

**AGILE MODELLING**

**WITH UML:**

**Enhancing Software Development Efficiency.**

**BY**

**NAME: TALABI DAMILARE HANNAH.**

**MATRIC NO: 21/7954**

**DEPARTMENT: COMPUTER SCIENCE.**

**LEVEL: 300.**

**COURSE: SYSTEM ANALYSIS AND DESIGN (CSC 311).**

**MARCH, 2024.**

Contents

[EXECUTIVE SUMMARY 3](#_Toc160290442)

[INTRODUCTION 4](#_Toc160290443)

[LITERATURE REVIEW 5](#_Toc160290444)

[DISCUSSION 8](#_Toc160290445)

[CONCLUSION 19](#_Toc160290446)

[RECOMMENDATIONS 20](#_Toc160290447)

[BIBLIOGRAPHY 23](#_Toc160290448)

# EXECUTIVE SUMMARY

**Agile modeling (AM)** is a practice-based approach for efficiently modelling and documenting software-based systems. It is a set of ideals, guidelines, and procedures that can be simply and effectively implemented into a software development project. It assists developers in designing a tailored software development process that satisfies their requirements and is adaptable enough to change course in the future.

The **Unified Modelling Language**, or **UML**, is a potent tool for creating and visualizing software systems that helps improve agile development's documentation, design, and communication. Using UML, agile teams can:

* Establish a visual language that developers, product owners, and other stakeholders can all understand.
* Support the creation of detailed design artifacts like class diagrams, sequence diagrams, and activity diagrams.
* Act as lightweight documentation that is easily updated as the project develops.

UML can thus offer a standardized means of graphically representing complex systems and their interconnections while preserving the flexibility and adaptability of agile development.

**Agile Modeling and Agile Methodologies**:

Agile approaches, including XP, Scrum, and Kanban, are broad frameworks for work management. One technique that can be used with these approaches is agile modeling.

Relationship:

* Project management, teamwork, and delivery are the main focuses of agile approaches.
* By offering system visualizations, Agile Modeling improves these approaches.

UML diagrams facilitate communication and development and fit in well with agile methodologies.

# INTRODUCTION

The process of developing software is complex and constantly shifting, requiring cooperation and ongoing modification from all parties involved. Conventional software engineering techniques, like the waterfall model, frequently fall short of users' and consumers' evolving needs and expectations.

By integrating agile ideas and practices into software modelling, Agile Modelling (AM) is a methodology that seeks to overcome such challenges. AM supports modelling that is developing, evolutionary, and iterative and that is motivated by value and feedback. In order to capture various aspects of the software system, AM also encourages the usage of diverse models and perspectives.

UML is a standard notation that provides similar syntax for describing the relationships behavior, and structure of software systems. A variety of diagram formats that can be used for many audiences and purposes are supported by UML, including class, use case, sequence, and state diagrams. Additionally, a variety of software development strategies, including agile methods, are compatible with UML.

But there are some complications in the relationship between AM and UML. On the one hand, UML might be considered a helpful tool for AM since it makes documentation, communication, and software model analysis easier. However, because it can add needless complexity, rigidity, and overhead to the modeling process, UML can also be considered as a barrier to AM. Therefore, the question arises: how can UML be effectively used in AM?

This project seeks to explore this issue and offer some perspectives and recommendations for applying UML in AM. Its central claim is that UML can be helpful for AM if it is used in a flexible, pragmatic, and customized manner. It will also review the literature that has already been written about AM and UML, weigh the benefits and drawbacks of UML in AM, and provide some case studies and examples of UML in AM.

# LITERATURE REVIEW

In the discipline of software engineering, numerous researchers and practitioners have expressed interest in and debated the integration of agile approaches and UML. On the one hand, some contend that UML can improve agile processes by giving software models a visual representation and a shared language, which can help with documentation, analysis, and communication. However, others argue that by adding needless complexity, rigidity, and overhead to the modelling process, UML might undermine agile techniques.

Several studies have been conducted to investigate the benefits and challenges of using UML in agile environments. For example, Ambler and Jeffries (2002), for instance, carried out a case study of an XP project that included UML as a drawing tool and discovered that while UML was helpful for expressing and exploring design concepts, it should not be overused or formalized. Comparably, Cao and Ramesh (2008) polled 106 software professionals who employed UML in agile projects and discovered that while UML was beneficial for enhancing productivity, quality, and communication, it also revealed certain obstacles and challenges, like a lack of standardization, tool support, and training.

A number of best practices and frameworks have also been put out to help with the efficient application of UML in agile projects. In this instance, the Agile Modelling (AM) approach was introduced by Ambler (2004) and applies agile techniques and ideas to software modelling. AM supports modelling that is incremental, evolutionary, and iterative and that is motivated by value and feedback. In order to capture various facets of the software system, AM also encourages the usage of diverse models and perspectives. Additionally, AM offers several recommendations for applying UML in agile projects, including these:

* Make sure your models are created with a specific goal in mind, such as interacting with stakeholders, verifying requirements, or investigating design options.
* Model in compact steps: Don't model too much or too far ahead of time; instead, only make models that are adequate for the present iteration.
* Model at the proper level of abstraction: Don't model information that isn't relevant or unimportant; instead, only make models that are appropriate for the target audience and the level of detail needed.
* Model with appropriate notation: Don't utilize extraneous or perplexing symbols or elements, and limit your use of UML notation to what is appropriate for the type of model and its intended use.
* Choose the appropriate tool for the job: Steer clear of clumsy, costly, or incompatible tools and only utilize modