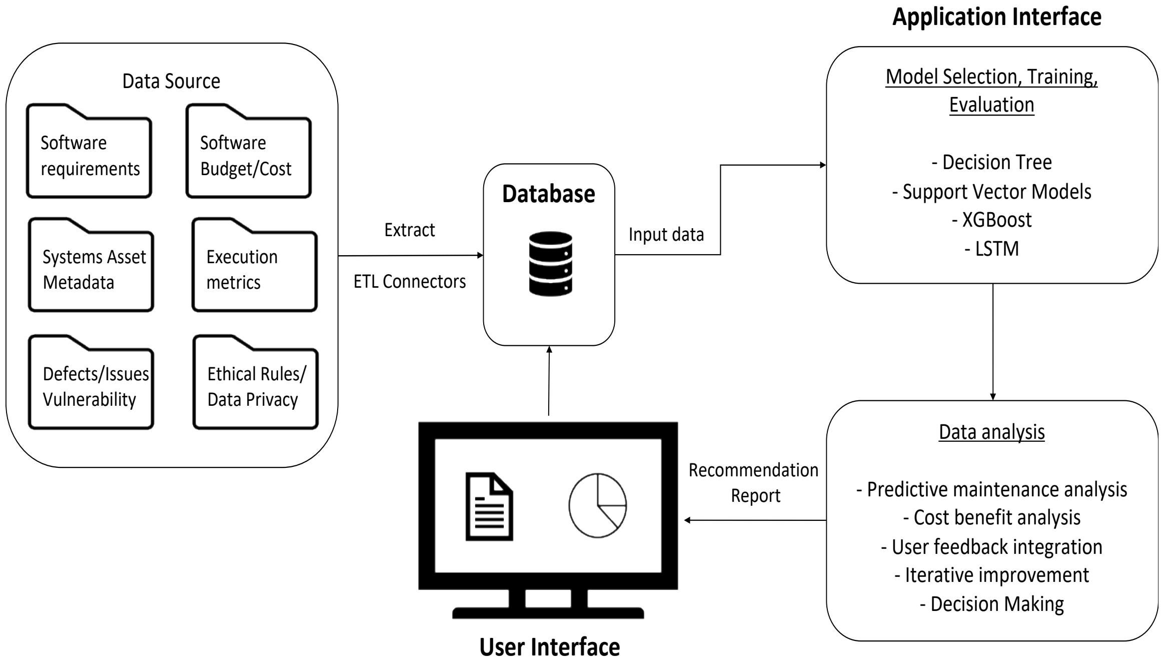
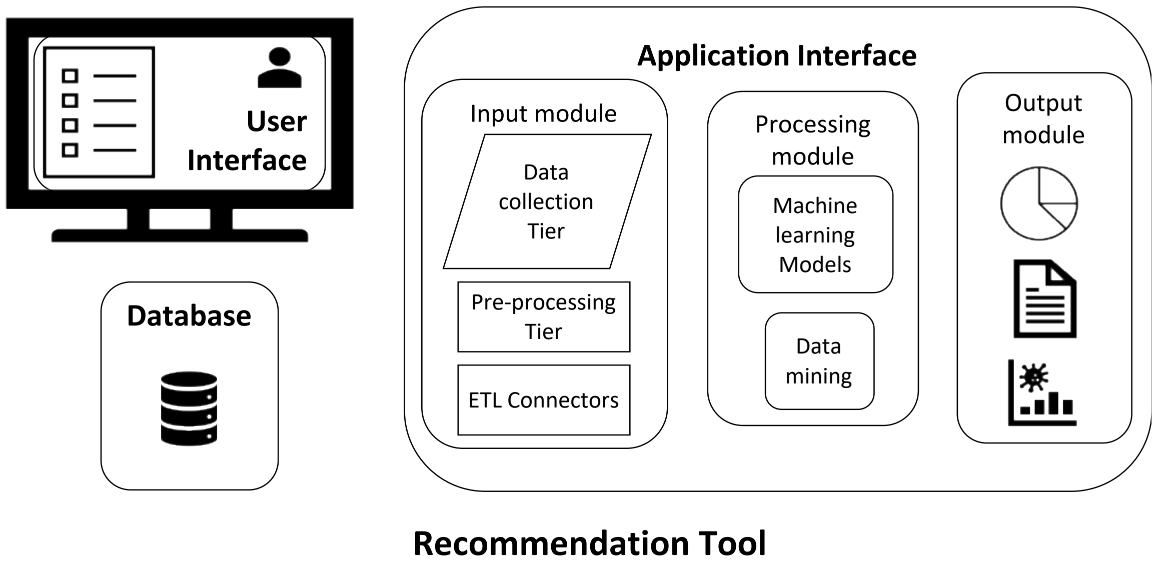

Application Interface
Data Source
Software requirements
Software Budget/Cost
Systems Asset Metadata
Execution metrics
Defects/Issues Vulnerability
Ethical Rules/ Data Privacy
Extract
ETL Connectors
Database
Input data
Model Selection, Training, Evaluation
- Decision Tree
- Support Vector Models
- XGBoost
- LSTM
Data analysis
- Predictive maintenance analysis
- Cost benefit analysis
- User feedback integration
- Iterative improvement
- Decision Making
Recommendation Report
User Interface

FIG. 2



The data is gathered from the public web sites, software release documentation and organization case studies, listed below.

- ➢ Software Requirements (Business objective, System needs, Software features, Hardware specifications, Cloud vs non-Cloud infrastructure supported).
- ➢ Software Cost (Invest vs Operate Cost i.e., installation, renewals, upgradation and decommission costs)
- ➢ System Asset Metadata (Software features, version information, service level objective, end of life date).
- ➢ Software Issues (Compatibility issues with other software and hardware, Security Vulnerabilities, Quality defects, integration errors)
- ➢ Software Execution metrics (configurations, change management, performance metrics and maintenance activity insights)

The Hypotheses are:

- ➢ System Asset data determines the right time for upgrade and decommission.
- ➢ Software and hardware systems compatibility is the key factor.
- ➢ Software and hardware system metrics help in taking timely decisions in upgrading, retiring or consolidating systems assets.

- Baseline costs (also known as planned costs) determine operational cost guidance year on year.
- Operational costs (also known as actual costs) drive systems upgrades and systems decommissions.
- The Information Technology audits and findings drive compliance requirements and available time to upgrade or decommission a particular software.
- Business service level objectives and new features drive Systems upgrades and decommissioning efforts.

1.1 Research Problem

1. Identifying what is the right time of software upgrade or decommission.
    a. Today, there is no decision tree to identify risks with a given software running on hardware since software performance is function of time besides many other factors.
2. Decision making on Software upgrade or decommission is a complex and lengthy exercise. Simplifying this process reduces manual effort involved.
    a. Today, there is no unified approach or solution to programatically run through the checklist in the entire process.
    b. There is need of continuous data integration required for software upgrades in terms of finding software stable version, updating software dependencies, checking platform compatibility, identifying end of life components, in-time defects resolution, prioritizing business requirements, considering data privacy needs and keeping a check on the overall invest and operate cost.
3. Setup a systematic process that generates data insights on a continuous basis.
    a. Propose user interface to upload the input data
    b. Build the Python based application that performs data exploration, describes variables and relationships
    c. Run machine learning models on the aggrgated data to derive key recommendations

1.2 Purpose

Helping Research community, Information Technology Industry at large, with this research proposal and thesis.

1.2 Scope of Research

1. Listing all the input data sets.
2. Data modelling
3. Define variables that need continous review i.e., operational risks, year on year technology spend, time and effort going in defect fixes, security vulnerabilities, systems degradation, service level objective misses, good to have business needs, remediating deprecated software and hardware, cloud migration for better system scalability and availaibility etc.
4. Propose a multi utility application with an UI and backend process framework that simulates Generative Artificial Intelligence Engine.
5. Apply Time Series, Decision Tree, Supervised Machine Learning and many other Machine Learning Models on the aggregated data.
6. Propose software upgrades and decomissions life cycle as a continuos process, monitored and executed on the systems data to generate recommendations in a timely manner, helping stakeholders to take decisions.

1.3 Significance of the Study

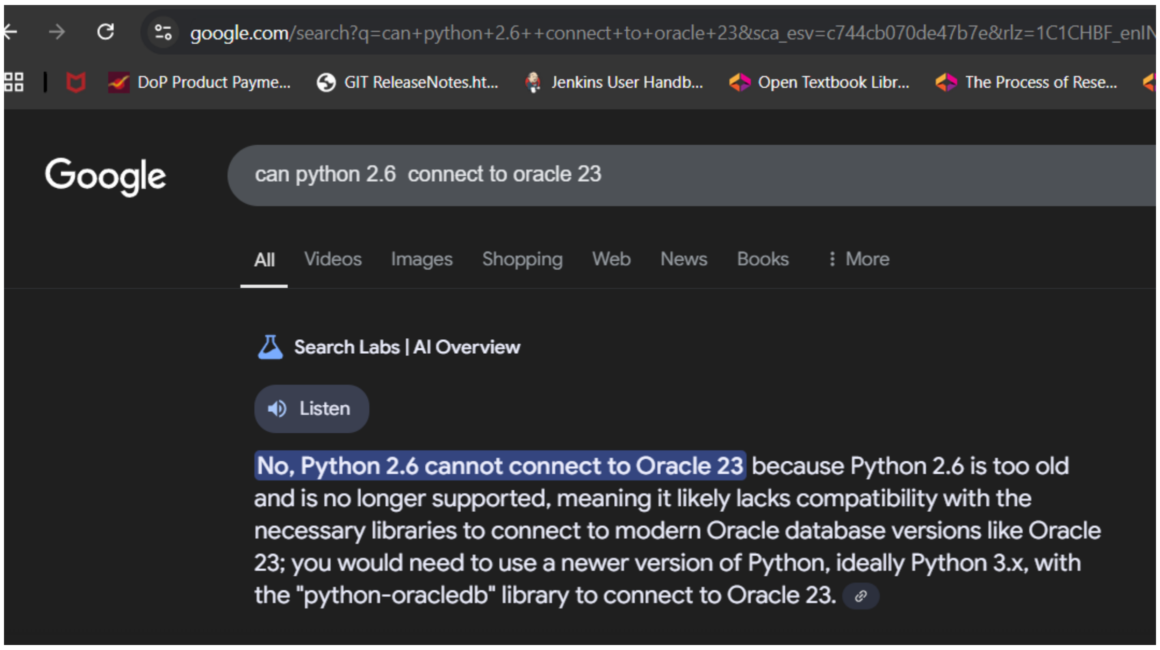
The software version management, compatibility with other software and hardware, software library dependency management, fixing security vulnerabilities, tracking invest and operate cost, tracking end of life systems, tracking turn around time in releasing new business features to customers etc are function of time and effort. There is no unified approach to integrate all this data and derive key insights for the stakeholders.

1.4 Research Questions

1. How to integrate software and hardware versions information that are compatible to each other?
2. How to integrate end of life information for a given software and hardware version ?
3. How to track cost factor, vulnerabilities and open defects in using a choosing a particular software library on a particular platform ?
4. What is the right time for a software upgrade or decommission (replace with another one)?
5. How to setup a systematic process that simulates ensemble learning on the software asset data and help in the data driven decision making?

For example, consider an application in Python 3.1 connecting to Oracle 11g R2 database. When Oracle needs upgrade to 23 version or any stable version next to 11R2, the feasibility check is needed.

See Google search screenshot below, there is a clear dependency and compatibility between two software here i.e., Python 2.6 (client) and Oracle 23 (database).


google.com/search?q=can+python+2.6++connect+to+oracle+23&sca_esv=c744cb070de47b7e&rlz=1C1CHBF_enIN
DoP Product Payme...
GIT ReleaseNotes.ht...
Jenkins User Handb...
Open Textbook Libr...
The Process of Rese...
Google
can python 2.6 connect to oracle 23
All
Videos
Images
Shopping
Web
News
Books
More
Search Labs | AI Overview
Listen

No, Python 2.6 cannot connect to Oracle 23 because Python 2.6 is too old and is no longer supported, meaning it likely lacks compatibility with the necessary libraries to connect to modern Oracle database versions like Oracle 23; you would need to use a newer version of Python, ideally Python 3.x, with the "python-oracledb" library to connect to Oracle 23.

CHAPTER 2:
REVIEW OF LITERATURE

2.1 Available Research - Findings

Every Organization has Digital Transformation goal today and one of the primary objectives is “paperless and seamless experience to the customers”. This may call for Software upgrade or a Software replacement for outdated Software Systems. There is an interesting publication on this topic by Harvard Business Review (Pombriant D, 2021). The traditional approach of Software selection often means buying the market leader or at least finding a technology solution that provides about 80% of business needs. In either case, existing software needs an upgrade (Bachwani R., et al, 2012) to address the deficiencies or go for new software, decommission (Zijden S.V.D., 2022) the old one. However, there is no guidance on what the right time is to upgrade the software or go with replacement. Key here is cost for investing in new software vis a vis upgrading existing software. If new software was chosen for obvious reasons, the existing software needs systematic decommission.

The modernization of legacy systems (Ortiz-Ochoa M., 2016) explains the problems with legacy systems but there is no guidance on data drive decision making for the systems modernization.

The periodic review on software assets data (Nouh F., 2016) is a manual task for the Systems Engineering team but is a critical responsibility to keep track of validity status of the products and software in use. There is an opportunity to relate this data with performance of software, service level agreement misses, upgrade need due to dependent software or hardware components undergoing upgrades replacements etc.

The Software performance metrics (Iqbal M., Khalid M. and Khan M.N.A., 2013) is another important piece of information to track. There is no guidance available on

defining success or error rate with software in use with respect to performance metrics. There is a need to adopt a centralized tool or methodology to capture the above-mentioned data so the software footprint like – execution traces, response times, memory spikes, space consumption, CPU overhead, cyber security issues, integration errors etc. can be known, consolidated as issues and then success likelihood can be defined.
The Quality Assurance of a Software is yet another important metric. The quicker the fault prediction (Singh M. and Chabbra J.K., 2021) and issue resolution, the easier the Software Upgrades. There is no guidance on how these metrics can be used to define the success likelihood of software upgrade or software decommission needs today.
In addition to the above, the Software Security Assurance is one more important metric that cuts across the software development and operate life cycle stages. The Software Security Metrics (Saarela M., et al, 2017) helps in continuous evaluation and resolution of security issues in the software. Integrating with SNYK (snyk.com) , which sources from National Vulnerability Database vulnerabilities and severity associated, there is a need to prioritize error resolution to stabilize the current state of software.

There are some key studies and research papers on software upgrades and decommissions:

1. Software Upgrade Strategies: Research from arXiv evaluates different software update strategies in response to security threats. The study compares strategies like immediate updates, planned updates (with delays), and reactive updates based on vulnerability disclosures. It provides insights into how enterprises can optimize their update intervals to balance security risks and operational stability.

2. Case Study on Packaged Software Upgrades: A study from the European Journal of Information Systems explores upgrade decisions in large organizations. It examines the transition of two widely used software products, SAP and Windows, highlighting how

cost, operational disruptions, and organizational factors play significant roles in deciding to upgrade.

3. Survey on Operating System Upgrades: This research investigates user behavior related to OS upgrades, focusing on why users often delay or avoid updating their systems. Factors such as fear of bugs, software compatibility, and the perceived complexity of new versions are common reasons for hesitancy.

4. Software Aging and Rejuvenation: A systematic review on software aging, which refers to the gradual degradation of software performance over time, outlines various approaches to manage this issue. The study emphasizes the importance of regular software rejuvenation and upgrades to prevent failures due to aging.

5. Risk in IT Development and Decommissioning: Research has highlighted the risks associated with decommissioning outdated IT systems, including data loss, security vulnerabilities, and operational disruptions. These studies underline the need for careful planning and phased execution when retiring legacy systems.

These provide a well-rounded perspective on the complexities involved in upgrading or decommissioning software systems, with a particular emphasis on security, cost, and organizational impacts.

2.2 Available Research - Opportunities

Below is the summary.

S. No.	**Papers/Publications**	**Available Research**	**Future Opportunities**
[1]	Fadi Nouh: 2016 ‘SAM Software Asset Management’, ResearchGate publication	This details importance of Software Asset Management which helps reduce IT costs and limit operational, financial, and	Identify critical software asset information like asset expiry, asset renewal date, cost to upgrade

		legal risks related to the ownership and use of software.	etc. and assign weightage
[2]	Rekha Bachwani. Olivier Crameri. Ricardo Bianchini and others: 2012 ‘Recommendation system for software upgrades’, ResearchGate publication	Authors in this paper have come up with recommendations for software upgrade by users, by checking the new user environment and inputs compare with existing users data whose upgrade succeeded or failed.	Collect the data such as software version, compatible platform version, other dependent software versions, cost for upgrade etc. and assign weightage
[3]	Harco Leslie Hendric Spits Warnars and 6 others: 2017 ‘Software metrics for fault prediction using machine learning approaches’, ResearchGate publication	In this paper, authors described possible software metric for fault prediction by using machine learning algorithm	Use these metrics, define weightage, derive if software upgrade is needed by training data with multiple Machine Learning models like Time Series forecasting, various Supervised Machine Learning models.
[4]	Investopedia, 2022. Feasibility Study Importance: Investopedia.com feasibility-study business essentials publication	The publishing organization has explained the feasibility checks of using any Software for Business essentials.	Need to provide weightage for Business requirements along with feasibility checks that can help choose a software

[5]	Stefan Van Der Zijden: 2022 ‘Gartner: Three key tasks needed to decommission applications’, ComputerWeekly.com	Author emphasized on the key tasks and associated effort in the decommission of an application	Consider the cost, time and effort required for the software decommission and define weightage.
[6]	Ajay Kumar and Kamaldeep Kaur: 2022 ‘Recommendation of Regression Techniques for Software Maintainability Prediction with Multi-Criteria Decision-Making’, ResearchGate	In this, the authors have recommended regression techniques for software maintainability prediction	Use these metrics and define confidence levels on the software maintainability parameters
[7]	Denis Pombriant: 2021 ‘Do you have the right software for your digital transformation’, Harvard Business Review	This article covers the need of robust, systematic, and unified systems replacing outdated systems for Digital Transformation goals, however for decision making, there is a need to capture, measure and relate all of this.	Consider the below dependencies and assign weightage: software with another software, software with a given hardware, number of data copies and integrations, number of user screens etc.
[8]	Shahid Iqbal. Muhammad Khalid and M.N.A. Khan: 2013 ‘A Distinctive Suite of Performance Metrics for Software Design’, ResearchGate publication	This research evaluates and highlights the importance of various metrics in the software performance around software design, development, and	Use these metrics and define weightage on software performance

		management, however there is a need to measure all this.	
[9]	Marko Saarela, Shohreh Hosseinzadeh, Sami Hyrynsalmi and Ville Leppänen: 2017 ‘Measuring Software Security from the Design of Software’, ResearchGate publication.	This paper studies kinds of security measurement tools (i.e., metrics) that are available, and what these tools and metrics reveal about the security of software	Use these metrics and further define weightage on widely occurring software security vulnerabilities based on OWASP[10] NVD[12] and SNYK[11] guidance
[10]	OWASP	This site stores security vulnerabilities	Reference only
[11]	SNYK	This site scans the code in GITHUB repositories and provides the list of dynamic scans and open-source vulnerabilities and fixes for the same.	Reference only
[12]	NVD	NVD stands for National Vulnerability Database, a comprehensive online database maintained by the United States Department of Homeland Security to provide detailed information on vulnerabilities in software and other systems. The database contains a vast	Reference only

		collection of vulnerability descriptions, including key factors such as severity, impact, and effective dates. The NVD is freely accessible and widely used by cybersecurity professionals, researchers, and organizations to identify and prioritize vulnerabilities, patch software, and improve overall security	

CHAPTER 3:

METHODOLOGY

This Research is based on Quantitative and Experimental methodology.

3.1 Hypotheses

Software Upgrade hypotheses

When considering software upgrades, several hypotheses can guide the decision to proceed with an upgrade. Here are some possible hypotheses to explore:

1. Improved Performance Hypothesis
   - Hypothesis: Upgrading the software will improve overall system performance, including speed, reliability, and efficiency.
   - Test: Compare system performance metrics such as processing speed, response times, and error rates between the current version and the upgraded version.

2. Enhanced Security Hypothesis
   - Hypothesis: The upgrade will enhance the security of the system, providing better protection against vulnerabilities, malware, and breaches.
   - Test: Analyze security features in the upgrade, such as updated encryption protocols, patching of known vulnerabilities, and stronger authentication mechanisms.

3. Feature Enhancement Hypothesis
   - Hypothesis: The new software version includes additional features and functionality that will benefit the organization, improve user experience, or streamline operations.
   - Test: Review the release notes for new features, gather feedback from users on desired features, and assess the impact of these improvements on workflows.

4. Increased Compatibility Hypothesis
   - Hypothesis: The software upgrade will improve compatibility with other systems, tools, or platforms, allowing better integration and interoperability.
   - Test: Test the upgraded software's ability to communicate and integrate with other systems, APIs, and modern platforms that the current version may not support.

5. Vendor Support Hypothesis
   - Hypothesis: Upgrading will extend the duration of vendor support, ensuring that the system receives ongoing maintenance, patches, and updates.
   - Test: Verify the vendor's support timeline for both the current and upgraded versions and evaluate the availability of future updates and maintenance.

6. Cost Efficiency Hypothesis

- Hypothesis: The software upgrade will result in long-term cost savings, either through better resource utilization, reduced downtime, or lower maintenance costs.
- Test: Estimate the upgrade costs versus the potential savings from improved efficiency, reduced operational costs, or avoided expenses from running outdated software.

7. User Satisfaction Hypothesis
   - Hypothesis: Users will experience increased satisfaction and productivity with the upgraded software, thanks to improved usability, performance, or new features.
   - Test: Conduct user surveys or pilot tests to gauge user reaction to the upgraded software and its impact on their work processes.

8. Scalability Hypothesis
   - Hypothesis: Upgrading the software will improve its scalability, allowing it to handle larger volumes of data, more users, or increased workloads without degradation in performance.
   - Test: Test the software's performance under increased loads or in a simulated environment to assess how well it scales after the upgrade.

9. Compliance and Regulatory Hypothesis
   - Hypothesis: The upgrade will ensure that the software remains compliant with updated industry regulations, standards, or legal requirements.
   - Test: Compare the compliance features of the current and upgraded versions, checking for any new certifications or regulatory updates the upgrade provides.

10. Stability and Bug Fixes Hypothesis
   - Hypothesis: The software upgrade will resolve bugs and known issues in the current version, leading to a more stable and reliable system.
   - Test: Review the release notes or bug tracker for fixes included in the upgrade and test for system stability improvements by monitoring error logs and crash reports.

11. Reduced Technical Debt Hypothesis
   - Hypothesis: Upgrading the software will reduce technical debt, making the system easier to maintain and more future-proof by modernizing the codebase or architecture.
   - Test: Evaluate the underlying technology, code structure, and maintainability of the upgraded version compared to the current one and assess long-term maintenance requirements.

12. Market Competitiveness Hypothesis
   - Hypothesis: Upgrading the software will make the organization more competitive in its market by enabling the use of cutting-edge features, automation, or improved analytics.
   - Test: Assess how the new features or improvements can enhance the organization's market position, comparing against competitors' capabilities.

13. Innovation Enablement Hypothesis

   - Hypothesis: The upgrade will enable the organization to innovate more rapidly, by providing tools, frameworks, or features that facilitate new product development or process improvements.
   - Test: Analyze the new capabilities introduced in the upgrade, especially those related to automation, AI, or integration with other innovation-driven platforms.

14. Ease of Adoption Hypothesis
   - Hypothesis: The software upgrade will be easy to adopt by the current user base, with minimal disruption, training, or adaptation required.
   - Test: Pilot the upgrade with a small group of users, evaluate the learning curve, and identify any barriers to adoption, such as required training or significant changes in the user interface.

Each of these hypotheses would need to be validated through testing, user feedback, and technical assessments to determine if a software upgrade is beneficial.

Software Decommissions hypotheses

When decommissioning software, various hypotheses might guide the decision-making process. Here are some potential hypotheses that could be considered:
1. Cost Savings Hypothesis
- Hypothesis: Decommissioning the software will lead to significant cost savings by eliminating licensing fees, support contracts, and maintenance costs.
- Test: Compare the current operational costs of maintaining the software versus the projected costs of replacing or retiring it.

2. Technology Obsolescence Hypothesis
- Hypothesis: The software is obsolete and no longer meets the organization's technological needs or integrates well with newer systems.
- Test: Evaluate how well the software supports modern technology standards, APIs, and workflows. Assess the impact of using outdated technology on efficiency and compatibility.

3. Security and Compliance Hypothesis
- Hypothesis: Decommissioning the software will reduce security risks and help ensure compliance with updated regulations or industry standards.
- Test: Analyse the software for vulnerabilities, outdated encryption protocols, or unsupported software components. Review any compliance gaps associated with continuing its use.

4. Low Usage Hypothesis
- Hypothesis: The software is underutilized, and the value it provides does not justify its continued operation.
- Test: Examine usage logs, user feedback, and frequency of access to determine if the software is essential or if its functions can be performed more efficiently through other tools.

5. Performance Degradation Hypothesis
- Hypothesis: The performance of the software is negatively affecting business operations, causing bottlenecks or inefficiencies.
- Test: Measure system performance, including speed, reliability, and downtime, and assess the impact on business productivity.

6. Transition to Cloud/Modern Systems Hypothesis
- Hypothesis: Migrating to cloud-based solutions or modern platforms will offer better scalability, flexibility, and lower overall operational costs compared to maintaining the legacy system.
- Test: Evaluate the benefits of cloud infrastructure or modern alternatives in terms of scalability, cost efficiency, and ability to integrate with other services.

7. Duplicate Functionality Hypothesis
- Hypothesis: The software provides functionality that is duplicated by other systems or tools, making it redundant.
- Test: Conduct an inventory of features and assess overlap with other existing or planned systems, determining whether consolidation can occur.

8. User Dissatisfaction Hypothesis
- Hypothesis: Users are dissatisfied with the software due to poor user experience, lack of features, or frequent errors, which reduces productivity.
- Test: Gather user feedback, survey employees or customers, and review incident logs to determine whether the software is causing frustration or inefficiency.

9. End-of-Life (EOL) Vendor Support Hypothesis
- Hypothesis: The software vendor has declared the software as end-of-life, meaning it will no longer provide updates, patches, or support, making the software unsustainable.
- Test: Review vendor communications and support policies to determine if ongoing support is viable and compare it with the timeline for replacement or decommissioning.

10. Data Migration Complexity Hypothesis
- Hypothesis: The complexity and risk of migrating critical data from the software to a new system justifies its continued use, at least in the short term.
- Test: Analyse the effort required for data migration, the risk of data loss or corruption, and the compatibility between the legacy system and new systems.

Each hypothesis is validated through data collection and analysis, including technical feasibility study and implementation. Details are given below.

Systems Asset documentation

This is a critical bookkeeping activity for any Organization as they are shipped from different vendors and so there are known issues, compatibility gaps between the system assets available vs needed vs used, number of resources needed vs utilized, systems uptime vs downtime, systems idle time vs busy time in Production and Non-Production environments of the Data Centers. The continued monitoring involved here is manual in nature to track what versions are being used, what are being upgraded, what are decommissioned, which code or configurations are obsolete, redundancy factors needed for systems high availability, tracking system alert behavior and patterns, backup and resiliency of system assets, tracking defects and their resolution, and of course reviewing when to go for upgrade or decommissioning of software or hardware.

Hypothesis: asset data determines when system upgrade is needed.

Software and Hardware Compatibility
Software and Hardware compatibility refers to affinity between software version and associated hardware platform. It is measured by success rate of regular health checks including security scans, execution time, defect resolution turnaround time, system response time after patching or upgrading exercise. With ever increasing demand in software usage along with Artificial Intelligence capabilities and Digital Transformation needs, decisions are taken at the top level and then cascaded to the lower levels. This often leads to improper planning of assets needed to upgrade or decommission because there will be a need of tracking existing issues, open defects and security risks to resolve, tracking end of life components, replacing them with right assets at the right time with minimum down time. So, the level of uncertainty associated with software upgrade or software decommission is usually high when the health of Software and Hardware is not tracked. Hence the need to collect data and metrics associated to assets, on a continued basis.

Hypothesis: software and hardware systems compatibility influence systems upgrade, or systems decommission.

Collecting Metrics
Collecting metrics is an important task in the software and hardware health check activity. The metrics such as software issues, hardware issues, time taken for regular patches, new issues, security findings, increase in operational cost, increase or decrease in renewal cost, additional upgrade cost, service level objectives w.r.t assets performance, system components to retire etc need to be saved at a centralized location, dependencies to be determined and reviewed periodically.

Hypothesis: software and hardware system metrics help in taking timely decisions in upgrading, retiring, consolidating assets.

Planned Cost
The cost incurred with software and hardware assets installation and maintenance is another important aspect. The same is used as reference against operational cost to see

if there are any deviations. The overall IT expenditure of an organization in a given fiscal year considers this as baseline cost.

Hypothesis: baselined planned costs determine operational cost guidance year on year.

Actual Cost

This is the accrued cost in maintenance of software and hardware assets. The overall IT expenditure of an organization comprises of this actual budget spent in the fiscal year against the planned budget guidance start of the year. The profit or loss margins of an organization depend on this important piece of information.

Hypothesis: operational costs drive systems upgrades and systems decommissions.

Internal Audit

The organizations do periodic internal IT system audits of software and hardware components nearing upgrades, end of life, having security risks and vulnerabilities, performance issues to name a few. This is a planning and monitoring exercise where everyone acts according to guidelines defined by the IT Systems head of the department, under the supervision of the Chief Technology Officer and Chief Information Officer. The information tracking needs to be maintained at a certain centralized location, so root cause analysis can take place when things don’t go as expected. Therefore, there is a need to use Machine Learning models to churn the system assets data and system activity data for better decision making considering current and future needs of IT systems.

Hypothesis: The IT audit findings drive Systems upgrade and decommissioning need.

3.2 Research Design

The Research design is to collect various data sets, model the data, load that to relational tables, analyze the data, aggregate the data per software asset version and then find the evidence for the hypotheses by applying the machine learning models on the resultant aggregated data.

3.3 Population and Sample

Software versions and compatible versions of other software and hardware

28 years data of Java versions,

- Java JDK 1.0 released in 1996 and current stable version is 21

30 years data of Python versions.

  - Python 1.0 released in 1994 and current stable version is 3.12

22 years data of Microsoft .NET version and end of life/end of support data.

  - .NET framework 1.0 released in 2002 and current stable version is 4.8

32 years data of Oracle and MySQL

  - Oracle 7 released in 92 and current stable version is 21c

Install and operate cost of software on hardware platform

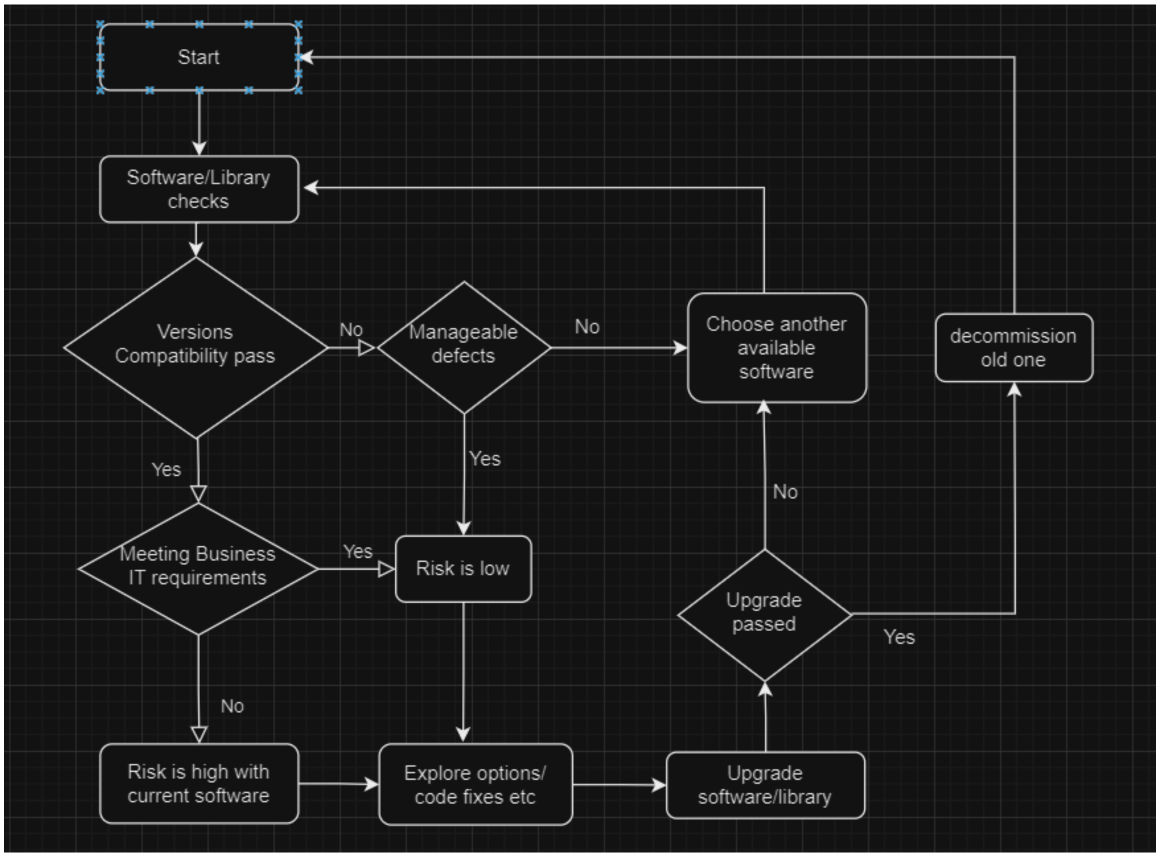
Open defects and security vulnerabilities associated with software library

Sample data is ~ 200K records

3.4 Instrumentation

The data is collected from the publicly available software websites, release documentation pages, such data is loaded comma separated files. Python program is created to read such data into pandas data frames, identify risk by applying pareto law based on cumulative defects of the software and dependent software defects. Below are the steps to follow.

1. Iterate through the software library version
    a. Check if the version and its dependent software version are compatible to each other. If yes, go to 2. below. If no, get the open defects + security vulnerabilities count if they can be resolved before end of life. If yes, go to 2. Below. If no, go to step.b
    b. Choose another software , pick the feasible version which has lower install cost, lesser defects, passing compatibility and repeat from 1.
2. Check if meeting business requirements and no missed Service Level Objectives. If yes, go to 3.
3. Perform software upgrade, test with configuration changes, perform testing and go live validations. Decommission the old software.


Start
Software/Library checks
Versions Compatibility pass
No
Manageable defects
No
Choose another available software
decommission old one
Yes
Yes
No
Meeting Business IT requirements
Yes
Risk is low
Upgrade passed
Yes
No
Risk is high with current software
Explore options/ code fixes etc
Upgrade software/library

Thus, the risk assessment and guidance helps stakeholders address the problems and take informed decisions.

The overall approach is integrate this in the program that automates the manual steps required in identifying bottlenecks with use of Data Mining and Machine Learning Models.

Below mentioned are several case studies that provide valuable lessons on software upgrades, highlighting both successful and failed expercises across different industries.

1. Microsoft’s Transition from Windows XP to Windows 10

- Background: Many organizations used Windows XP long after it had been declared end-of-life (EOL) in 2014. This created a need for upgrades to newer operating systems like Windows 7 or Windows 10.

- Challenges: Upgrading from XP to Windows 10 required hardware compatibility checks, application re-engineering, and user training. Some organizations faced compatibility issues with legacy applications, which had been built for XP.

Lessons:

- Lesson 1: Upgrading critical software systems often necessitates simultaneous hardware upgrades.

- Lesson 2: Compatibility testing is essential, especially for legacy systems that may not run properly on new platforms.

- Lesson 3: Proactive migration planning helps avoid the risks associated with running unsupported software, such as security vulnerabilities and lack of vendor support.

- Outcome: Successful migrations occurred where organizations thoroughly planned the transition, accounted for downtime, and communicated changes to users.

2. Netflix’s Migration to the Cloud

- Background: Netflix transitioned from a traditional data center infrastructure to a fully cloud-based architecture in order to better scale and meet demand. This required significant software upgrades.

- Challenges: During the migration, Netflix encountered challenges in maintaining service uptime while re-architecting its systems to function in the cloud. Additionally, they had to train engineers on new tools and methodologies suited to cloud architecture.

Lessons:

- Lesson 1: Incremental upgrades, such as shifting parts of the infrastructure to the cloud over time, can minimize risk and allow for ongoing operations without major disruptions.

- Lesson 2: Cloud-based upgrades enable greater scalability but require new operational processes and tools, which may necessitate re-skilling the workforce.

- Lesson 3: A phased approach allows for testing and validation at each stage, reducing the risk of full-system failure during upgrades.

- Outcome: Netflix’s cloud migration was highly successful, enabling it to scale rapidly to meet global demand and maintain service reliability.

3. Adobe Creative Cloud: Transition from License-Based to Subscription Model

- Background: Adobe transitioned its Creative Suite software from a perpetual license model to a subscription-based model with Creative Cloud in 2013, which involved significant software and business model upgrades.

- Challenges: Many users were resistant to the subscription model, preferring to own perpetual licenses. Adobe also had to ensure that the cloud-based delivery of software maintained high performance and reliability.

Lessons:

- Lesson 1: Upgrading to a new software delivery model can impact customer satisfaction if not communicated effectively. Transparency about the benefits and long-term value is crucial.

- Lesson 2: New software models, such as SaaS (Software as a Service), require careful consideration of pricing structures, value propositions, and customer education.

- Lesson 3: Continuous updates and cloud delivery can improve customer retention, as users consistently receive the latest features and improvements.

- Outcome: Adobe successfully transitioned its business model, becoming a leader in the subscription software market, with increased recurring revenue and customer base growth.

4. UK National Health Service (NHS) ERP Upgrade

- Background: The NHS decided to upgrade its ERP (Enterprise Resource Planning) system to better manage healthcare operations and patient data. This upgrade aimed to improve financial management and patient services.

- Challenges: The NHS faced technical and organizational challenges during the upgrade, including integration issues with legacy systems, data migration challenges, and employee training.

Lessons:

- Lesson 1: Large-scale software upgrades require significant investment in training and change management to ensure end users can utilize the new system effectively.

- Lesson 2: Integrating upgraded software with legacy systems can be complex and requires detailed planning and testing.

- Lesson 3: Proper stakeholder engagement is critical. Getting feedback from healthcare staff, administrative personnel, and IT teams early on can prevent disruptions.

- Outcome: Despite initial setbacks, the NHS achieved improvements in financial oversight and patient management, but only after addressing post-upgrade usability issues.

5. Slack's Backend Infrastructure Upgrade

- Background: Slack, the team collaboration tool, upgraded its backend infrastructure to improve performance and handle larger user volumes. The goal was to reduce downtime and improve the speed of message delivery.

- Challenges: During the upgrade, Slack experienced some downtime and performance issues, particularly when moving away from a monolithic architecture to microservices.

Lessons:

- Lesson 1: Upgrading a high-availability system requires meticulous planning and testing to avoid service interruptions for end users.

- Lesson 2: Switching to new architectural paradigms (e.g., microservices) during an upgrade can introduce unforeseen complications and necessitates new monitoring and troubleshooting tools.

- Lesson 3: Post-upgrade performance should be continuously monitored, with a strategy in place to address emerging issues quickly.

- Outcome: Slack successfully transitioned to a more scalable infrastructure, reducing downtime and improving overall system responsiveness, but not without temporary challenges along the way.

6. Target's Website Upgrade (2011)

- Background: In 2011, Target upgraded its website infrastructure, shifting from a third-party platform (Amazon) to an in-house platform. This upgrade was meant to improve user experience and provide greater control over the e-commerce platform.

- Challenges: The transition led to a series of outages, particularly during high-traffic periods, like the launch of a major fashion collection. The new platform struggled to handle the load, leading to customer frustration.

Lessons:

- Lesson 1: High-traffic, customer-facing software upgrades require robust load testing and contingency planning for handling demand spikes.

- Lesson 2: Insufficient testing or inadequate infrastructure can lead to loss of revenue and brand reputation if the upgraded system fails during peak usage times.

- Lesson 3: Cross-functional collaboration (e.g., between marketing, IT, and customer service) is crucial when launching upgrades that impact customer experience directly.

- Outcome: Target recovered from the initial challenges, but the experience highlighted the importance of rigorous testing and scalability planning before a major software upgrade.

7. Toyota’s Transition to SAP ERP

- Background: Toyota implemented SAP ERP across its global operations to standardize processes and improve data management.

- Challenges: The upgrade was massive in scale and scope, spanning multiple geographies, and it required Toyota to rethink its existing processes to fit within SAP’s framework.

Lessons:

- Lesson 1: Global software upgrades require alignment between local and corporate needs, as well as thorough testing in different operational contexts.

- Lesson 2: Business process reengineering is often necessary when upgrading enterprise systems. Companies must be prepared to adapt their processes to fit the new system.

- Lesson 3: Having a dedicated project management office (PMO) to oversee large upgrades helps ensure that timelines, budgets, and resources are managed effectively.

- Outcome: Toyota successfully implemented the SAP ERP system, streamlining operations and improving data visibility, but the transition required years of planning and execution.

These case studies emphasize the importance of planning, stakeholder engagement, and testing when upgrading software. Key lessons include the need for phased rollouts, comprehensive testing, and careful consideration of user experience to ensure a smooth transition.

3.5 Data Collection Procedures

Download publicly available data from software release documentation.

3.6 Data Analysis

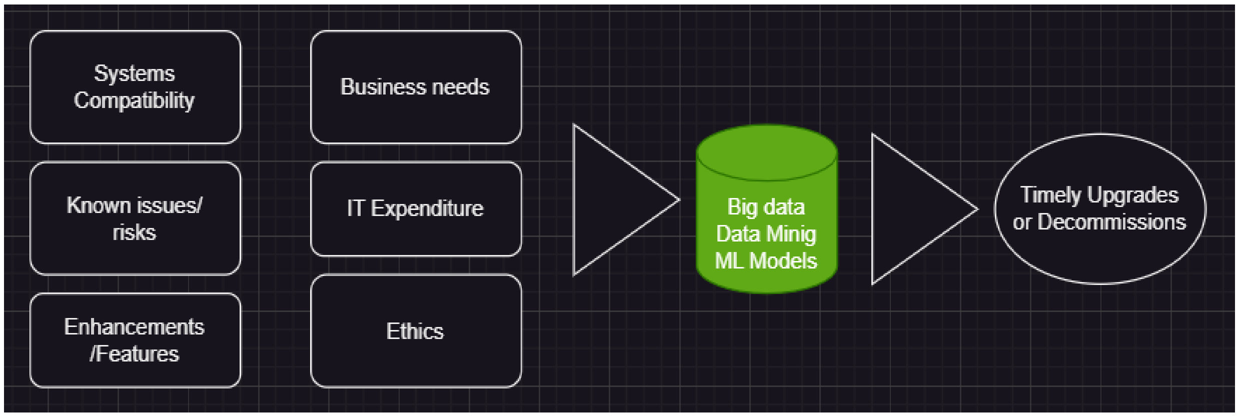
Load the collected data in to csv files, use Python libraries to create the data frames, join the data sets, explore and mine the data so Machine Learning models can be run on it.

3.7 Limitations

The research is based on few data sets from software vendors like Java, Python, Oracle , MySQL and Microsoft .NET as they are the popular technologies today. These softwares will in turn have many libraries with dependencies and compatibility versions. There are many other softwares, libraries like e-Business Suite products, open source by products that can be used in future.

3.8 Proposed prototype solution

Given below is the process flow proposed for Software Upgrades and Decommissions Life Cycle along with Software Development Life Cycle. This is a continuous process for generating recommendation insights from the input data sets on a regular basis.



CHAPTER 4:

EXPERIMENTATION RESULTS

4.1 Extracting software and hardware compatible version information

This information is downloaded from the software release documentation pages. For example, Oracle, Java, Python, MySQL and Microsoft .NET data is collated as below.

Software/ Library	**Version**	**Release Year**	**General Compatibility Notes**
Oracle Database	11g	2007	Compatible with Java 6, Java 7; connectors provided for .NET.
	12c	2013	Supports Java 7, Java 8; connectors for Python (cx_Oracle), .NET.
	18c	2018	Compatible with Java 8, Java 11; supports Python connectors and .NET through Oracle Data Provider.
	19c	2019	Compatible with Java 8, Java 11, and Java 17 (LTS); Python connectors, .NET Core, and .NET 5/6 supported.
	21c	2021	Compatible with Java 11 and Java 17; supports Python connectors and .NET 6.
Java JDK	6	2006	Compatible with Oracle Database 10g/11g; commonly used in MySQL 5.x applications; limited .NET interop with JNI.
	7	2011	Improved performance; compatible with Oracle Database 11g/12c, MySQL 5.5+, .NET integrations via web services.
	8 (LTS)	2014	Widely compatible with Oracle Database 11g/12c, MySQL 5.5+, and Python 2/3 through JDBC-ODBC bridges.
	11 (LTS)	2018	Compatible with Oracle Database 12c/19c; better performance with MySQL 8.0; integrates well with .NET through web APIs.
	17 (LTS)	2021	Preferred for Oracle 19c/21c; compatible with MySQL 8.0; works alongside Python 3.7+ and .NET Core/6.
MySQL	5.5	2010	Compatible with Java 6, Java 7, and Python 2/3; basic compatibility with .NET.
	5.7	2015	Preferred with Java 8 and Python 3.6+; supported by .NET Core connectors.
	8	2018	Best suited for Java 11/17 and Python 3.7+; compatible with .NET Core/6 connectors.

Microsoft .NET	.NET Framework 4.x	2010–2019	Works with Oracle 11g/12c and MySQL 5.x; some interop with Java (web services).
	.NET Core 3.x	2019	Preferred for Oracle 18c/19c and MySQL 8.0; supports Python libraries.
	.NET 5	2020	Compatible with Oracle 19c/21c and MySQL 8.0; works with Java APIs through REST.
	.NET 6	2021	Best for Oracle 21c and MySQL 8.0; highly compatible with Java APIs and Python 3.8+.
Python	2.7	2010	Limited support for Oracle 11g/12c and MySQL 5.x; deprecated in most modern setups.
	3.6	2016	Compatible with Oracle 12c/18c and MySQL 5.7/8.0; limited interop with Java/.NET.
	3.7	2018	Improved compatibility with Oracle 18c/19c, MySQL 8.0; works alongside Java 11+.
	3.9	2020	Preferred for Oracle 19c/21c and MySQL 8.0; integrates well with Java 17 and .NET Core/6.
	3.10+	2021+	Best compatibility with modern Oracle, MySQL, and .NET ecosystems; works seamlessly with Java 17+.

4.2 Listing software versions compatibility information

This information can be downloaded from the software release documentation pages. For example, Oracle , Java, Python, MySQL and Microsoft .NET data is collated as below.

Software Version	**Dependent Software Version**	**Compatible**
Java 6	Oracle Database 10g	Yes
Java 6	Oracle Database 11g	Yes
Java 7	Oracle Database 11g	Yes
Java 7	Oracle Database 12c	Yes
Java 8	Oracle Database 11g	No
Java 8	Oracle Database 12c	Yes
Java 8	Oracle Database 18c	Yes
Java 8	Oracle Database 19c	Yes
Java 9	Oracle Database 12c	No
Java 9	Oracle Database 18c	No
Java 9	Oracle Database 19c	No
Java 10	Oracle Database 12c	No
Java 10	Oracle Database 18c	No

Java 10	Oracle Database 19c	No
Java 11	Oracle Database 12c	Yes
Java 11	Oracle Database 18c	Yes
Java 11	Oracle Database 19c	Yes
Java 11	Oracle Database 21c	Yes
Java 12	Oracle Database 19c	Yes
Java 12	Oracle Database 21c	Yes
Java 13	Oracle Database 19c	Yes
Java 13	Oracle Database 21c	Yes
Java 14	Oracle Database 19c	Yes
Java 14	Oracle Database 21c	Yes
Java 15	Oracle Database 19c	Yes
Java 15	Oracle Database 21c	Yes
Java 16	Oracle Database 19c	Yes
Java 16	Oracle Database 21c	Yes
Java 17	Oracle Database 19c	Yes
Java 17	Oracle Database 21c	Yes
Java 18	Oracle Database 19c	Yes
Java 18	Oracle Database 21c	Yes
Java 19	Oracle Database 19c	Yes
Java 19	Oracle Database 21c	Yes
Java 21	Oracle Database 12c	No
Java 21	Oracle Database 18c	No
Java 21	Oracle Database 19c	Yes
Java 21	Oracle Database 21c	Yes
Java 21	Oracle Database 23c	Yes
Python 2.7	Oracle 10g	Yes
Python 2.7	Oracle 11g	Yes
Python 2.7	Oracle 12c	Yes
Python 2.7	Oracle 18c	Yes
Python 2.7	Oracle 19c	Yes
Python 3.4	Oracle 10g	No
Python 3.4	Oracle 11g	Yes
Python 3.4	Oracle 12c	Yes
Python 3.4	Oracle 18c	Yes
Python 3.4	Oracle 19c	Yes
Python 3.5	Oracle 10g	No
Python 3.5	Oracle 11g	Yes
Python 3.5	Oracle 12c	Yes
Python 3.5	Oracle 18c	Yes

Python 3.5	Oracle 19c	Yes
Python 3.6	Oracle 10g	No
Python 3.6	Oracle 11g	Yes
Python 3.6	Oracle 12c	Yes
Python 3.6	Oracle 18c	Yes
Python 3.6	Oracle 19c	Yes
Python 3.7	Oracle 10g	No
Python 3.7	Oracle 11g	Yes
Python 3.7	Oracle 12c	Yes
Python 3.7	Oracle 18c	Yes
Python 3.7	Oracle 19c	Yes
Python 3.8	Oracle 10g	No
Python 3.8	Oracle 11g	Yes
Python 3.8	Oracle 12c	Yes
Python 3.8	Oracle 18c	Yes
Python 3.8	Oracle 19c	Yes
Python 3.9	Oracle 10g	No
Python 3.9	Oracle 11g	No
Python 3.9	Oracle 12c	Yes
Python 3.9	Oracle 18c	Yes
Python 3.9	Oracle 19c	Yes
Python 3.10	Oracle 10g	No
Python 3.10	Oracle 11g	No
Python 3.10	Oracle 12c	Yes
Python 3.10	Oracle 18c	Yes
Python 3.10	Oracle 19c	Yes
Python 3.11	Oracle 10g	No
Python 3.11	Oracle 11g	No
Python 3.11	Oracle 12c	Yes
Python 3.11	Oracle 18c	Yes
Python 3.11	Oracle 19c	Yes
Oracle 8i	.NET Framework 1.0	No
Oracle 8i	.NET Framework 1.1	No
Oracle 8i	.NET Framework 2.0	No
Oracle 8i	.NET Framework 3.0	No
Oracle 8i	.NET Framework 3.5	No
Oracle 9i	.NET Framework 1.0	No
Oracle 9i	.NET Framework 1.1	No
Oracle 9i	.NET Framework 2.0	No
Oracle 9i	.NET Framework 3.0	No

Oracle 9i	.NET Framework 3.5	No
Oracle 10g	.NET Framework 1.0	No
Oracle 10g	.NET Framework 1.1	No
Oracle 10g	.NET Framework 2.0	Yes
Oracle 10g	.NET Framework 3.0	Yes
Oracle 10g	.NET Framework 3.5	Yes
Oracle 11g	.NET Framework 2.0	Yes
Oracle 11g	.NET Framework 3.0	Yes
Oracle 11g	.NET Framework 3.5	Yes
Oracle 11g	.NET Framework 4.0	Yes
Oracle 11g	.NET Framework 4.5	Yes
Oracle 12c	.NET Framework 4.0	Yes
Oracle 12c	.NET Framework 4.5	Yes
Oracle 12c	.NET Framework 4.6	Yes
Oracle 12c	.NET Framework 4.7	Yes
Oracle 12c	.NET Framework 4.8	Yes
Oracle 18c	.NET Framework 4.5	Yes
Oracle 18c	.NET Framework 4.6	Yes
Oracle 18c	.NET Framework 4.7	Yes
Oracle 18c	.NET Framework 4.8	Yes
Oracle 18c	.NET 5.0	Yes
Oracle 18c	.NET 6.0	Yes
Oracle 19c	.NET Framework 4.5	Yes
Oracle 19c	.NET Framework 4.6	Yes
Oracle 19c	.NET Framework 4.7	Yes
Oracle 19c	.NET Framework 4.8	Yes
Oracle 19c	.NET 5.0	Yes
Oracle 19c	.NET 6.0	Yes
Oracle 21c	.NET Framework 4.5	Yes
Oracle 21c	.NET Framework 4.6	Yes
Oracle 21c	.NET Framework 4.7	Yes
Oracle 21c	.NET Framework 4.8	Yes
Oracle 21c	.NET 5.0	Yes
Oracle 21c	.NET 6.0	Yes
Oracle 21c	.NET 7.0	Yes
Oracle 21c	.NET 8.0	Yes
Java 1.0	MySQL 3.x	No
Java 1.0	MySQL 4.x	No
Java 1.1	MySQL 3.x	No
Java 1.1	MySQL 4.x	No

Java 1.2	MySQL 4.x	No
Java 1.2	MySQL 5.0	No
Java 1.3	MySQL 4.x	No
Java 1.3	MySQL 5.0	No
Java 1.4	MySQL 4.x	No
Java 1.4	MySQL 5.0	No
Java 1.4	MySQL 5.1	No
Java 5.0	MySQL 5.0	Yes
Java 5.0	MySQL 5.1	Yes
Java 5.0	MySQL 5.5	Yes
Java 6	MySQL 5.0	Yes
Java 6	MySQL 5.1	Yes
Java 6	MySQL 5.5	Yes
Java 6	MySQL 5.6	Yes
Java 7	MySQL 5.1	Yes
Java 7	MySQL 5.5	Yes
Java 7	MySQL 5.6	Yes
Java 7	MySQL 5.7	Yes
Java 8	MySQL 5.5	Yes
Java 8	MySQL 5.6	Yes
Java 8	MySQL 5.7	Yes
Java 8	MySQL 8.0	Yes
Java 9	MySQL 5.6	Yes
Java 9	MySQL 5.7	Yes
Java 9	MySQL 8.0	Yes
Java 10	MySQL 5.6	Yes
Java 10	MySQL 5.7	Yes
Java 10	MySQL 8.0	Yes
Java 11	MySQL 5.6	Yes
Java 11	MySQL 5.7	Yes
Java 11	MySQL 8.0	Yes
Java 12	MySQL 5.7	Yes
Java 12	MySQL 8.0	Yes
Java 13	MySQL 5.7	Yes
Java 13	MySQL 8.0	Yes
Java 14	MySQL 5.7	Yes
Java 14	MySQL 8.0	Yes
Java 15	MySQL 8.0	Yes
Java 16	MySQL 8.0	Yes
Java 17	MySQL 8.0	Yes

Java 18	MySQL 8.0	Yes
Java 19	MySQL 8.0	Yes
Java 20	MySQL 8.0	Yes
Java 21	MySQL 8.0	Yes
MySQL 3.x	.NET Framework 1.0	No
MySQL 3.x	.NET Framework 1.1	No
MySQL 3.x	.NET Framework 2.0	No
MySQL 3.x	.NET Framework 3.0	No
MySQL 3.x	.NET Framework 3.5	No
MySQL 4.x	.NET Framework 1.0	No
MySQL 4.x	.NET Framework 1.1	No
MySQL 4.x	.NET Framework 2.0	Yes
MySQL 4.x	.NET Framework 3.0	Yes
MySQL 4.x	.NET Framework 3.5	Yes
MySQL 5.0	.NET Framework 1.0	No
MySQL 5.0	.NET Framework 1.1	No
MySQL 5.0	.NET Framework 2.0	Yes
MySQL 5.0	.NET Framework 3.0	Yes
MySQL 5.0	.NET Framework 3.5	Yes
MySQL 5.1	.NET Framework 2.0	Yes
MySQL 5.1	.NET Framework 3.0	Yes
MySQL 5.1	.NET Framework 3.5	Yes
MySQL 5.5	.NET Framework 3.5	Yes
MySQL 5.5	.NET Framework 4.0	Yes
MySQL 5.5	.NET Framework 4.5	Yes
MySQL 5.6	.NET Framework 4.0	Yes
MySQL 5.6	.NET Framework 4.5	Yes
MySQL 5.6	.NET Framework 4.6	Yes
MySQL 5.7	.NET Framework 4.5	Yes
MySQL 5.7	.NET Framework 4.6	Yes
MySQL 5.7	.NET Framework 4.7	Yes
MySQL 5.7	.NET Framework 4.8	Yes
MySQL 8.0	.NET Framework 4.6	Yes
MySQL 8.0	.NET Framework 4.7	Yes
MySQL 8.0	.NET Framework 4.8	Yes
MySQL 8.0	.NET 5.0	Yes
MySQL 8.0	.NET 6.0	Yes
MySQL 8.0	.NET 7.0	Yes
MySQL 8.0	.NET 8.0	Yes
Python 2.7	MySQL 3.x	No

Python 2.7	MySQL 4.x	No
Python 2.7	MySQL 5.0	Yes
Python 2.7	MySQL 5.1	Yes
Python 2.7	MySQL 5.5	Yes
Python 2.7	MySQL 5.6	Yes
Python 2.7	MySQL 5.7	Yes
Python 2.7	MySQL 8.0	No
Python 3.4	MySQL 5.1	Yes
Python 3.4	MySQL 5.5	Yes
Python 3.4	MySQL 5.6	Yes
Python 3.4	MySQL 5.7	Yes
Python 3.4	MySQL 8.0	No
Python 3.5	MySQL 5.5	Yes
Python 3.5	MySQL 5.6	Yes
Python 3.5	MySQL 5.7	Yes
Python 3.5	MySQL 8.0	No
Python 3.6	MySQL 5.6	Yes
Python 3.6	MySQL 5.7	Yes
Python 3.6	MySQL 8.0	Yes
Python 3.7	MySQL 5.6	Yes
Python 3.7	MySQL 5.7	Yes
Python 3.7	MySQL 8.0	Yes
Python 3.8	MySQL 5.7	Yes
Python 3.8	MySQL 8.0	Yes
Python 3.9	MySQL 5.7	Yes
Python 3.9	MySQL 8.0	Yes
Python 3.10	MySQL 5.7	Yes
Python 3.10	MySQL 8.0	Yes
Python 3.11	MySQL 5.7	Yes
Python 3.11	MySQL 8.0	Yes

4.3 Knowing end of life information of a given software library version

This information can be downloaded from the software release documentation pages. For example, Oracle, Java, Python, MySQL and Microsoft .NET data is collated as below.

Software Version	**Release Date**	**End of Life (EOL) Date**
JDK 1.0	23-01-1996	01-01-2000
JDK 1.1	19-02-1997	01-01-2000
JDK 1.2	08-12-1998	30-10-2003
JDK 1.3	08-05-2000	05-02-2006

JDK 1.4	13-02-2002	30-10-2008
JDK 5.0	30-09-2004	30-10-2009
JDK 6	11-12-2006	31-12-2018
JDK 7	28-07-2011	29-07-2022
JDK 8	18-03-2014	2030-12-31 (LTS)
JDK 9	21-09-2017	01-03-2018
JDK 10	20-03-2018	30-09-2018
JDK 11	25-09-2018	2026-09-30 (LTS)
JDK 12	19-03-2019	30-09-2019
JDK 13	17-09-2019	30-03-2020
JDK 14	17-03-2020	30-09-2020
JDK 15	15-09-2020	30-03-2021
JDK 16	16-03-2021	30-09-2021
JDK 17	14-09-2021	2029-09-30 (LTS)
JDK 18	22-03-2022	30-09-2022
JDK 19	20-09-2022	31-03-2023
JDK 20	21-03-2023	30-09-2023
JDK 21	19-09-2023	2031-09-30 (LTS)
Oracle 7	01-06-1992	31-12-2004
Oracle 8	01-06-1997	31-12-2004
Oracle 8i	01-03-1999	31-12-2006
Oracle 9i	01-06-2001	31-07-2007
Oracle 10g	01-01-2003	31-07-2013
Oracle 10g R2	01-09-2005	31-07-2015
Oracle 11g	01-08-2007	31-08-2015
Oracle 11g R2	01-09-2009	31-12-2021
Oracle 12c	01-06-2013	31-07-2018
Oracle 12c R2	01-03-2016	31-03-2022
Oracle 18c	16-02-2018	30-06-2021
Oracle 19c	13-01-2019	2027-04-30 (Extended)
Oracle 21c	01-01-2021	31-12-2023
Oracle 23c	01-04-2023	31-12-2031
Python 1.0	26-01-1994	27-10-1996
Python 1.5	31-12-1998	05-09-2000
Python 1.6	05-09-2000	12-06-2002
Python 2.0	16-10-2000	22-06-2001
Python 2.1	17-04-2001	01-04-2006
Python 2.2	21-12-2001	01-04-2006
Python 2.3	29-07-2003	11-03-2008
Python 2.4	30-11-2004	18-10-2008

Python 2.5	19-09-2006	26-05-2011
Python 2.6	01-10-2008	29-10-2013
Python 2.7	03-07-2010	01-01-2020
Python 3.0	03-12-2008	27-06-2009
Python 3.1	27-06-2009	09-04-2012
Python 3.2	20-02-2011	20-02-2016
Python 3.3	29-09-2012	29-09-2017
Python 3.4	16-03-2014	18-03-2019
Python 3.5	13-09-2015	13-09-2020
Python 3.6	23-12-2016	23-12-2021
Python 3.7	27-06-2018	27-06-2023
Python 3.8	14-10-2019	14-10-2024
Python 3.9	05-10-2020	05-10-2025
Python 3.10	04-10-2021	04-10-2026
Python 3.11	24-10-2022	24-10-2027
Python 3.12	02-10-2023	02-10-2028
.NET Framework 1.0	13-02-2002	14-07-2009
.NET Framework 1.1	24-04-2003	08-10-2013
.NET Framework 2.0	07-11-2005	2011-07-12 (mainstream), 2016-07-12 (extended)
.NET Framework 3.0	06-11-2006	2011-07-12 (mainstream), 2016-07-12 (extended)
.NET Framework 3.5	19-11-2007	09-01-2029
.NET Framework 3.5 SP1	18-11-2008	09-01-2029
.NET Framework 4.0	12-04-2010	12-01-2016
.NET Framework 4.5	15-08-2012	12-01-2016
.NET Framework 4.5.1	17-10-2013	12-01-2016
.NET Framework 4.5.2	05-05-2014	26-04-2023
.NET Framework 4.6	20-07-2015	26-04-2023
.NET Framework 4.6.1	30-11-2015	26-04-2023
.NET Framework 4.6.2	02-08-2016	12-01-2027
.NET Framework 4.7	05-04-2017	12-01-2027
.NET Framework 4.7.1	17-10-2017	12-01-2027
.NET Framework 4.7.2	30-04-2018	09-01-2029
.NET Framework 4.8	18-04-2019	09-01-2029
.NET Framework 4.8.1	09-08-2022	09-01-2029
MySQL 4.1	05-10-2004	31-12-2009
MySQL 5	19-10-2005	31-12-2012
MySQL 5.1	14-11-2008	31-12-2013
MySQL 5.5	03-12-2010	03-12-2018

MySQL 5.6	05-02-2013	05-02-2021
MySQL 5.7	21-10-2015	21-10-2023
MySQL 8	19-04-2018	2026-04-01 (tentative)

4.4 Tracking open defects and resolution time

JIRA can be used to track work for the features, defects and security vulnerabilities. The open defects and security vulnerabilities with a software library version can be retrieved from the publicly available Bug database, collated as below.

Software Version	Open Issue
Python 3.6	Issue 12345: Memory leak in dict implementation
Python 3.6	Issue 12346: Crash on exit with threading
Python 3.7	Issue 22345: Incorrect behavior in asyncio
Python 3.7	Issue 22346: Race condition in subprocess module
Python 3.8	Issue 32345: Deprecation warning in urllib
Python 3.8	Issue 32346: Performance regression in json module
Python 3.9	Issue 42345: Error handling in importlib
Python 3.9	Issue 42346: Incorrect exception chaining
Python 3.10	Issue 52345: Bug in pattern matching
Python 3.10	Issue 52346: Type hinting issues with new annotations
Python 3.11	Issue 62345: Crash in new PEG parser
Python 3.11	Issue 62346: Regression in multiprocessing module
Java 8	JDK-123456: Issue with Stream API
Java 8	JDK-654321: Memory leak in lambda expressions
Java 11	JDK-112233: Performance regression in G1 GC
Java 11	JDK-332211: NullPointerException in Optional API
Java 17	JDK-445566: Incorrect behavior in Pattern Matching
Java 17	JDK-665544: Bug in Foreign Function & Memory API
.NET Core 2.1	Issue 12345: Memory leak in HttpClient
.NET Core 2.1	Issue 12346: Incorrect behavior in LINQ
.NET Core 3.1	Issue 22345: Crash on exit with async/await

.NET Core 3.1	Issue 22346: Performance regression in JSON serialization
.NET 5	Issue 32345: Deprecation warning in EF Core
.NET 5	Issue 32346: Bug in Blazor WASM
.NET 6	Issue 42345: Error handling in SignalR
.NET 6	Issue 42346: Incorrect exception in HttpClient
.NET 7	Issue 52345: Crash in new minimal API
.NET 7	Issue 52346: Bug in MAUI project
Oracle 11g	Bug 123456: Performance degradation in PL/SQL
Oracle 11g	Bug 654321: Incorrect index usage in optimizer
Oracle 12c	Bug 234567: Data corruption in RAC environment
Oracle 12c	Bug 765432: Memory leak in shared pool
Oracle 18c	Bug 345678: Issues with JSON functions
Oracle 18c	Bug 876543: Deadlock detected in ASM
Oracle 19c	Bug 456789: Problems with Data Guard sync
Oracle 19c	Bug 987654: SQL execution plan instability
Oracle 21c	Bug 567890: Issues with new JSON Data Type
Oracle 21c	Bug 098765: RAC node eviction under load

4.5 Track security vulnerabilties and resolution time

OWASP website can be referred for any critical security vulenrabilities while NVD and Snyk systems can be integrated with code repositories (Git) to identify security vulnerabilities during code build and security scan phases. This helps resolving prioritizing and tracking issue resolution.

4.6 Knowing software service level objectives pass and fail

Software performance metrics need to be tracked periodically on the runtime health check and errors if any.This helps in documenting service level objective and outages. To identify service level objective misses, continuous checks are needed on the system asset activities every week usually includes software installations, upgrades, patches, system reboots , health check monitoring etc.

4.7 Be aware of software version functionality and features

Functional features, capabilities supported by specific software version will be part of the release documentation.

4.8 Knowing the right time for software upgrade or decommission

Below is the comprehensive checklist every application support team goes through manually, to evaluate the right time for the software upgradation or replacement. This is a complex exercise that takes a lot of time and effort.

- Software performance needs to be monitored if any performance degradation after applying patches on the hardware.
- Number of defects need to be tracked for each release.
- Software Cost and additional Hardware or firmware update cost need to be calculated for each release or upgrade.
- Check IT and Business requirements needed for Digital transformation.
- Check version compatibility with other software and hardware and if there are any open defects.
- Check end of life of the software, associated hardware and operating system. If there is any end-of-life component, need to check possible options including – cost, time and effort required for replacement software. If feasible, time to upgrade. If not feasible, evaluate the risk and cost for the replacement software, setup and configure, test and deploy, then remove the old software version from the system configuration. Once this is done, it is time to decommission the old software from the System.

All the above mentioned becomes even more complex due to heterogenous input data that changes based on critical decisioning and not capturing the right System statistics. This needs systematic data analysis, hence the need of an automated systems data integration and monitoring system.

4.9 Cost Optimization with a Business Use Case:

Consider a business use case where an application is using MySQL DB 4.1 and Oracle 11g databases running on JDK 7 version. The Java 7 needs upgrade to JDK 8 due to the criticality of security vulnerabilities. There are MySQL DB compatibility issues with JDK 7 hence a decision is needed whether to upgrade MySQL DB or not. Since JDK 8 upgrade is required in this scenario, Oracle 11g also needs an upgrade to 12C, to avoid version compatibility issues.

Here comes the need of cost comparison. Below are details captured as part of this research.

Software Version	**Dependent Software Version**	**Compatible to Dependent software**	**Software Upgrade possible**
Oracle 11g	Java JDK 7	No	No
Oracle 12C	Java JDK 8	Yes	Yes
Mysqldb 4.1	Java 8, 11, 17	Yes	Yes
Mysqldb 4.1	Oracle 7	No	No

Oracle pricing reference:

https://www.oracle.com/in/autonomous-database/upgrade-standard-edition-byol/compare-tco/

MySQL pricing reference: https://www.oracle.com/in/mysql/pricing/

- **On-premises Oracle 11g database costs:**
  - (C11I) install (1 instance) =$46,398
  - (C11O) operate (renewal of 1 instance) =$11,550 per year
- **On-premises MySQL 8 database costs:**
  - MySQL median cost (1 instance) =$1425.4 x 12 =$17,104.8
- **Software Engineering Wages:**

  Let "a" be the cost incurred per year in fixing defects/security vulnerabilities
- **On-premises Oracle 12c database cost:**
  - (C12I) install i.e., upgrade from 11g to 12C=$4,350

  - (C12O) operate=\$15,000per year
- Time Horizon =5years

Step 1: Calculating Total Costs between Oracle lower and higher versions:

C11total=46,398+(11,550×5) + (ax5) =\$(1,04,148+a5)

C12total=4,350+(15,550×5) =\$(59,100)

Step 2: Compute Cost Savings

Cost Savings=C11total−C12total=\$(1,04,148+a5) − \$(59,100) > \$45,000

Step 3: Calculating Total Costs between Oracle and MySQL higher versions:

M11total = (17,104.8×5) =\$(85,524)

C12total (calculated value from Step.1) => \$(59,100) < \$(85,524)

Hence, Oracle Database upgrade to 12C is cheaper even after considering data migration from MySQL to Oracle DB.

Inference

The above details prove MySQL DB can be decommissioned after migrating data to Oracle 12C besides upgrading Oracle11g to Oracle12C.

Note: Data migration can be done with SQL Developer tool with no additional cost. Software Engineering cost to update Java code pointing to MySQL is very minimal.

Below is the relational data modeling on the input data sets.

System Asset Master Data

(surrogate keys are not defined)

Column	**Description**	**Relation**
System_asset_name	Software or hardware system component name	One to one with software or hardware system
Asset_Version	Version of the software or hardware system asset	One to one with system asset
Asset_EOL_date	This is software or hardware expiry date	One to one with system asset
Supported_platform	This denotes operating system, on-premises or cloud specification	One to one with system asset

System Asset Mapping Data

(surrogate keys are not defined)

Column	**Description**	**Relation**
System_asset_name	Software or hardware system component name	One to one with software or hardware system
Dependent_asset_name	This denotes the dependent software, hardware, or operating system (on-premises or cloud) specification	One to one with system asset

System Budget Master Data

(surrogate keys are not defined)

Column	**Description**	**Relation**
System_version	Software or hardware system component version	Many to one with software or hardware system asset
License_name	Specification of license i.e., enterprise single user, multiuser, single instance, multi instance, operating system association etc.	One to one with software or hardware system version
Planned_cost	This is software or hardware version cost when purchased, deployed	One to one with software or hardware system version
Actual_cost	This is software or hardware version cost when accrued/invoiced	One to one with software or hardware system version
Operation_cost	This is software or hardware version cost when accrued/invoiced year on year or at periodic intervals as applicable	One to one with software or hardware system version
Upgrade_cost	This is software or hardware version upgrade when accrued/invoiced year on year	One to one with software or hardware system version

	or at periodic intervals as applicable	
Decommission_cost	This is software or hardware version decommission cost when accrued/invoiced year on year or at periodic intervals as applicable	One to one with software or hardware system version

System Activity Master Data

(surrogate keys are not defined)

Column	**Description**	**Relation**
System_version	Software or hardware system component version	Many to one with software or hardware system asset
Activity_date	This is software or hardware version used	Many to one with software or hardware system version
Activity_code	Activity code description like PATCH UPDATE, RESTART/REBOOT, DOWNTIME etc.	Many to one with activity date
Activity time	Time taken for the maintenance task as mentioned in the activity code	Many to one with activity date
Upgraded_version	Version info of the system if upgraded	Many to one with activity date

Next_activity_date	Next maintenance activity date of the software or hardware version	one to one with activity date
System_asset_sla_passed	1 or 0 representing pass or failed	one to one with activity date
Additional_cost_incurred	Additional cost incurred if any software or hardware failures and replaced with other recommended software or hardware entities	one to one with activity date
Known_issue_count	This is collected from the system errors, warnings or defects encountered from previous activities, or from day-to-day operations tracker	one to one with software or hardware version

System State Metrics

(surrogate keys are not defined)

Column	**Description**	**Relation**
System_name	Software or hardware system component name	One to one with software or hardware system
system_version	This is software or hardware version used	One to many with software or hardware system
dep_sw_cnt	Software count on which a software is dependent on	One to many with software version

dep_hw_cnt	Hardware count on which a software is dependent on	One to many with software version
eol_hw_cnt	End of life hardware count associated to software	Many to one with software entity
eol_sw_cnt	End of life software count associated to hardware	Many to one with hardware entity
sw_eol_upg_cost_reqd	This is boolean flag representing if additional cost needed to upgrade the end-of-life software	Many to one with software entity
hw_eol_upg_cost_reqd	This is boolean flag representing if additional cost needed to upgrade the end-of-life hardware	Many to one with hardware entity
hw_maint_cost_reqd	This is boolean flag representing if additional cost needed to maintain/ operate the end-of-life hardware	Many to one with hardware entity
sw_maint_cost_reqd	This is boolean flag representing if additional cost needed to maintain/ operate the end-of-life hardware	Many to one with hardware entity
sw_defects_cnt	The defects count with software version used	Many to one with software version
hw_defects_cnt	The defects count with hardware version used	Many to one with hardware version

hw_min_sla	Minimum service level agreement time in milli seconds for the hardware availability (up and running)	One to one with hardware entity
sw_min_sla	Minimum service level agreement time in milli seconds for the software availability (up and running)	One to one with software entity
hw_upg_recommend	Boolean flag to represent if hardware upgrade needed	One to one with hardware entity
hw_decom_recommend	Boolean flag to represent if hardware decommission is needed	One to one with hardware entity
sw_upg_recommend	Boolean flag to represent if software upgrade needed	One to one with software entity
sw_decom_recommend	Boolean flag to represent if software decommission is needed	One to one with software entity

Application and Results of Machine Learning Models applied on this research data sets.

How K-NN Classifier Works

K-Nearest Neighbors (K-NN) is a **supervised learning algorithm** used mainly for classification tasks. It works on the principle that data points close to each other are likely to belong to the same category or class. The algorithm classifies a new data point based on the **"k" closest labeled data points** in the dataset.

Key Steps of the K-NN Algorithm:

1. **Choose the value of K**:
   - Select the number of neighbors (k). This is typically an odd number to avoid ties, and the performance of the classifier heavily depends on this parameter.
2. **Compute the distance**:
   - For each data point in the training set, calculate the distance between the new data point and the existing data points using a distance metric like **Euclidean distance** or **Manhattan distance**.
3. **Find the K-nearest neighbors**:
   - After calculating the distances, sort the data points and pick the k closest ones to the new data point.
4. **Assign the label**:
   - Look at the most common class or category among the k nearest neighbors. The class with the most votes becomes the prediction for the new data point.
5. **Classify the data point**:
   - The algorithm assigns the new data point to the class based on the majority vote from its neighbors.

Example of K-NN in Action:

To apply, we have a dataset of software with features like current version, risk. We want to classify a new software version based on these features. The data contains two classes: Upgrade and Decommission.

Dataset Example:

Software	**Risk**	**Next Version**	**Label**
Oracle 10g	Medium	11	Upgrade
Oracle 9i	High		Decommission
.NET 4.1	High		Decommission
.NET 4.5	Medium	4.8	Upgrade
JDK 11	Low	17	Upgrade

K-NN Process:

1. We choose **k = 3**, meaning we will consider the 3 nearest software.
2. Calculate the **Euclidean distance** between the new software version and each software version in the dataset.
3. Identify the 2 nearest neighbours based on distance.
4. Let's say the nearest neighbours include two upgrade and one decommission
5. Based on the majority (2 votes for upgrade, 1 vote for decommission), we classify the new software version as upgrade.

In summary, K-NN classifies data points by looking at the most common class among its nearest neighbours, making it an intuitive, easy-to-understand algorithm for classification tasks. However, it can be computationally expensive with large datasets.

How Decision Trees Work

A **Decision Tree** is a **supervised machine learning algorithm** used for both classification and regression tasks. It uses a tree-like model of decisions, where data is split recursively based on certain conditions. The aim is to break down a dataset into smaller and smaller subsets while incrementally developing a tree structure with decision nodes and leaf nodes.

- **Decision Nodes**: These represent conditions on a feature.
- **Leaf Nodes**: These represent the final output or decision (the class label or value).

Each internal node of the tree corresponds to a **feature** of the dataset, and each branch represents a **decision rule**. The final leaf node represents the **outcome** or **class label**.

Key Concepts:

1. **Root Node**: This is the first decision node, representing the best feature to split the data on.
2. **Splitting**: Dividing the dataset into subsets based on the values of a feature.
3. **Decision Rule**: At each decision node, a rule is applied (like "Is next software version is compatible to current dependent software version?").
4. **Leaf Node**: After repeated splits, the tree will eventually reach a point where further splitting doesn't add value. This is the final output (classification or regression value).

The model makes decisions by traversing the tree from the **root** to the **leaf** node, following the conditions laid out in the decision nodes.

Example: Decision Tree for Classification

Let's say we have a dataset to classify software upgrade based on its version, end of life date, compatibility with next version of dependent software.

Dataset Example:

Software Version	**Dependent Software Version**	**Compatible to Dependent software**	**Software Upgrade?**
Oracle 7	Java JDK 8	No	No
Oracle 8	Java JDK 11	Yes	Yes

Software Version	**Dependent Software Version**	**Compatible to Dependent software**	**Software Upgrade?**
Mysqldb 4.1	Windows 11	Yes	Yes
Mysqldb 4.1	Windows 10	No	No

Decision Tree Process:

1. **Select the Root Node**:
    - The algorithm evaluates each feature (e.g., version, dependent version, compatibility) to find the **best split**. It typically uses metrics like **Gini Impurity** or **Information Gain** (based on entropy) to measure how well a feature splits the data.
    - For example, if "software version" splits the data into more homogeneous groups (i.e., "Software Upgrade?" is mostly the same for any other minor version following major version), and “Software version” will be chosen as the root node.
2. **Split the Data**:
    - The dataset is divided into subsets based on the root node’s decision.
    - Example split based on **Software Version**:
        - If Version is major i.e., 4.0, then subset is software of this version and any other minor versions i.e., 4.x, can be upgraded.
        - If Version is not compatible i.e., 4.0, then subset is software needs replacement and decommission.
3. **Continue Splitting**:
    - For each subset, the algorithm continues to split based on the remaining features (like "other dependent version" or "dependent software compatibility").
    - Example split based on **other software compatibility**:
        - If other software version compatibility= Yes, then split further.

        - If other software version compatibility = No, split further or decide on replacement with decommissioning the software.

4. **Reach the Leaf Nodes**:
    - Once further splitting does not add any significant benefit or the subset is pure (i.e., all members of the subset have the same outcome), the process stops.
    - Leaf nodes now represent whether a software is upgradeable or ready for replacement followed by decommissioning.

Example Decision Tree Structure:

- **Root**:
    - **Is Oracle DB ≤ 9?**
        - Yes:
            - **Is dependent software Java JDK >= 8?**
                - Yes: Upgrade (Leaf Node = Yes)
                - No: Decommission (Leaf Node = No)
        - No:
            - **Is dependent software Java JDK <= 8?**
                - No: Decommission (Leaf Node = Yes)
                - Yes: Upgrade (Leaf Node = No)

Key Considerations:

- **Overfitting**: Decision trees can be prone to overfitting, especially with deep trees that perfectly classify the training data but perform poorly on unseen data. Pruning methods are often used to combat this.
- **Pruning**: This is a technique used to reduce the complexity of the tree by trimming parts of the tree that have little importance.

Summary:

A Decision Tree classifies data by splitting it into branches based on feature values. The tree grows until it reaches a decision at the leaf nodes. The goal is to create a tree that makes accurate predictions by finding the best splits at each node

How Support Vector Machine (SVM) Works

Support Vector Machine (SVM) is a **supervised learning algorithm** primarily used for **classification**, but it can also be used for **regression** tasks. The goal of an SVM is to find the best boundary or **hyperplane** that separates the data points of different classes as distinctly as possible.

SVM is based on the concept of finding a **hyperplane** that maximally separates two classes of data points. The best hyperplane is the one that leaves the maximum margin between the data points of both classes. This boundary is referred to as the **maximum margin hyperplane**.

Key Concepts:

1. **Hyperplane**:
   - A hyperplane in an n-dimensional space (n being the number of features) is a decision boundary that separates data points. For example, in a 2D space, the hyperplane is a line, while in a 3D space, it's a plane.
2. **Support Vectors**:
   - These are the data points that lie closest to the hyperplane. They are crucial because the position of the hyperplane is determined by them. The goal of SVM is to maximize the margin between the hyperplane and these support vectors.
3. **Margin**:
   - The margin is the distance between the hyperplane and the nearest data points from each class. The larger the margin, the better the classifier generalizes to unseen data.
4. **Linear vs. Non-Linear SVM**:
   - If the data is **linearly separable**, SVM finds a linear hyperplane that divides the two classes.
   - If the data is **non-linearly separable**, SVM uses a technique called the **kernel trick** to map the data into a higher-dimensional space where a hyperplane can separate the classes.

Example of SVM for Classification:

Let's consider a binary classification task where we want to classify **software upgrade** and **software decommissions** based on their **version and compatibility information.**

Dataset Example:

Software Version	**Dependent Software Version**	**Compatible to Dependent software**	**Software Upgrade?**
Oracle 7	Java JDK 8	No	No
Oracle 8	Java JDK 11	Yes	Yes
Mysqldb 4.1	Windows 11	Yes	Yes
Mysqldb 4.1	Windows 10	No	No

SVM Process:

1. **Find the Hyperplane**:
   - In a 2D space (features: software version and dependent software version), the SVM tries to find a **line** (hyperplane) that separates software upgrade need from software decommission need.
2. **Identify Support Vectors**:
   - The closest software upgrades and decommissions to the separating line are identified as **support vectors**. These data points are critical because they define the margin.
3. **Maximize the Margin**:
   - SVM maximizes the distance (margin) between the hyperplane and the support vectors. The larger the margin, the better the classification.
4. **Classification**:
   - Once the hyperplane is determined, new software can be classified based on their position relative to this boundary.

Example Visualization:

If you plot **software version** on the x-axis and **dependent software version** on the y-axis, the SVM will draw a line that best separates the software upgrade needed from the software decommission needed, ensuring that the nearest upgrade versions and decommission versions (support vectors) are as far away from the line as possible. The equation of this line is the **hyperplane**.

If Data is Linearly Separable:

- SVM draws a **straight line** (hyperplane) that separates the upgradeable versions and decommission versions perfectly.

If Data is Non-Linearly Separable:

- Sometimes, it's impossible to separate the data points with a straight line. In such cases, SVM uses a **kernel function** (like the **RBF kernel**) to map the data into a higher-dimensional space where it becomes linearly separable. This allows the SVM to classify more complex datasets.

Example: Non-Linearly Separable Data

Imagine that the software versions in the dataset have common features i.e., overlapping features, and a straight line cannot perfectly separate apples and oranges. In this case, SVM uses the **kernel trick** to transform the data into a higher dimension. In this new dimension, it becomes possible to draw a linear hyperplane that separates the two classes.

Kernel Functions:

- **Linear Kernel**: Used when the data is linearly separable.
- **Polynomial Kernel**: Maps the data to a higher degree polynomial space.
- **Radial Basis Function (RBF) Kernel**: Used for non-linearly separable data. It creates decision boundaries in complex, curved shapes.

Summary:

- **Support Vector Machines (SVM)** classify data by finding the hyperplane that best separates the classes with the **maximum margin** between the nearest data points (support vectors) of each class.
- If the data is not linearly separable, SVM uses the **kernel trick** to map the data to a higher-dimensional space where it can be separated by a hyperplane.

SVM is widely used in applications such as image classification, bioinformatics, and text categorization, where clear decision boundaries are required.

How Linear Regression Works

Linear Regression is a **supervised learning algorithm** primarily used for **regression** tasks. The goal of linear regression is to model the relationship between a **dependent variable** (also called the response or target variable) and one or more **independent**

variables (also known as predictors or features). This relationship is represented by a straight line that best fits the data points.

In simple terms, linear regression predicts the value of the target variable based on the values of the predictors, assuming a linear relationship between them.

Types of Linear Regression:

1. **Simple Linear Regression**: There is only one independent variable (predictor).
2. **Multiple Linear Regression**: There are two or more independent variables (predictors).

The Linear Equation:

For **simple linear regression**, the relationship between the dependent variable (Y) and the independent variable (X) is expressed as: Y=β0+β1X+ϵY = \beta_0 + \beta_1X + \epsilonY=β0+β1X+ϵ

- YYY = Dependent variable (predicted outcome)
- XXX = Independent variable (predictor)
- β0\beta_0β0 = Y-intercept (the value of Y when X = 0)
- β1\beta_1β1 = Slope (rate of change in Y for a unit change in X)
- ϵ\epsilonϵ = Error term (captures the deviation from the true relationship)

In **multiple linear regression**, the equation becomes: Y=β0+β1X1+β2X2+...+βnXn+ϵY = \beta_0 + \beta_1X_1 + \beta_2X_2 + ... + \beta_nX_n + \epsilonY=β0+β1X1+β2X2+...+βnXn+ϵ where X1,X2,...,XnX_1, X_2, ..., X_nX1,X2,...,Xn are the multiple predictors.

Key Steps in Linear Regression:

1. **Fit a Line**:
   - Linear regression finds the line (in simple linear regression) or hyperplane (in multiple linear regression) that best fits the data points by minimizing the error.
2. **Calculate the Best Fit**:
   - The method used to find the best-fitting line is called **Ordinary Least Squares (OLS)**, which minimizes the sum of the squared differences between the actual and predicted values.
3. **Prediction**:
   - Once the model is trained, the linear regression equation can be used to make predictions for new data points.

Example of Simple Linear Regression:

Here we are predicting the **software upgrade based on** risk of the software version used.

Dataset Example:

Software	**Current Version**	**Risk**	**Target Version**
Oracle Database	10g	High	12
Java JDK	8	High	11
Java JDK	11	Low	17
.NET	4.1	High	4.8

Linear Regression Process:

1. **Visualize the Data**:
   - Plot the data points on a graph, where the **x-axis** represents the **current version,** and the **y-axis** represents the **target version**.
   - The data points might look like a scatter plot.
2. **Fit a Line to the Data**:
   - The algorithm will fit a line to the data points such that the error between the **actual target version** and the **predicted target version** is minimized.
3. **Make Predictions**:
   - Using the fitted line, we can now predict the target version of software for any given software.

Residuals (Error Terms):

The difference between the target version and the predicted target version for each data point is called a **residual**. Linear regression aims to minimize the sum of the squared residuals.

Key Assumptions of Linear Regression:

1. **Linearity**: The relationship between the dependent and independent variables is linear.
2. **Homoscedasticity**: The variance of residuals (errors) is constant across all values of the independent variables.
3. **Independence**: Observations are independent of each other.

4. **Normality of Residuals**: The residuals (errors) are normally distributed.

Example Visualization:

Imagine a plot with **software version** on the **x-axis** and **target versions** on the **y-axis**. The data points form a scatter plot, and the linear regression model fits a straight line through these points. This line represents the predicted price for each square footage.

Multiple Linear Regression Example:

Let's say we extend the software target version prediction model by including more predictors, such as:

- **Dependent software version**
- **Dependent software version compatibility**
- **Operating cost**

In this case, we use **multiple linear regression**. The equation might look like this: Target Version cost=operating cost × number of target software version - operating cost x number of dependent software version

Here, the coefficients represent the impact of each feature on the price:

- For each target software version chosen, the operating cost may increase or decrease when compared to dependent software version cost
- For each current software version, the overall operating cost of current software may increase or decrease when compared to dependent software version cost.

Summary:

- **Linear Regression** models the relationship between dependent and independent variables using a straight line.
- In **simple linear regression**, there is one independent variable, while in **multiple linear regression**, there are multiple predictors.
- The algorithm fits the best line to the data points by minimizing the error and then uses this line to make predictions for unseen data.

Linear regression is widely used in fields such as finance, economics, real estate, and engineering, where there is a need to predict continuous outcomes based on input variables.

How Time Series Prediction Works

Time series prediction is a type of supervised learning where the model analyzes past data points, which are recorded over time, to predict future values. In a time series, data is sequential, meaning the order of data points matters, unlike in typical regression or classification tasks where the order of data points is irrelevant.

Time series prediction can be applied in various fields such as finance (stock prices), weather forecasting, sales forecasting, and more. The key goal is to understand the underlying patterns, trends, and seasonality in the data to make accurate predictions.

Key Components of Time Series Data:

1. **Trend**: The long-term upward or downward movement in the data.
2. **Seasonality**: Regular patterns that repeat over a fixed period, such as daily, monthly, or yearly cycles.
3. **Noise**: Random fluctuations in the data that don't follow any identifiable pattern.
4. **Cyclic Patterns**: Irregular, non-fixed cycles influenced by external factors, such as economic cycles.

Time Series Models:

Several models are commonly used for time series forecasting, including:

1. **ARIMA (Auto Regressive Integrated Moving Average)**
2. **Exponential Smoothing (ETS)**
3. **Long Short-Term Memory (LSTM)** Networks
4. **Facebook Prophet**

Each of these models has its own strengths and applications, depending on the characteristics of the data.

1. ARIMA Model (Auto Regressive Integrated Moving Average):

ARIMA is one of the most popular models for time series forecasting, particularly for short-term predictions. It combines three components:

- **AutoRegression (AR)**: A model that uses the dependency between an observation and a number of previous observations.

- **Integrated (I)**: Differencing the raw observations (subtracting an observation from the previous observation) to make the time series stationary (i.e., having constant mean and variance).
- **Moving Average (MA)**: A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Example of ARIMA for Time Series Forecasting:

Let's say you have a dataset that contains **count of critical issue pending resolution for a given software** over the last 2-3 years (considered every month), and you want to forecast software version defects for the next couple of quarters.

Dataset Example:

Year and Month	Defects pending resolution
Jan 2022	100
Feb 2022	120
Mar 2022	130
...	...
Dec 2024	200

The steps for forecasting with ARIMA are as follows:

1. **Make the Series Stationary**:
    - Check if the data has a trend and make it stationary by removing the trend or applying differencing.

For example, if the defects pending resolution are increasing over time, you might apply a differencing step: Differenced DefectCount=DefectCount (time) −DefectCount (time-1)

2. **Identify AR, I, and MA Terms**:
    - Use tools like the **ACF (Auto Correlation Function)** and **PACF (Partial Auto Correlation Function)** plots to identify the lag terms for AR and MA.
    - ARIMA (p, d, q) represents the order of AR, I, and MA terms, where:
        - **p**: Number of lag observations for the AR term.
        - **d**: Degree of differencing applied to make the series stationary.

        - **q**: Number of lagged forecast errors in the prediction equation for the MA term.

3. **Fit the ARIMA Model**:
    - Train the ARIMA model using historical sales data.
4. **Make Predictions**:
    - Once the model is trained, use it to forecast future defect pending resolution count for the next couple of quarters

For example, ARIMA might predict the defect count which are pending resolution for January 2025 to be around 210 based on the past trends and patterns.

2. Exponential Smoothing (ETS):

Exponential Smoothing models apply smoothing coefficients to past data points to predict future values. The more recent the data, the higher the weight assigned to it. ETS models are good for data with trends and seasonality.

Example:

Using **Holt-Winters Exponential Smoothing**, you can model the trend and seasonality in sales data.

1. **Level**: The current state of the series.
2. **Trend**: The slope (growth or decline) of the series over time.
3. **Seasonality**: Cyclical patterns repeating every year, quarter, month, etc.

3. Long Short-Term Memory (LSTM) Neural Networks:

LSTM is a type of **recurrent neural network (RNN)** specifically designed for sequence data like time series. Unlike ARIMA or ETS, which are classical statistical models, LSTM is a deep learning model.

- **How it Works**: LSTM networks maintain a "memory" of past observations and learn long-term dependencies. They are capable of learning both short- and long-term patterns in the data.

Example of LSTM for Time Series Prediction:

Let’s take **monthly software patch activity data and its service level objective (SLO)** data over the last 10 years, and you want to compare the expected SLO and actual SLO for the next couple of months i.e., finding missed SLO in the software version patch activity. The reasons in SLO miss could be errors during patching, more system restarts than expected and firmware errors etc.

Steps:

1. **Preprocess the Data**: Split the time series data into smaller sequences (e.g., the last 3 months) and predict the next month SLO miss values.
2. **Train the LSTM Model**: Feed these sequences into the LSTM network. The LSTM learns both short-term and long-term dependencies in the data.
3. **Predict Future Values**: Use the trained LSTM model to predict the next couple of months SLO misses.

4. Facebook Prophet:

Prophet is an open-source forecasting tool developed by Facebook. It is specifically designed for time series data that has strong seasonality and trends. It is easy to use and can automatically detect seasonal patterns, holidays, and special events that may affect predictions.

Example of Prophet for Time Series Prediction:

Let's consider forecasting **software version defects**. With time, if software is not upgraded, that software version will have more security vulnerabilities with dependent software defects and compatibility issues.

Steps:

1. **Model Setup**: Prophet requires a data frame with two columns: the date (ds) and the value to forecast (y), which is unresolved software version defects in this case.

Ds	**y**
2022-01-01	1000
2022-02-01	1500
...	...
2024-11-01	3000

2. **Fit the Prophet Model**: Train Prophet to capture the trend and monthly defect count (unresolved) with specific software version.
3. **Predict Future Traffic**: After training, Prophet predicts software defects for the next few months, considering the data trend.

Summary:

- **Time series prediction** involves analysing sequential data to forecast future values based on past observations.

- Classical models like **ARIMA** and **ETS** are effective for short-term, trend-based, and seasonal data.
- **LSTM** networks excel at capturing long-term dependencies in more complex, non-linear data.
- **Prophet** is a user-friendly, robust tool that handles trends, seasonality, and special events.

Time series forecasting is widely used in banking, finance, economics, business, and environmental science to predict sales, stock prices, weather and more.

Software quality data – fault predictions

The comparison below outlines the strengths and weaknesses of each model when applied to software fault prediction:

Model	**Use in Software Fault Prediction**	**Strengths**	**Weaknesses**	**Best Use Case in Software Fault Prediction**
Time Series Models	Time series models like **ARIMA** are useful when software faults are highly dependent on time, e.g., predicting software failures at specific stages of the software life cycle (e.g., testing phase).	- Captures time-dependent trends and software versions - Effective for forecasting based on historical data	- Requires data with clear time-based patterns - Not suited for non-sequential data	Predicting the frequency of bugs or defects during software releases over time.
K-Nearest Neighbours (K-NN)	**K-NN** can be used to predict software faults based on the similarity of new code to previous code segments. It looks for similar patterns in software metrics like complexity, code churn, etc.	- Simple, interpretable - non-parametric - Good for small datasets - No training phase required	- Computationally expensive with large datasets - Sensitive to irrelevant or noisy features	When you have non-time-dependent software metrics and want to predict faults based on historical code churn.

Model	**Use in Software Fault Prediction**	**Strengths**	**Weaknesses**	**Best Use Case in Software Fault Prediction**
Decision Trees	**Decision trees** classify software components based on features like version dependency, complexity, open defects, learning decision rules to predict whether a software module is prone to failure.	- Easy to interpret - Handles non-linear relationships - Works with categorical and continuous data	- Prone to overfitting - Sensitive to data variation and noise	When the relationships between features and faults are non-linear, and explainability is important.
Linear Regression	**Linear regression** models are used to predict software faults as a linear combination of software metrics such as length of the code, code changes to resolve defects, and dependency with other software.	- Simple and interpretable - Effective for linear relationships	- Assumes a linear relationship - Not suitable for non-linear data - Can be influenced by outliers	When the relationship between software metrics and faults is expected to be linear, such as predicting fault density.
Support Vector Machines (SVM)	**SVM** classifies software modules as fault-prone or non-fault-prone by finding the optimal hyperplane that separates the two classes based on software metrics (e.g., code complexity, code churn, other software dependencies).	- Good for both linear and non-linear data - Effective with high-dimensional data - Robust to outliers	- Computationally expensive - Difficult to interpret - Requires careful tuning of hyperparameters	When the relationship between features and software faults is complex, non-linear, or when the dataset is high-dimensional.

1. Time Series Models:

- **How They Work**: Time series models like **ARIMA** or **LSTM** capture patterns and trends over time. In software fault prediction, they can model fault occurrences over time (e.g., during development or testing phases).
- **Use Case**: Useful in software systems with time-based patterns of failures, such as periodic software updates or recurring defects.

Example: Predicting when future software faults will occur based on the history of faults during previous testing cycles.

2. K-Nearest Neighbours (K-NN):

- **How It Works**: K-NN is a lazy learning method that makes predictions by finding the most similar past software components based on metrics like complexity, past bug reports, etc.
- **Use Case**: Useful when fault prediction is based on the similarity between current and past software metrics.

Example: Predicting whether a new software module will have defects based on how similar it is to past modules with known fault statuses.

3. Decision Trees:

- **How They Work**: Decision trees split data based on feature values, learning rules that classify software modules as fault-prone or non-fault-prone.
- **Use Case**: Good for explaining which software metrics (e.g., complexity, lines of code, previous bug density) contribute to fault prediction.

Example: A decision tree may learn that "modules with more than 500 lines of code and high complexity are prone to failure," making it easy to interpret and act on.

4. Linear Regression:

- **How It Works**: Linear regression fits a linear equation to predict the number of software faults based on input features like code complexity or bug history.
- **Use Case**: Suitable when the relationship between software metrics and faults is expected to be linear.

Example: Predicting the number of faults in a software module as a function of lines of code, where an increase in the number of lines increases the fault probability linearly.

5. Support Vector Machines (SVM):

- **How It Works**: SVM finds the optimal hyperplane that separates fault-prone and non-fault-prone software components based on features.
- **Use Case**: Effective when the relationship between metrics and faults is non-linear, or when there are many features (high-dimensional data).

Example: Predicting whether a software module will have faults based on a combination of features like code complexity and past defect history, even if the data has non-linear patterns.

CHAPTER VI: SUMMARY, IMPLICATIONS, AND RECOMMENDATIONS

5.1 Summary

This research identified the data required for decision making in software upgrades or decommissioning process, proposes integrating all such data on a weekly basis so this can be loaded as relational data sets. The data sets are analyzed on a weekly basis applying Data Mining techniques such as association, classification and prediction by training and testing with Machine Learning models such as K-NN Classifier, Decision Trees, Support Vector Machine, Linear Regression and Time Series prediction. This research proposes all this as software upgrades and decommission life cycle which is a continuous process that keeps generating recommendations in a timely manner for stakeholders to review and take appropriate decisions proactively.

ML Model	Definition	Example Software/Library used
K-NN Classifier	K-Nearest Neighbors (K-NN) is a supervised learning algorithm used for classification tasks. It classifies new data points based on their proximity to existing labeled data points.	**Scikit-learn**: Python library that includes K-NN as part of its neighbors module (sklearn.neighbors.KNeighborsClassifier)

Decision Trees	Decision Trees are supervised learning models that split data into branches to reach a decision. The decision-making is structured in a tree format, where each node represents a feature, and each branch represents a decision or rule.	**Scikit-learn**: Python's tree.DecisionTreeClassifier is used for classification, and DecisionTreeRegressor for regression tasks.
Support Vector Machine (SVM)	SVM is a supervised learning model used for both classification and regression tasks. It finds a hyperplane in an N-dimensional space that distinctly classifies the data points.	**LibSVM**: Integrated in various platforms like Scikit-learn (sklearn.svm.SVC for classification, SVR for regression)
Linear Regression	Linear Regression is a supervised learning technique that models the relationship between a dependent variable and one or more independent variables using a linear approach.	**Scikit-learn**: Python library offering sklearn. linear_model.LinearRegression for performing linear regression tasks.
Time Series Prediction	Time series prediction refers to analysing time-dependent data to predict future points. Common techniques include ARIMA, LSTM, and Prophet for forecasting trends based on past values.	**Facebook Prophet**: Open-source forecasting tool for time series data. **Statsmodels**: Python package that supports ARIMA and other time series models.

Conclusion:

- **Time Series Models** are best for predicting faults when there are clear, time-based patterns in software failure history.
- **K-NN** is simple but effective when the dataset is small, and predictions are based on similarity to past software components.
- **Decision Trees** provide interpretable rules for fault classification but are prone to overfitting.
- **Linear Regression** is suited to fault prediction when the relationship between faults and software metrics is linear.
- **SVM** is highly effective in complex, high-dimensional, or non-linear software fault prediction tasks.

In general, the choice of the model depends on the nature of the software fault data (time-based, linear/non-linear, feature-rich, etc.) and the interpretability required by stakeholders.

5.2 Implications

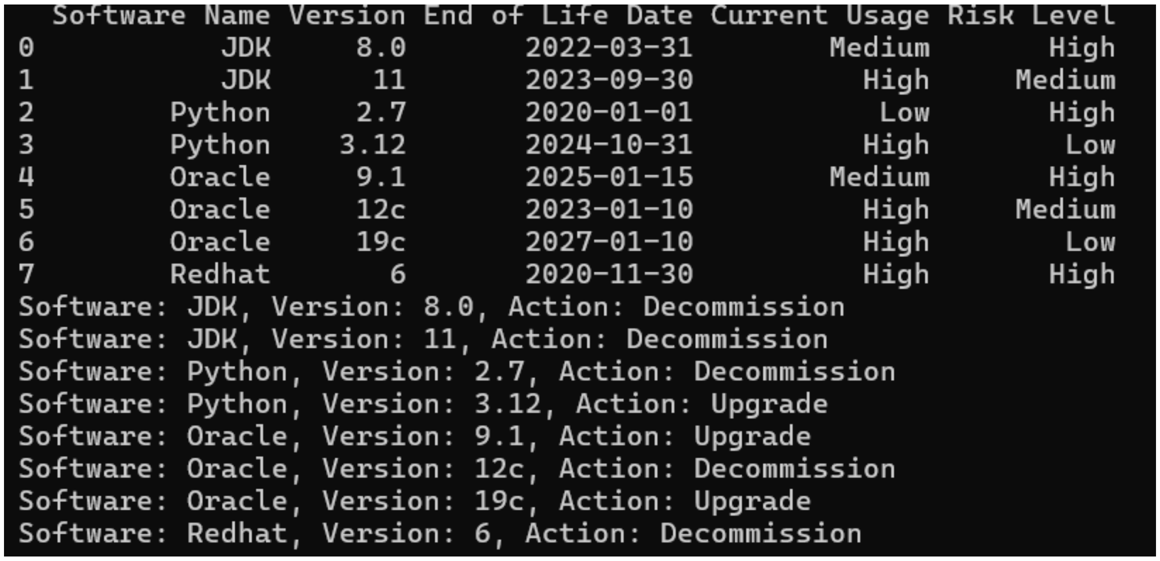
Time, Effort and Complexity in the decision making of software upgrades and decommissions is one common challenge faced by Organizations in the Information Technology industry today. The proposed findings, idea and prototype solution as part of this research can be adopted to address the problem statement and can be enhanced for any further use cases. This research work can provide great foundational framework in building a software as a service product with automation of use cases.

5.3 Recommendations for Future Research

With evolution and adoption of Generative Artificial Intelligence, Organizations may identify new data sets in terms of gathering evidence documentation on successful software upgrades, root cause analysis on version upgrade issues, decommissions time and effort etc. This research can be expanded further for building recommendation systems for such use cases supporting private Large Language Models which provides capabilities such as Chat bots on the Organizational data considering data privacy. The overall idea and concept remain the same.

5.4 Conclusion

The prototype solution is built in Python programming language which is capable of data integrations and executing machine learning models. The proposed result of process execution is depicted below, showing recommended action on a given software/library.



```
  Software Name Version End of Life Date Current Usage Risk Level
0           JDK     8.0       2022-03-31        Medium       High
1           JDK      11       2023-09-30          High     Medium
2        Python     2.7       2020-01-01           Low       High
3        Python    3.12       2024-10-31          High        Low
4        Oracle     9.1       2025-01-15        Medium       High
5        Oracle     12c       2023-01-10          High     Medium
6        Oracle     19c       2027-01-10          High        Low
7        Redhat       6       2020-11-30          High       High
Software: JDK, Version: 8.0, Action: Decommission
Software: JDK, Version: 11, Action: Decommission
Software: Python, Version: 2.7, Action: Decommission
Software: Python, Version: 3.12, Action: Upgrade
Software: Oracle, Version: 9.1, Action: Upgrade
Software: Oracle, Version: 12c, Action: Decommission
Software: Oracle, Version: 19c, Action: Upgrade
Software: Redhat, Version: 6, Action: Decommission
```

While experimenting, the recommendation percentage produced by Support Vector Machine, Random Forest and Decision Tree performed better out of all the models, giving the lowest testing MSE (Mean Squared Error) value.

REFERENCES

Bachwani R., et al (2012) 'Recommendation system for software upgrades', *ResearchGate publication* [online]. Available at https://www.researchgate.net/publication/234128761_Mojave_A_Recommendation_System_for_Software_Upgrades

Iqbal M., Khalid M. and Khan M.N.A. (2013) 'A Distinctive Suite of Performance Metrics for Software Design', *ResearchGate publication* [online]. Available at https://www.researchgate.net/publication/270526905_A_Distinctive_Suite_of_Performance_Metrics_for_Software_Design

Kumar K. and Kaur K. (2022) 'Recommendation of Regression Techniques for Software Maintainability Prediction with Multi-Criteria Decision-Making', *ResearchGate publication* [online]. Available at https://www.researchgate.net/publication/362917558_Recommendation_of_regression_techniques_for_software_maintainability_prediction_with_multi-criteria_decision_making

Nouh F. (2016) 'SAM Software Asset Management', *ResearchGate publication* [online]. Available at https://www.researchgate.net/publication/291818013_SAM_Software_Asset_Management

Ortiz-Ochoa M. (2016) 'Identifying and Prioritizing Modernization of Legacy Systems', *ResearchGate publication* [online]. Available at https://www.researchgate.net/publication/294874894_A_Practical_Approach_to_Identifying_and_Prioritizing_Modernization_of_Legacy_Systems

Pombriant D. (2021) 'Do you have the right software for your digital transformation', *Harvard Business Review, 2021(8).* Available at https://hbr.org/2021/08/do-you-have-the-right-software-for-your-digital-transformation

Saarela M., et al (2017) 'Measuring Software Security from the Design of Software', *ResearchGate publication* [online]. Available at https://www.researchgate.net/publication/321140241_Measuring_Software_Security_from_the_Design_of_Software

Singh M. and Chabbra J.K. (2021) ‘Software Fault Prediction Using Machine Learning Models and Comparative Analysis’, *ResearchGate publication* [online]. Available at https://www.researchgate.net/publication/350490953_Software_Fault_Prediction_Using_Machine_Learning_Models_and_Comparative_Analysis

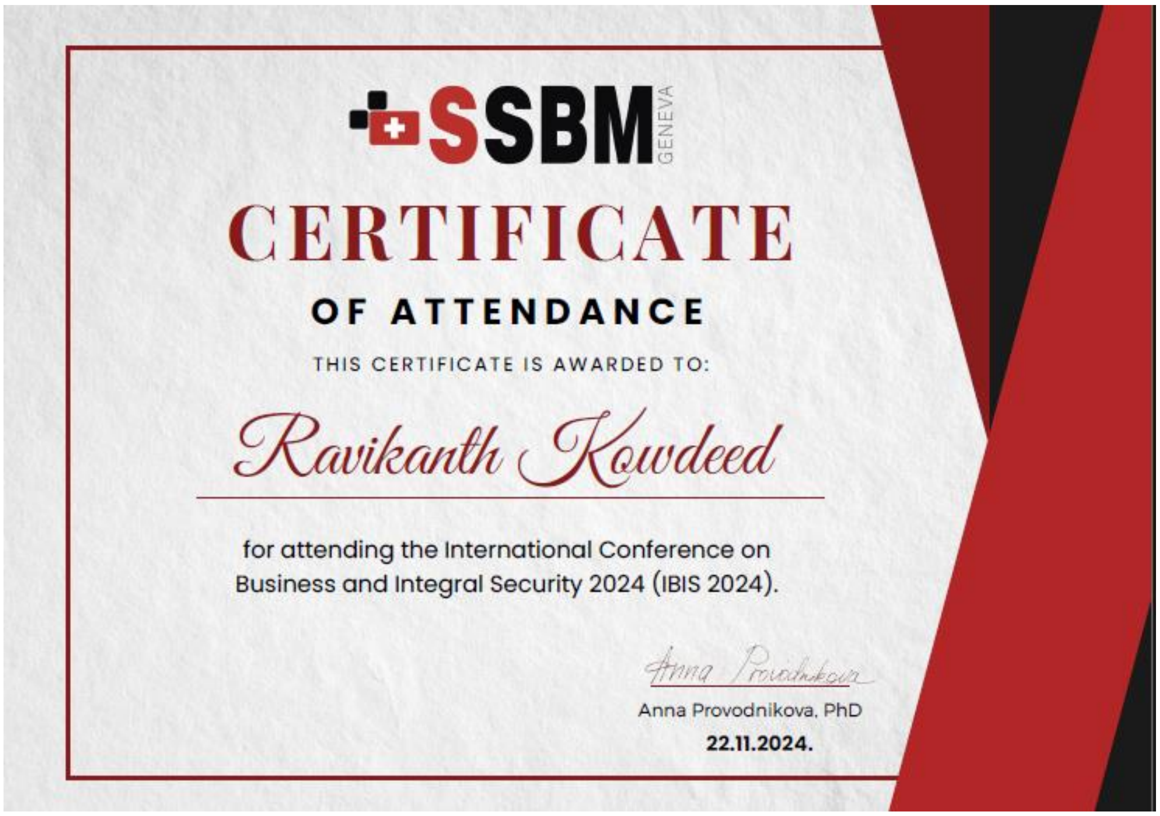
Zijden S.V.D. (2022) ‘Three key tasks needed to decommission applications’, *Computer Weekly article*, [online]. Available at https://www.computerweekly.com/opinion/Gartner-Three-key-tasks-needed-to-decommission-applications .

APPENDIX

As part of this research, one paper is submitted for GBIS 2024 and below papers are published in the international journals – IOSR and IIJSRT, links below. Also, a patent has been filed.

1. Below paper was submitted for GBIS 2024

GENERATING RECOMMENDATION INSIGHTS AS PART OF THE SOFTWARE UPGRADE AND DECOMMISSIONS LIFE CYCLE IN INFORMATION TECHNOLOGY INDUSTRY



SSBM GENEVA

CERTIFICATE

OF ATTENDANCE

THIS CERTIFICATE IS AWARDED TO:

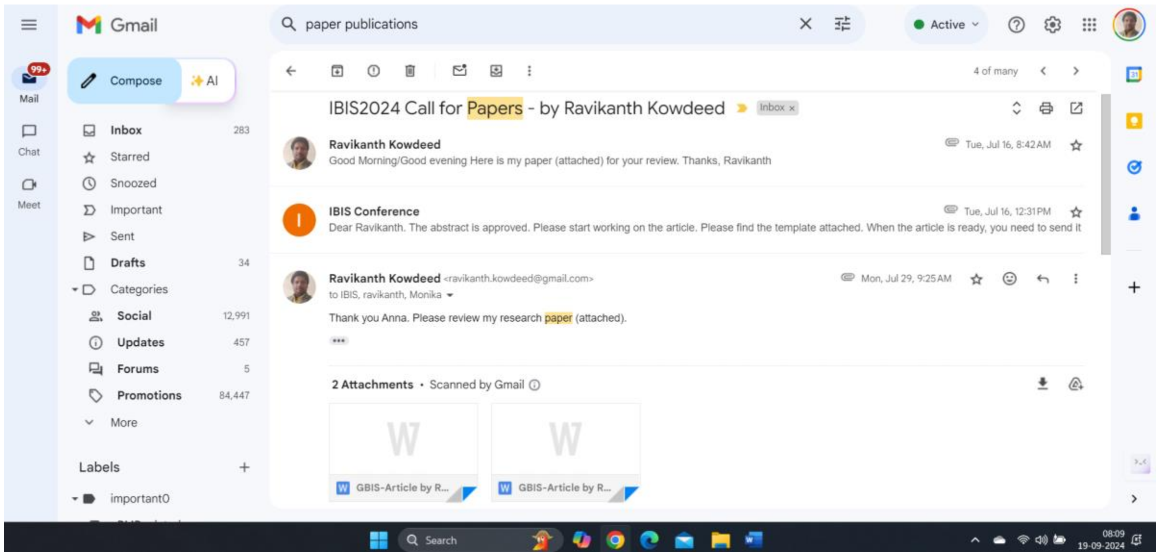
Ravikanth Kowdeed

for attending the International Conference on Business and Integral Security 2024 (IBIS 2024).

Anna Provodnikova

Anna Provodnikova, PhD

22.11.2024.



IBIS2024 Call for Papers - by Ravikanth Kowdeed

Ravikanth Kowdeed — Tue, Jul 16, 8:42 AM
Good Morning/Good evening Here is my paper (attached) for your review. Thanks, Ravikanth

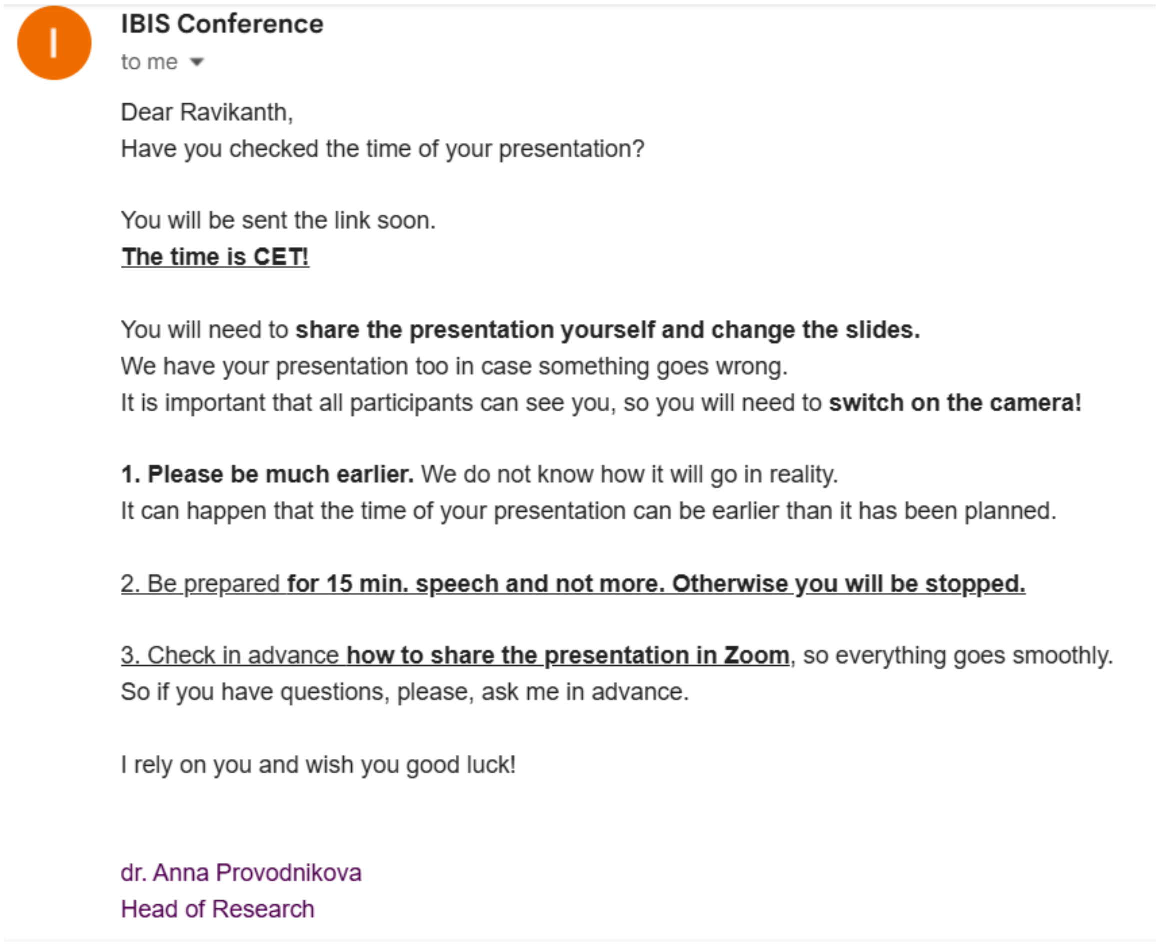
IBIS Conference — Tue, Jul 16, 12:31 PM
Dear Ravikanth. The abstract is approved. Please start working on the article. Please find the template attached. When the article is ready, you need to send it

Ravikanth Kowdeed <ravikanth.kowdeed@gmail.com> — Mon, Jul 29, 9:25 AM
to IBIS, ravikanth, Monika

Thank you Anna. Please review my research paper (attached).

2 Attachments • Scanned by Gmail

GBIS-Article by R...
GBIS-Article by R...



IBIS Conference
to me

Dear Ravikanth,
Have you checked the time of your presentation?

You will be sent the link soon.
The time is CET!

You will need to **share the presentation yourself and change the slides.**
We have your presentation too in case something goes wrong.
It is important that all participants can see you, so you will need to **switch on the camera!**

1. Please be much earlier. We do not know how it will go in reality.
It can happen that the time of your presentation can be earlier than it has been planned.

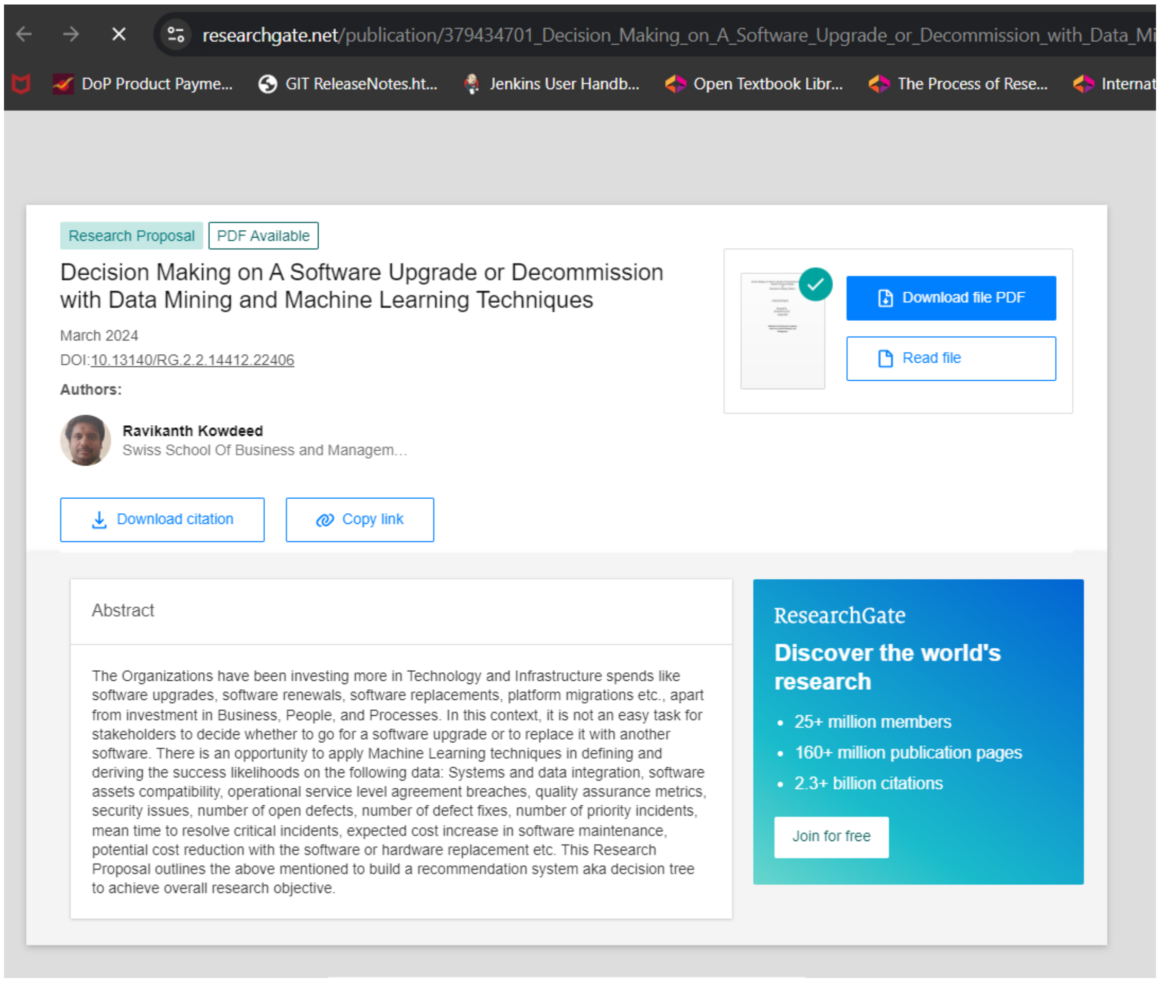
2. Be prepared **for 15 min. speech and not more. Otherwise you will be stopped.**

3. Check in advance **how to share the presentation in Zoom**, so everything goes smoothly.
So if you have questions, please, ask me in advance.

I rely on you and wish you good luck!

dr. Anna Provodnikova
Head of Research

2. Papers published:
(a)
https://www.iosrjournals.org/iosr-jce/pages/26(2)Series-3.html
(b)
https://www.ijisrt.com/decision-making-on-a-software-upgrade-or-decommission-with-data-mining-and-machine-learning-techniques-in-information-technology-industry



Decision Making on A Software Upgrade or Decommission with Data Mining and Machine Learning Techniques

March 2024

DOI:10.13140/RG.2.2.14412.22406

Authors:

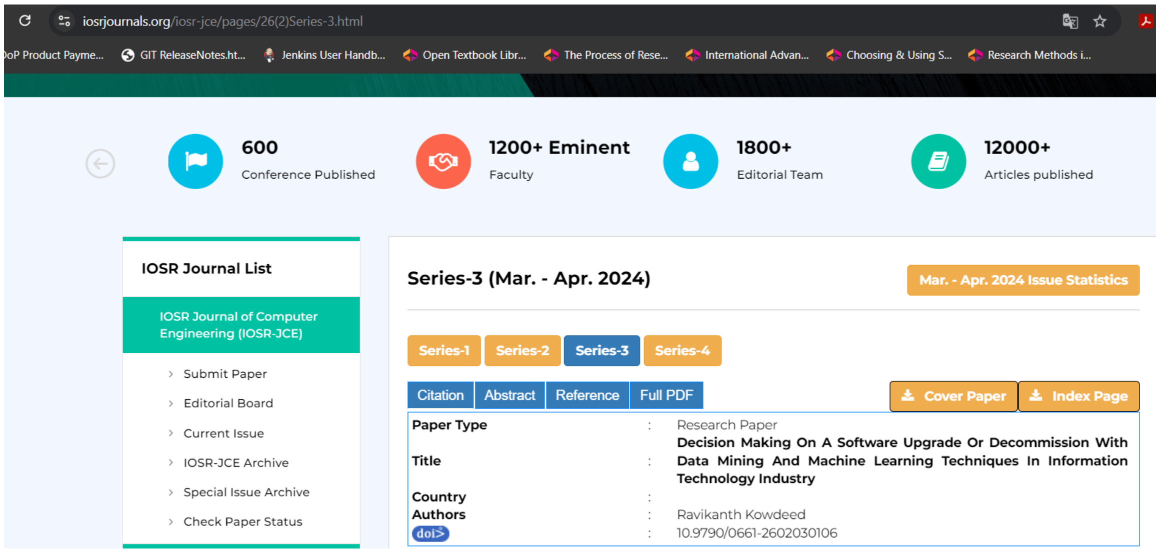
Ravikanth Kowdeed
Swiss School Of Business and Managem...

Abstract

The Organizations have been investing more in Technology and Infrastructure spends like software upgrades, software renewals, software replacements, platform migrations etc., apart from investment in Business, People, and Processes. In this context, it is not an easy task for stakeholders to decide whether to go for a software upgrade or to replace it with another software. There is an opportunity to apply Machine Learning techniques in defining and deriving the success likelihoods on the following data: Systems and data integration, software assets compatibility, operational service level agreement breaches, quality assurance metrics, security issues, number of open defects, number of defect fixes, number of priority incidents, mean time to resolve critical incidents, expected cost increase in software maintenance, potential cost reduction with the software or hardware replacement etc. This Research Proposal outlines the above mentioned to build a recommendation system aka decision tree to achieve overall research objective.

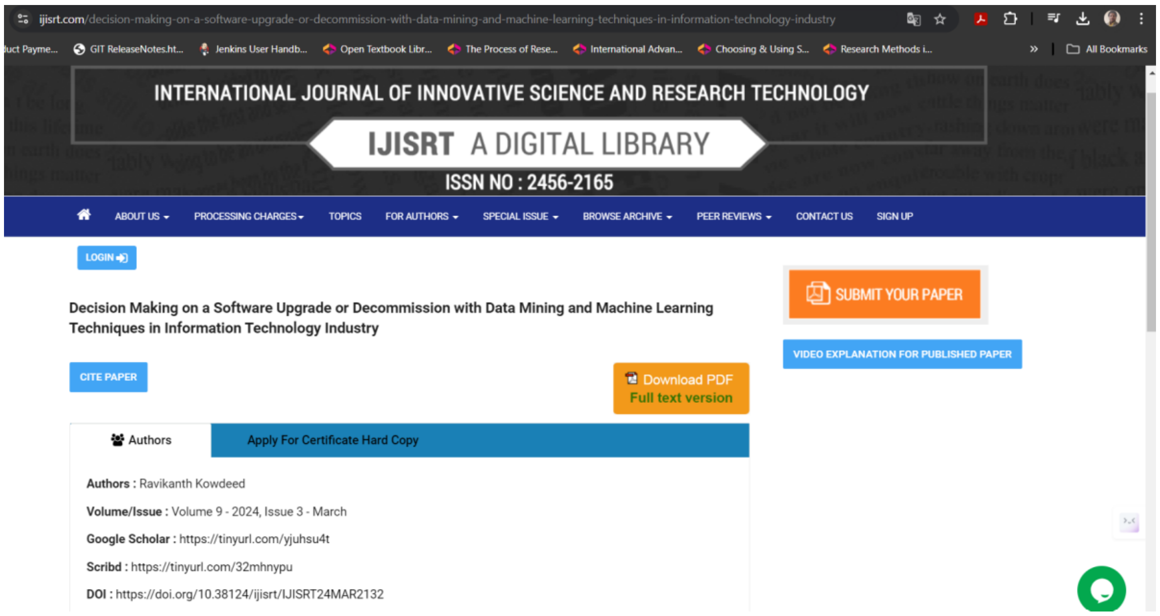
ResearchGate
Discover the world's research
- 25+ million members
- 160+ million publication pages
- 2.3+ billion citations

Join for free



Series-3 (Mar. - Apr. 2024)

Paper Type	:	Research Paper
Title	:	Decision Making On A Software Upgrade Or Decommission With Data Mining And Machine Learning Techniques In Information Technology Industry
Country	:	
Authors	:	Ravikanth Kowdeed
doi>	:	10.9790/0661-2602030106


ijisrt.com/decision-making-on-a-software-upgrade-or-decommission-with-data-mining-and-machine-learning-techniques-in-information-technology-industry
INTERNATIONAL JOURNAL OF INNOVATIVE SCIENCE AND RESEARCH TECHNOLOGY
IJISRT A DIGITAL LIBRARY
ISSN NO : 2456-2165
ABOUT US
PROCESSING CHARGES
TOPICS
FOR AUTHORS
SPECIAL ISSUE
BROWSE ARCHIVE
PEER REVIEWS
CONTACT US
SIGN UP
LOGIN
SUBMIT YOUR PAPER
VIDEO EXPLANATION FOR PUBLISHED PAPER
CITE PAPER
Download PDF Full text version
Authors
Apply For Certificate Hard Copy

Decision Making on a Software Upgrade or Decommission with Data Mining and Machine Learning Techniques in Information Technology Industry

Authors : Ravikanth Kowdeed

Volume/Issue : Volume 9 - 2024, Issue 3 - March

Google Scholar : https://tinyurl.com/yjuhsu4t

Scribd : https://tinyurl.com/32mhnypu

DOI : https://doi.org/10.38124/ijisrt/IJISRT24MAR2132

3. Patent details:

- Patent Application number: 202441042345
- Invention title: A recommendation tool for software upgradation or decommission and its method thereof
- Type of application: Complete application
- Complete application filing date: 31st May 2024

FORM 2

The Patent Act 1970

(39 of 1970)

&

The Patent Rules, 2005

(See Section 10 and Rule 13)

COMPLETE SPECIFICATION

TITLE OF THE INVENTION

A recommendation tool for software upgradation or decommission and its method thereof

Name and address of the applicant:

Name:	Ravikanth Kowdeed
Nationality:	Indian

Address: 4-48/23, Lane 2, Pratap nagar, Kismatpur, Rajendra Nagar, Hyderabad, Telangana, Pin code: 500086

PREAMBLE OF THE INVENTION

The following complete specification particularly describes the invention and the way it is performed:

FIELD OF INVENTION

[001] The present invention relates to a recommendation tool that recommends a software upgradation or decommission that is both data and model driven.

BACKGROUND

[002] Many organizations are investing more in Technology and Infrastructure that includes software upgrades, software renewals, software replacements, platform migrations and so on apart from its further investment in Business, People, and Processes. In this context, it is not an easy task for stakeholders to decide whether to go for a software upgrade or to replace it with another software.

[003] The patent document **US10585773** discloses a method to manage economics and operational dynamics of various information technology (IT) systems wherein a computer collects data indicative of operation of a plurality of hardware components and collects data indicative of operation of a plurality of software components. The computer creates a first qualitative value representing a hardware status of the plurality of the hardware components and a second qualitative value representing a software status of the plurality of the software components. The first and second qualitative values are displayed in graphical form for evaluation by a system operator, and the computer computes a probability of life expectancy for the plurality of hardware components and the plurality of software components based on said first and second qualitative values and utilizing cognitive and artificial intelligence-based calculations to determine the probability.

[004] The document **IN202341044600** describes a 5G network life cycle management system using machine learning techniques has a data service module

(200), a data supervision module (201), a management module (202) and a cloud server (203). The data service module (200) connected with 5G network (204) for obtaining service data. The data supervision module (201) configured with the data service module (200) for monitoring 5G network data. The management module (202) for managing a plurality of business data comprising of hospital, financial, school and office business data and the cloud server (203) comprising of an Supplier Lifecycle (SLC) manager module (205), a cloud orchestrator module (206), and an Software Defined Networking (SDN) controller module (207) managing 5G network life cycle and generate a graph for sending to user device.

[005] However, in the aforementioned documents, recommendation to upgrade or decommission a software by collecting data and training machine learning models is not disclosed. There is no unified approach or solution to consolidate data and relationships of Information Technology Assets, Software Upgrades, Software costs, Software defects, Software Performance Metrics, Security issues, IT system versions, service level objectives etc. Due to this, the decision making of software upgrades and software decommissioning is a tedious process and takes more time and effort.

[006] Therefore, there is a need to build a solution that can integrate and validate the collected data from different variables for fault prediction using Data Mining and Machine Learning techniques and recommend for software upgradation and decommission.

OBJECT OF THE INVENTION

[007] The principal object of the invention is to provide a recommendation tool for software upgradation or decommission using data mining and machine learning models.

[008] Another object of the invention is to provide various components of the tool comprising an user interface, an application interface and a database wherein the user interface provides navigation pages that depicts visual representation of the options selected by the user and outputs that are generated by the application interface, and the database stores all the reports of the application interface.

[009] Another object of the invention is to disclose the variables that are used by the application interface including software version, hardware version, end of life for software, end of life for hardware, additional costs to upgrade and/or maintain associated software and hardware, respective defect density and availability of minimum service level agreement time for hardware and software, respectively.

[0010] Another object of the invention is to utilize time series analysis by decision tree and SVTM to train the collected data for a decision making process by the application interface.

[0011] Another object of the invention is to provide a process of working of the recommendation tool including the steps of data collection by the input module, machine learning model selection, train and evaluate the machine learning models by the processing module, data analysis by data mining and machine learning models, automated recommendation, iterative improvements, generation of reports and storing in the database.

[0012] These and other objects and characteristics of the present invention will become apparent from the further disclosure to be made in the detailed description given below.

SUMMARY OF THE INVENTION

[0013] This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

[0014] The invention discloses a recommendation tool that recommends for software upgradation or decommission using data mining and machine learning models and its method thereof.

[0015] It is one aspect of the present invention to disclose various components of the tool comprising an user interface, an application interface and a database wherein the user interface provides visual representation of the reports and outputs that are generated by the application interface, and the database stores all the reports of the application interface.

[0016] It is another aspect of the present invention to disclose the variables that are used to collect data by the application interface including software version, hardware version, end of life for software, end of life for hardware, additional costs to upgrade and/or maintain associated software and hardware, respective defect density and availability of minimum service level agreement time for hardware and software, respectively.

[0017] It is other aspect of the present invention to utilize time series model and decision tree to train the collected data for a decision making process in the application interface.

[0018] It is yet other aspect of the present invention to provide a process of working of the recommendation tool including the steps of data collection by the input module, machine learning model selection, train and evaluate the machine learning models by the processing module, data analysis by data mining and machine learning models, automated recommendation, iterative improvements, generation of reports and storing in the database.

[0019] These together with other objects of the invention, along with the various features of novelty which characterize the invention, are pointed out with particularity in the disclosure. For a better understanding of the invention, its operating advantages and the specific objects attained by its uses, reference should be had to the accompanying drawings and descriptive matter in which there are illustrated preferred embodiments of the invention.

BRIEF DESCRIPTION OF DRAWINGS

[0020] The foregoing and other features of embodiments will become more apparent from the following detailed description of embodiments when read in conjunction with the accompanying drawings. In the drawings, like reference numerals refer to like elements.

[0021] In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the embodiments of the invention. It is apparent, however, to one skilled in the art that the embodiments of the invention may be practiced without these specific details or with an equivalent arrangement. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the embodiments of the invention.

[0022] **FIG. 1** (in this document) illustrates a block diagram that represents all the essential components of the recommendation tool.

[0023] **FIG. 2** (in this document) illustrates a process of automated recommendation for software upgrades or decommission by the recommendation tool.

DETAILED DESCRIPTION OF INVENTION

[0024] The embodiments herein and the various features and advantageous details thereof are explained more fully with reference to the non-limiting embodiments that are illustrated in the accompanying drawings and / or detailed in the following description. Descriptions of well-known components and processing techniques are omitted to not unnecessarily obscure the embodiments herein. The examples used herein are intended merely to facilitate an understanding of ways in which the embodiments herein may be practised and to further enable those of skill in the art to practice the embodiments herein. Accordingly, the examples should not be construed as limiting the scope of the embodiments herein.

[0025] Reference in this specification to “one embodiment” or “an embodiment” means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present disclosure. The appearance of the phrase “in an embodiment” in various places in the specification are not necessarily all referring to the same embodiment, nor are separate or alternative embodiments mutually exclusive of other embodiments. Moreover, various features are described which may be exhibited by some embodiments and not by others. Similarly, various requirements are described which may be requirements for some embodiments but not for other embodiments.

[0026] Moreover, although the following description contains many specifics for the purposes of illustration, anyone skilled in the art will appreciate that many variations and/or alterations to said details are within the scope of the present disclosure. Similarly, although many of the features of the present disclosure are described in terms of each other, or in conjunction with each other, one skilled in the art will appreciate that many of these features can be provided independently of other features. Accordingly, this description of the present disclosure is set forth without any loss of generality to, and without imposing limitations upon the present disclosure.

[0027] The term “recommendation tool” or “tool” are interchangeably used in the Specification where all these terms have same context and meaning as it describes a recommendation tool for software upgradation or decommission of the invention.

[0028] The invention disclosed herein provides a recommendation tool (100) for software upgradation or decommission using data mining and machine learning models.

[0029] **FIG. 1** illustrates a block diagram that represents all the essential components of the recommendation tool (100) including an user interface (101), an application interface (102) and a database (103).

[0030] In an embodiment, the user interface (101) is accessed through a login page wherein each restricted user is provided with a unique login id and password. After logging in the tool, the user interface is configured to display all the available options for the user. The options include a home page, a dashboard page, inventory, hardware, software, licenses, reports, technology spend and so on. Technically, the user interface provides visual representation of the reports and outputs that are generated by the application interface.

[0031] The dashboard provides a consolidated data on the total annual spend on the software and hardware tools, apps and devices, total number of managed applications, total number of managed licences and details of underutilized accounts, application usage by individual users including active and inactive users, and divisional spends by individual departments. It also provides details on the upcoming renewal of software and hardware assets for the next one year.

[0032] The managed applications include a list of all software and hardware application, their respective annual spend, licenses details, assigned users and active users of each applications and their respective departments. This also includes a filter option to shortlist the details. On selecting a specific application, the page navigates to another page which represents activity status of the selected application by all the users in the organization. It also provides integrated status of each application with that of other applications. The activity status includes date and time at which each users used the specific application.

[0033] The inventory option in the user interface aids the user to initiate discovery of all software and hardware tools, apps and devices when it is logged in by the designated user. The technology spend option provides purchase and spend details of all the existing software and hardware tools that is used in any organization or industry. The details include the list of vendors and their respective cost analysis including purchase and its respective maintenance.

[0034] Similarly, the hardware, software, and licenses navigation pages also provides consolidated details of the existing hardware, software apps and tools with their respective licenses. All the navigation pages in the user interface are integrated with the application interface of the recommendation tool such that different types of reports are generated according to the user’s requirement. The reports are exported in either csv or xlsx for documentation purpose.

[0035] In another embodiments, the application interface (102) includes an input module (104), a processing module (105) and an output module (106). The input module (104) further comprises data collection tier and pre-processing tier that collects and pre-processes data from sinks, data APIs and files including csv, pdfs and documents, and ETL connectors. The processing module (105) analyses and processes the data collected by the input module (104) using data mining and machine learning models.

[0036] The data collection tier in the input module (104) collects data from the public web sites, software release documentation, organization case studies and feedback surveys conducted in the communities of practice and communities of technology interest groups that is related to software requirements, software budget, system asset metadata, software issues, and execution metrics whereas the pre-process tier ensures consistency, completeness, and accuracy that involve cleaning, formatting, and standardizing the data.

[0037] The ETL connectors in the input module (104) extract and integrate data from different systems including System logs through Platform level files or shared data stores, Defects, Cost considerations, Ethics or Security or Data Privacy dictionaries, Business needs, and through APIs and load the integrated data to the database.

[0038] After data collection by the input module (104), data analysis is performed by the processing module (105). The machine learning models in the processing module (105) performs time series analysis using decision tree and support vector models; sentimental analysis using XGBoost; anomaly detection using isolation forest time series and sequential data analysis using long short term memory (LSTM). The processing module (105) fuses data from different sources and analyses using machine learning models to identify patterns, trends, and anomalies.

[0039] Based on the insights gathered from data analysis, the processing module (105) generates automated recommendations for software upgrades or decommission that are presented in the output module. The output module (106) is integrated with the user interface (101) to provide output based on the options selected by the user in the form of both visual representation and recommendation reports.

[0040] The database (103) in the recommendation tool (100) serves two purposes including gathering the input data from the input module (104) to generate the desired data needed for machine learning models in the processing module (105), and store all the gathered data and the automated recommendation for software upgrades or decommission.

[0041] In yet another embodiment, the process of automated recommendation for software upgrades or decommission by the recommendation tool (100) is disclosed as shown in **FIG 2**. The steps in the process of automated recommendation for software upgrades or decommission by the recommendation tool (100) includes:

(a) Data collection: The input module (104) gathers weekly data from various source of data by the ETL connectors including software requirements, software budget, system asset metadata, software issues, and execution metrics and consolidates the collected metrics every month wherein

the data on software requirements includes functional, non-functional and domain requirements;

the data on software budget includes component version of the software, specification of license, cost of software version when purchased or deployed and accrued or invoiced, upgrade and/or decommission cost at periodic intervals;

the data on system asset metadata on premises or cloud includes component name, version and expiry date of the software system, its respective supporting operating system, and its dependent hardware assets;

the data on software issues includes data collected from the system errors, warnings or defects encountered from previous activities, or from day-to-day operations tracker; and

the data on execution metrics includes component name and version of the software and its associated hardware, count and its respective end of life of each software and its associated hardware, additional cost to upgrade or maintain the end of life software and its associated hardware, and minimum service level agreement time in milli seconds for the software availability and its associated hardware;

(b) Machine learning Model selection: The models that are selected in the processing module (105) include decision tree and support vector models for time series analysis, XGBoost for sentimental analysis, long short term memory (LSTM) for sequential data analysis and isolation forest time series for anomaly detection;

(c) Train and evaluation of the models in the processing module: The input data is split into training and validation sets to evaluate model performance wherein the training set is used as input to train the selected machine learning models and the validation set is used to assess their performance in predicting software requirements, costs, asset metadata, issues, and execution metrics. Metrics such as accuracy, precision, recall, and F1-score are measured depending on the specific tasks and objectives.

(d) Data analysis by data mining and machine learning models: The data analysis by the processing module (105) includes predictive maintenance analysis, cost benefit analysis and user feedback integrations;

wherein the predictive maintenance analysis anticipate potential issues and performance degradation in the software system by analysing historical data on software execution metrics and system asset metadata such that the trained models in the processing module (105) can predict when software components are likely to reach end-of-life or experience compatibility issues with other software or hardware;

wherein the cost and benefit analysis evaluate the potential costs and benefits of proposed software updates by considering factors such as installation costs, renewal fees, upgrade expenses, and decommissioning costs such that the trained models in the processing module recommend updates that provide the greatest value to the organization while minimizing operational expenses.

wherein the models in the processing module (105) integrates user feedback into the update recommendation process, allowing stakeholders to provide input and prioritize features or fixes based on user preferences and pain points, and ensures that updates are tailored to meet the needs of end-users and improve overall satisfaction with the software.

(e) Automated recommendation: Based on the insights gathered from data analysis, the processing module (105) generates automated recommendations for software upgrades or decommission by taking into account factors such as business objectives, system needs, hardware specifications, and cloud vs non-cloud infrastructure support, ensuring that updates are aligned with organizational goals and technical requirements.

(f) Iterative improvement: The machine learning models in the recommendation tool (!00) continuously learns and adapts based on feedback and performance metrics by iteratively refining its models and algorithms such that the tool (100) can improve the accuracy and relevance of its recommendations over time, leading to more effective software upgrade or decommission decisions.

(g) Generation of report: The automated recommendations for software upgrades or decommission are generated as reports by the processing module (105) and is sent to the output module (106) that is further represented in the user interface (101).

(h) Storing of data: The gathered input data from the input module (104) and the recommendations by the processing module (105) are stored in the database (103) for future references.

[0042] The upgrade success is determined using all the above input data and monthly data mining by applying The Pareto principle wherein 80 percent of the output from a given situation or system is determined by 20 percent of the input. Based on this principle, the recommendation on typical distribution and likelihood of software upgrade or decommission is decided by the recommendation tool.

[0043] In an example embodiment, Let us take the mysqldb library used in the Python program behind an online application saving orders to MySQL database. Input data collected on a weekly basis for the recommendation process includes system performance metrics, connectivity errors in the logs, result of windows server updates, result of .net patch updates, security defects/vulnerabilities, MySQL database patches, python updates, application downtime, SLA misses if any and so on. The Output by the recommendation tool is a recommendation report with data insights including software, software version, platform, platform version, recommended for upgrade (y/n), and recommended for decommission (y/n) as shown in Table 1 below:

Table 1 Recommendation output for a specific software using the tool

Software	Software version	Platform	Platform version	Recommended for upgrade (y/n)	Recommended for decommission (y/n)
mysqldb lib	4.1	Windows	11	Y	N
mysql client	4.0	Windows	11	Y	N

[0044] Industrial applicability: By implementing the software upgrade or decommission recommendation tool disclosed herein, organizations can streamline the update process, reduce manual effort, and make more informed decisions based on data-driven insights. This concept represents a novel approach to software maintenance and enhancement, leveraging machine learning to optimize the software development lifecycle.

[0045] The present software upgrade or decommission recommendation tool can also be applied for:

(a) Executive reporting systems can use the above process as part of Data Analytics.

(b) Applications like Cyber Security Audit reports and Privately setup GenAI systems can be benefitted.

(c) The tool has potential to be built as SaaS product and can be used by IT Organizations users, particularly Executive body members

[0046] The foregoing description of the specific embodiments will so fully reveal the general nature of the embodiments herein that others can, by applying current knowledge, readily modify and/or adapt for various applications such specific embodiments without departing from the generic concept, and, therefore, such adaptations and modifications should and are intended to be comprehended within the meaning and range of equivalents of the disclosed embodiments. It is to be understood that the phraseology or terminology employed herein is for the purpose of description and not of limitation. Therefore, while the embodiments herein have been described in terms of preferred embodiments, those skilled in the art will recognize that the embodiments herein can be practiced with modification within the spirit and scope of the embodiments as described herein.

Claims:

I claim:

1. A recommendation tool (100) for software upgradation or decommission, comprising:

an user interface (101),

an application interface (102), and

a database (103);

wherein the user interface (101) is configured to display all the available options for the user;

wherein the application interface (102) includes an input module (104) comprising data collection tier and pre-processing tier that collects and pre-processes data from sinks, data APIs and files including csv, pdfs and documents, and ETL connectors, a processing module (105) containing machine learning models to process the collected data and an output module (106) that is integrated with the user interface (101) to provide output based on the options selected by the user in the form of both visual representation and recommendation reports;

wherein the ETL connectors extract and integrate data from different data source and load the integrated data to the database (103); and

wherein the database (103) gathers the input data from the input module (104) to generate the desired data needed for machine learning models in the processing module (105), and store all the gathered data and the automated recommendation for software upgrades or decommission.

2. The recommendation tool for software upgradation or decommission *as claimed in claim 1, wherein the machine learning models include decision tree, support vector machine model, XGBoost, LSTM and isolation forest time series.*

3. The recommendation tool for software upgradation or decommission *as claimed in claim 1, wherein the user interface (101) is accessed by the registered users by logging in with their respective credentials.*

4. The process of automated recommendation for software upgrades or decommission by the recommendation tool (100) comprising the steps of:

(a) Data collection wherein the input module (104) gathers weekly data from various source of data by the ETL connectors including software requirements, software budget, system asset metadata, software issues, and execution metrics and consolidates the collected metrics every month wherein the data on software requirements includes functional, non-functional and domain requirements;

the data on software budget includes component version of the software, specification of license, cost of software version when purchased or deployed and accrued or invoiced, upgrade and/or decommission cost at periodic intervals;

the data on system asset metadata on premises or cloud includes component name, version and expiry date of the software system, its respective supporting operating system, and its dependent hardware assets; the data on software issues includes data collected from the system errors, warnings or defects encountered from previous activities, or from day-to-day operations tracker; and the data on execution metrics includes component name and version of the software and its associated hardware, count and its respective end of life of each software and its associated hardware, additional cost to upgrade or maintain the end-of-life software and its associated hardware, and minimum service level agreement time in milli seconds for the software availability and its associated hardware; (b) Machine learning Model selection wherein the models that are selected in the processing module (105) include decision tree and support vector models for time series analysis, XGBoost for sentimental analysis, isolation forest time series for anomaly detection and long short-term memory (LSTM) for sequential data analysis;

(c) Train and evaluation of the models in the processing module wherein the input data is split into training and validation sets to evaluate model performance, wherein the training set is used as input to train the selected machine learning models and the validation set is used to assess their performance in predicting software requirements, costs, asset metadata, issues, and execution metrics, wherein metrics such as accuracy, precision, recall, and F1-score are measured depending on the specific tasks and objectives.

(d) Data analysis by data mining and machine learning models wherein the data analysis by the processing module (105) includes predictive maintenance analysis, cost benefit analysis and user feedback integrations; wherein the predictive maintenance analysis anticipates potential issues and performance degradation in the software system by analysing historical data on software execution metrics and system asset metadata such that the trained models in the processing module (105) can predict when software components are likely to reach end-of-life or experience compatibility issues with other software or hardware;

wherein the cost and benefit analysis evaluate the potential costs and benefits of proposed software updates by considering factors such as installation costs, renewal fees, upgrade expenses, and decommissioning costs such that the trained models in the processing module recommend updates that provide the greatest value to the organization while minimizing operational expenses.

wherein the models in the processing module (105) integrates user feedback into the update recommendation process, allowing stakeholders to provide input and prioritize features or fixes based on user preferences and pain points, and ensures that updates are tailored to meet the needs of end-users and improve overall satisfaction with the software.

(e) Automated recommendation based on the insights gathered from data analysis, wherein the processing module (105) generates automated recommendations for software upgrades or decommission by considering factors such as business objectives, system needs, hardware specifications, and cloud vs non-cloud infrastructure support, ensuring that updates are aligned with organizational goals and technical requirements.

(f) Iterative improvement wherein the machine learning models in the recommendation tool (100) continuously learns and adapts based on feedback and performance metrics by iteratively refining its models and algorithms such that the tool (100) can improve the accuracy and relevance of its recommendations over time, leading to more effective software upgrade or decommission decisions.

(g) Generation of report wherein the automated recommendations for software upgrades or decommission are generated as reports by the processing module (105) and is sent to the output module (106) that is further represented in the user interface (101).

(h) Storing of data wherein the gathered input data from the input module (104) and the recommendations by the processing module (105) are stored in the database (103) for future references.

[0047] The invention provides an automated recommendation tool (100) for software upgradation or decommission comprising an user interface (101), an application interface (102), and a database (103) wherein *the* application interface (102) includes an input module (104) comprising data collection tier and pre-processing tier along with ETL connectors, a processing module (105) containing machine learning models to be trained and process the collected data and an output module (106) that is integrated with the user interface (101) to provide recommendation output. A process of automated recommendation for software upgrades or decommission by the recommendation tool (100) comprises the steps of data collection by the input module (104), machine learning model selection, training and evaluation of the machine learning models by the processing module (105), data analysis by data mining and machine learning models, automated recommendation, iterative improvements, generation of reports and storing in the database (103).

(*published in* **FIG. 1** in this document)

Below is the description of the **Software Asset Life Cycle** along with information on products and their features in today's market involves elaborating on the life cycle stages, key concepts, and various tools available.

1. Introduction to Software Asset Life Cycle

- **Definition** of software asset management (SAM) and software asset life cycle.
- **Importance of managing software assets** in organizations.
- **Overview of stages** in the software asset life cycle.

2. Stages of Software Asset Life Cycle

2.1 Planning and Procurement

- **Description**: Planning involves identifying the need for software, understanding requirements, budgeting, and establishing policies around software acquisition.
- **Key Products**:
  - **Flexera**: Helps optimize software procurement and ensure compliance.
  - **ServiceNow IT Asset Management**: Manages software purchases and ensures optimized usage.
- **Features**: License tracking, budget control, vendor management.

2.2 Acquisition

- **Description**: Acquiring the software through purchase, subscription, or licensing. Understanding licensing agreements and models.
- **Key Products**:
  - **Snow Software**: Automates license acquisition and helps track usage rights.
  - **Ivanti Asset Manager**: Manages software purchases and entitlements.
- **Features**: Contract management, vendor negotiation support, usage tracking.

2.3 Deployment and Installation

- **Description**: The process of installing and deploying software on various devices or servers. Ensuring that it is correctly installed according to licenses.
- **Key Products**:
    - **Microsoft SCCM (System Center Configuration Manager)**: Helps deploy software across large enterprises.
    - **ManageEngine Desktop Central**: Provides remote software installation and patching.
- **Features**: Remote installation, automation of software deployment, patch management.

3. Use and Management of Software Assets

3.1 Usage Monitoring and Optimization

- **Description**: Tracking how the software is being used to ensure it is in line with license agreements and to identify underutilized assets.
- **Key Products**:
    - **Flexera One**: Tracks software usage and helps optimize usage based on real needs.
    - **Certero for Cloud**: Monitors SaaS and cloud-based software usage.
- **Features**: License usage monitoring, software optimization recommendations, real-time insights.

3.2 Maintenance and Updates

- **Description**: Keeping the software up to date with patches, upgrades, and security updates.
- **Key Products**:
    - **Kaseya VSA**: Manages patching and updates for both on-premises and cloud-based software.
    - **Patch My PC**: A simple tool for automating updates.
- **Features**: Automatic patching, vulnerability scanning, update notifications.

4. Auditing and Compliance

- **Description**: Ensuring that the organization complies with the licensing agreements and preparing for software audits.
- **Key Products**:

    - **Snow License Manager**: Tracks license usage and ensures compliance.
    - **Flexera Compliance Manager**: Helps organizations stay compliant with software licensing.
- **Features**: License compliance reporting, audit readiness, automated license reconciliation.

5. Retirement and Disposal

- **Description**: When software is no longer needed or outdated, it needs to be decommissioned or retired from active use while ensuring proper documentation and compliance.
- **Key Products**:
    - **Ivanti IT Asset Management Suite**: Helps in retiring software assets and ensures proper license recovery.
    - **ServiceNow SAM**: Automates the retirement and reallocation of software assets.
- **Features**: License recovery, decommissioning automation, data wiping.

6. Challenges in Managing Software Asset Life Cycle

- **Complex Licensing**: The growing complexity of software licenses and subscriptions, such as hybrid on-premises and SaaS solutions.
- **Shadow IT**: Use of unsanctioned software by employees leading to compliance risks.
- **Cloud and SaaS Management**: Transitioning from traditional software to cloud-based and SaaS models requires different management approaches.
- **Compliance and Legal Issues**: Staying compliant with licensing terms to avoid costly penalties during audits.

7. Market Trends in Software Asset Management

- **AI and Automation in SAM**: AI-driven tools that automatically detect and optimize software assets.
- **Cloud-Based SAM Solutions**: The rise of cloud-native software asset management platforms.
- **SaaS and Hybrid Models**: Managing SaaS subscriptions alongside traditional software licenses.
- **Integration with ITSM (IT Service Management)**: SAM solutions that integrate with IT service management platforms for holistic IT governance.

8. Detailed Review of Popular Software Asset Management Tools

8.1 Flexera One

- **Overview**: A comprehensive SAM platform designed to optimize software and cloud costs while ensuring compliance.
- **Features**: License optimization, cloud cost management, audit readiness, SaaS management.

8.2 Snow Software

- **Overview**: Focuses on managing software licenses and cloud resources across a range of environments.
- **Features**: Software discovery, compliance auditing, SaaS and IaaS management, mobile app integration.

8.3 ServiceNow Software Asset Management (SAM)

- **Overview**: A powerful SAM tool integrated with the broader ServiceNow ITSM platform.
- **Features**: Software license management, software request automation, audit management, usage tracking.

8.4 Ivanti IT Asset Management

- **Overview**: Ivanti's solution integrates hardware and software asset management, offering strong automation capabilities.
- **Features**: License reconciliation, software usage monitoring, automated software retirement.

8.5 ManageEngine AssetExplorer

- **Overview**: An IT asset management tool that helps track and manage software licenses and assets throughout their life cycle.
- **Features**: License compliance tracking, asset discovery, asset retirement, contract management.

9. Case Studies and Use Cases

- Real-world examples of how organizations use SAM tools to streamline their software management, reduce costs, and ensure compliance.

10. Future of Software Asset Management

- **AI and Machine Learning**: Use of AI in predicting software usage patterns, license optimization, and compliance risks.

- **Blockchain in SAM**: The potential of blockchain technology for tracking software usage and licensing agreements in a transparent and secure way.
- **SaaS and Hybrid Licensing Models**: The rise of hybrid software usage models combining on-premises and cloud-based software.

In **Software Asset Management (SAM)**, various technologies are employed to optimize the management of software assets, ensure compliance with licensing agreements, reduce costs, and manage software usage throughout its life cycle. These technologies facilitate tracking, procurement, deployment, monitoring, auditing, and decommissioning of software in organizations.

Key Technologies in Software Asset Management

1. Software Discovery Tools

- **Technology Overview**: Software discovery tools are used to automatically scan and identify software installed on endpoints across the organization. These tools help in creating an inventory of all software assets, including their versions, licensing status, and usage.
- **Common Technologies**:
  - **Agent-Based Discovery**: Small software agents installed on endpoints collect data about installed software.
  - **Agentless Discovery**: Scans are conducted remotely to discover software without installing any agent on the endpoint.
- **Examples**:
  - **Flexera**: Provides discovery tools that identify installed software and track its usage.
  - **ServiceNow Discovery**: Integrated with its ITSM platform, ServiceNow Discovery automates the identification of software and hardware assets.

2. License Management Tools

- **Technology Overview**: License management tools track software licenses to ensure compliance with licensing agreements and optimize license usage. These tools monitor license usage across the organization, helping organizations avoid over-provisioning or under-utilization.
- **Common Technologies**:
  - **License Reconciliation Engines**: Compares actual software usage with purchased licenses to identify discrepancies.

    - **License Harvesting**: Automatically reclaims unused or underused software licenses and reallocates them to other users.
- **Examples**:
    - **Snow License Manager**: Tracks software license usage and provides compliance reports.
    - **Ivanti IT Asset Management**: Offers automated license management and license harvesting.

3. Cloud and SaaS Management

- **Technology Overview**: As organizations adopt more **cloud-based software** and **Software-as-a-Service (SaaS)** solutions, cloud and SaaS management tools become essential to track usage and costs. These tools help manage subscriptions, control shadow IT, and optimize cloud costs.
- **Common Technologies**:
    - **SaaS Subscription Management**: Monitors the number of SaaS subscriptions and tracks their usage to avoid redundant subscriptions.
    - **Cloud Cost Optimization**: Tracks the cost and usage of cloud-based software and infrastructure.
- **Examples**:
    - **Certero for Cloud**: Manages SaaS and IaaS (Infrastructure-as-a-Service) applications, providing visibility into usage and costs.
    - **Flexera One Cloud Cost Optimization**: Optimizes cloud costs by identifying underused or idle cloud resources.

4. Artificial Intelligence and Machine Learning

- **Technology Overview**: **AI and machine learning** technologies are increasingly being used in SAM to predict software usage patterns, optimize license management, and detect compliance risks. AI helps in automating routine SAM tasks like identifying under-utilized licenses or predicting the need for future software purchases.
- **Common Technologies**:
    - **Predictive Analytics**: Machine learning algorithms analyze past software usage patterns to predict future software demand.
    - **Anomaly Detection**: AI-powered anomaly detection identifies unusual or risky software usage that may violate compliance rules.
- **Examples**:

    - **Flexera AI-Powered License Optimization**: Uses AI to recommend optimized licensing strategies based on usage patterns.
    - **Snow Software**: Employs machine learning to forecast software consumption and potential license compliance issues.

5. Blockchain for Software License Management

- **Technology Overview**: **Blockchain** technology, with its immutable and transparent ledger, is emerging as a potential solution for managing software licenses. Blockchain can provide a secure and transparent record of license ownership and usage, ensuring that all parties have an accurate and tamper-proof record of software agreements.
- **Common Technologies**:
    - **Decentralized License Management**: Blockchain allows organizations to manage software licenses in a decentralized manner, reducing reliance on centralized entities for license verification.
    - **Smart Contracts**: Automatically enforce license agreements and trigger actions like renewals or deactivation when conditions are met.
- **Examples**:
    - **Chronicled**: Uses blockchain for secure software licensing management and automated compliance.
    - **IBM Blockchain**: Explores blockchain solutions for managing software licenses and ensuring compliance.

6. Automation and Orchestration Tools

- **Technology Overview**: Automation and orchestration tools help streamline and automate various SAM processes, from software deployment and patching to license tracking and retirement. Automation ensures that software is deployed, updated, and removed in a controlled and compliant manner.
- **Common Technologies**:
    - **Robotic Process Automation (RPA)**: Automates repetitive tasks like license renewals, software deployment, or report generation.
    - **IT Orchestration**: Integrates SAM tools with other IT management platforms, such as IT service management (ITSM) and configuration management databases (CMDBs).
- **Examples**:

    - **ServiceNow ITAM**: Provides workflow automation to manage software throughout its lifecycle, from procurement to retirement.
    - **Ivanti Neurons**: Offers automation capabilities for patch management, software deployment, and license management.

7. Software Metering

- **Technology Overview**: **Software metering** tools help organizations track the usage of specific software applications in real time, providing insights into which software is being used and how frequently. This data can be used to optimize licensing and reduce costs by eliminating unused or underused software.
- **Common Technologies**:
    - **Real-Time Usage Monitoring**: Monitors software usage across the organization to track how often each software product is used.
    - **Application Control**: Ensures that only authorized users have access to certain software applications based on metering data.
- **Examples**:
    - **ManageEngine Desktop Central**: Provides software metering capabilities to track software usage and control access.
    - **Flexera Software Metering**: Monitors software usage across the enterprise and provides reports on under-utilized software.

8. Cloud Licensing Management

- **Technology Overview**: As more organizations transition to cloud-based software, **cloud licensing management** tools are needed to handle the complexities of cloud licensing agreements, such as pay-as-you-go models, subscription-based licenses, and dynamic resource allocation.
- **Common Technologies**:
    - **Pay-As-You-Go License Management**: Tracks software licenses based on consumption in cloud environments, ensuring compliance with dynamic billing models.
    - **Hybrid Licensing**: Manages both traditional on-premises licenses and cloud-based licenses within the same system.
- **Examples**:
    - **Zylo**: Provides cloud-based license management for SaaS applications, ensuring cost optimization and compliance.

- **Flexera Cloud Cost Management**: Tracks cloud licensing and manages dynamic licensing agreements in cloud environments.

9. Patch Management and Security Technologies

- **Technology Overview**: **Patch management** is essential for keeping software up to date and ensuring that vulnerabilities are addressed. Security technologies integrated with SAM tools help monitor software for security threats and ensure that all installed software is regularly patched.
- **Common Technologies**:
    - **Automated Patching**: Automatically deploys security patches and updates to all installed software.
    - **Vulnerability Management**: Scans installed software for known vulnerabilities and ensures they are patched or mitigated.
- **Examples**:
    - **Microsoft SCCM (System Center Configuration Manager)**: Manages patching and updates for large software deployments.

Software Decommissioning Process: A Detailed Guide

The **software decommissioning process** involves systematically retiring software applications or systems that are no longer in use or are being replaced by newer technologies. Decommissioning software ensures that an organization avoids unnecessary costs associated with unused licenses, reduces security risks, and maintains compliance with regulatory requirements. Properly decommissioning software also ensures that valuable data is either archived or securely deleted.

This five-page documentation outlines the key steps involved in the software decommissioning process, ensuring that all important considerations, such as data retention, security, and compliance, are addressed.

1. Introduction to Software Decommissioning

1.1 Definition

Software decommissioning is the process of removing or retiring software applications from active use within an organization. This includes removing the software from active servers or workstations, terminating licenses, managing data retention, and ensuring that security risks associated with unused software are mitigated.

1.2 Importance of Software Decommissioning

Properly decommissioning software provides several benefits:

- **Cost Savings**: Organizations can eliminate the costs associated with maintaining licenses, hardware, and support for unused software.
- **Security**: Reduces the attack surface by removing outdated software that may have security vulnerabilities.
- **Compliance**: Helps organizations comply with data retention policies and software license agreements.
- **Optimization**: Frees up resources, such as storage and IT personnel, allowing them to focus on active systems and new technology deployments.

1.3 Key Stakeholders

- **IT and Operations Teams**: Responsible for uninstalling and retiring the software.
- **Legal and Compliance Teams**: Ensure that decommissioning meets regulatory and contractual obligations.
- **Finance Teams**: Manage the cancellation of licenses and subscriptions to reduce ongoing costs.
- **Security Teams**: Ensure that data associated with decommissioned software is securely archived or destroyed.

2. Steps in the Software Decommissioning Process

2.1 Step 1: Planning and Assessment

Before decommissioning software, a thorough planning and assessment phase is necessary to determine which software needs to be decommissioned and how it will impact the organization.

- **Inventory and Assessment**: Identify all software applications to be decommissioned. Assess the usage, business impact, and interdependencies with other systems.
  - Tools such as **Flexera** or **ServiceNow** can help with inventory management and determining software usage.
- **Stakeholder Engagement**: Engage relevant stakeholders, including end-users, IT, legal, and finance departments, to ensure that the decision to decommission is well-understood and agreed upon.
- **Data Retention Requirements**: Assess data retention policies to determine what data must be retained from the software (e.g., customer records, transaction data, compliance-related information) and what can be securely deleted.

- **Compliance Check**: Ensure that the software decommissioning complies with legal, regulatory, and contractual requirements, such as data privacy laws (e.g., GDPR) and software licensing agreements.

2.2 Step 2: Data Backup and Archiving

One of the most critical steps in decommissioning software is ensuring that any valuable data is properly handled. Data associated with the software may need to be archived for future reference, especially for regulatory or compliance purposes.

- **Identify Critical Data**: Determine which data needs to be retained, and for how long, based on organizational policies and regulatory requirements.
- **Backup and Archive**: Use appropriate data backup solutions (e.g., **Veeam**, **Acronis**) to securely archive critical data. Ensure that backups are stored in a secure and accessible location.
  - **Metadata Documentation**: Document the metadata related to the archived data, including what the data represents, its origin, and the retention duration.
- **Data Retention Policy**: Establish a data retention schedule to ensure that archived data is stored for the required duration and securely disposed of once it is no longer needed.

2.3 Step 3: License Management and Termination

Managing software licenses is a critical part of the decommissioning process, as organizations must ensure that they terminate unused licenses to avoid unnecessary costs.

- **License Inventory**: Use SAM tools (e.g., **Snow License Manager**, **Ivanti Asset Manager**) to create an inventory of licenses associated with the software.
- **Contractual Obligations**: Review software licensing contracts to identify any obligations related to decommissioning, such as early termination fees or contract expiration terms.
- **Terminate Subscriptions**: For SaaS applications, notify the vendor of the intention to terminate the subscription, and ensure that automatic renewals are cancelled.
- **Reclaim Unused Licenses**: If the software being decommissioned is still under active use in other parts of the organization, reassign or repurpose the licenses.

2.4 Step 4: Uninstallation and Removal

Once data has been secured and licenses have been managed, the next step is the technical process of uninstalling and removing the software from the organization’s infrastructure.

- **Uninstallation Plan**: Create a plan for uninstalling the software across all relevant devices (servers, desktops, etc.). Ensure that any dependencies or integrations with other systems are considered.
- **Use Automation Tools**: Leverage automation tools (e.g., **Microsoft SCCM**, **ManageEngine Desktop Central**) to remotely uninstall software across multiple endpoints.
- **Verify Software Removal**: After uninstallation, perform scans to verify that all instances of the software have been completely removed. This includes checking for any residual files, libraries, or configurations left behind.
- **Security Risks**: Ensure that any remaining components or vulnerabilities associated with the decommissioned software are patched or mitigated.

2.5 Step 5: Post-Decommission Audit and Reporting

After the software has been successfully decommissioned, it is important to audit the process to ensure that it was completed in compliance with organizational policies and industry regulations.

- **Audit Documentation**: Document the entire decommissioning process, including steps taken for data backup, license termination, and software removal.
  - Include information on the software version, the number of instances removed, licenses terminated, and the fate of associated data.
- **Compliance Reporting**: Prepare compliance reports as needed for legal or regulatory bodies, showing that the decommissioning was handled securely and in accordance with applicable laws (e.g., GDPR, HIPAA).
- **Stakeholder Sign-off**: Obtain sign-offs from all relevant stakeholders, including IT, legal, compliance, and finance teams, confirming that the software decommissioning was carried out correctly.
- **Security and Risk Review**: Conduct a final security review to ensure that there are no remaining vulnerabilities or risks associated with the decommissioned software.

3. Considerations for Software Decommissioning

3.1 Data Privacy and Security

- Ensure that any data associated with the decommissioned software is handled according to the organization’s data privacy policies.
- For highly sensitive data, such as personal customer information, consider using encryption and secure deletion methods to ensure data is not recoverable.

3.2 Cost Management

- Proper decommissioning helps reduce ongoing costs associated with unused software, such as licensing fees and maintenance support.
- Consider renegotiating licensing contracts with vendors to minimize early termination fees.

3.3 Legal and Regulatory Compliance

- Comply with all legal and regulatory obligations when decommissioning software. This includes maintaining data records for required durations and properly disposing of software and associated data when necessary.
- Work closely with the legal team to ensure all decommissioning actions are compliant with local and international laws, such as data privacy regulations.

4. Tools for Supporting the Decommissioning Process

Several tools can help organizations streamline the software decommissioning process, including:

- **ServiceNow IT Asset Management (ITAM)**: Helps manage the lifecycle of software assets, including decommissioning, by tracking inventory, licenses, and contracts.
- **Flexera**: Provides tools for software license management, compliance monitoring, and decommissioning of software assets.
- **Ivanti Asset Manager**: Assists in automating the software decommissioning process, reclaiming unused licenses, and ensuring compliance.
- **Microsoft SCCM**: Enables centralized management of software uninstallation across multiple devices in an organization.

5. Summary on Software Decommissioning

Properly decommissioning software is a crucial part of software asset management. It ensures that organizations are not burdened by unnecessary costs, reduces security risks, and helps maintain compliance with legal and regulatory obligations. By following the steps outlined in this document—planning and assessment, data backup, license management, uninstallation, and post-decommission audits—organizations can effectively manage the end-of-life for their software assets.

Effective decommissioning not only protects the organization from financial and security risks but also optimizes its IT infrastructure, making it leaner and more efficient.

Private Large Language Models: Key Details

Large Language Models (LLMs) like GPT, BERT, and others have demonstrated extraordinary capabilities in natural language understanding, generation, and processing. While most prominent LLMs (e.g., OpenAI's GPT-4 or Google's Bard) are cloud-based and proprietary, a growing interest has emerged in building and deploying private large language models (LLMs). These are models owned, controlled, and used internally by organizations or individuals, offering unique advantages and trade-offs. Below, we delve into critical details across development, deployment, applications, and challenges.

What are Private LLMs?

Private LLMs are artificial intelligence models developed, fine-tuned, or deployed for exclusive use within an organization or private context. Unlike public APIs provided by OpenAI or Google, private models are hosted on local infrastructure or private cloud environments, ensuring greater control, security, and customization.

Categories of Private LLMs:

In-House Developed Models:

Organizations with technical expertise build LLMs from scratch, often leveraging open-source frameworks like TensorFlow, PyTorch, or Hugging Face.

Fine-Tuned Pre-trained Models:

Organizations customize pre-trained, open-source LLMs like OpenLLaMA, Falcon, or GPT-NeoX using domain-specific data.

Licensed Commercial Models:

Enterprises deploy vendor-hosted LLMs on private infrastructure under strict data-use agreements.

Advantages of Private LLMs

Data Privacy and Security:

Data processed by private models never leaves the organization's infrastructure, minimizing risks of leaks or exposure to third parties.

This is particularly crucial for industries handling sensitive information, like banking and finance (GDPR adherence), or government agencies.

Customization:

Models can be fine-tuned on proprietary datasets, ensuring better performance on domain-specific tasks. For example, a software firm might fine-tune a model to generate software case study documentation and research documentation.

Cost Efficiency for Scale:

For organizations with substantial usage, hosting a private LLM can be cheaper than API-based services with per-request billing.

Control and Governance:

Organizations have full control over model updates, training, and usage policies. Internal models reduce reliance on third-party vendors, offering resilience against API outages or policy changes.

Technological Foundation and Development

Model Architecture:

Modern LLMs are based on transformer architectures, which excel in processing sequential data. The popular frameworks for development include Hugging Face Transformers, LangChain, and libraries from Meta’s PyTorch ecosystem.

Training Considerations:

Data Requirements: Building LLMs requires vast, high-quality datasets. Public corpora (e.g., Common Crawl, Wikipedia) and proprietary data are combined for training.

Computing Power: High-performance GPUs or TPUs (e.g., NVIDIA A100 or Google TPU v4) are essential for training models with billions of parameters.

Pretraining vs. Fine-Tuning: Organizations often rely on pre-trained open-source models (e.g., GPT-J or LLaMA) and focus resources on fine-tuning for their specific needs.

Deployment:

Deployment options include on-premises servers, private cloud environments, or edge devices, depending on latency, cost, and security requirements. Tools like Docker, Kubernetes, and Ray serve as critical infrastructure for scaling.

Use Cases of Private LLMs

Enterprise Productivity:

Automating document generation, summarization, and customer support. Enhancing search within proprietary databases (e.g., semantic search for research papers).

Healthcare and Life Sciences:

Private LLMs can process patient records, suggest treatments, and summarize medical literature while adhering to data protection laws.

Government and Defense:

Governments use private LLMs for intelligence analysis, secure communication, and policy drafting.

Education and Research:

Universities and research institutions fine-tune models to aid in research, grant writing, or customized learning platforms.

Challenges of Private LLMs

Resource Intensity:

Training large models demands significant computational resources, making it prohibitively expensive for smaller organizations. For instance, GPT-3 required tens of millions of dollars to train on 175 billion parameters.

Data and Bias:

Ensuring data quality is critical. Private models risk inheriting biases present in training datasets. Organizations must implement thorough validation and bias mitigation strategies.

Expertise Requirements:

Building and maintaining private LLMs necessitates advanced expertise in machine learning, software engineering, and data science. Many companies struggle to hire and retain such talent.

Maintenance and Updates:

Continuous updates to adapt to evolving language trends and data are resource intensive. Model "drift" can occur if not retrained periodically.

Future Trends in Private LLMs

Decentralized and Efficient Training:

Emerging technologies like model distillation and parameter-efficient tuning (e.g., LoRA) make it feasible to train smaller, task-specific models with less computing power.

b. Federated Learning

Federated learning allows multiple organizations to collaboratively train models without sharing raw data, preserving privacy.

Open-Source Expansion:

Initiatives like EleutherAI and Meta’s LLaMA contribute to the open-source ecosystem, enabling more organizations to develop private LLMs.

Specialized Models:

The focus is shifting toward domain-specific LLMs, which outperform general-purpose models in niche applications.

Summary

Private large language models represent a transformative technology for organizations that prioritize security, customization, and scalability. While resource-intensive, advances in open-source tools and training methodologies are lowering barriers, making private LLMs more accessible. Organizations should carefully evaluate their needs, infrastructure, and expertise before embarking on the journey of private LLM development or deployment.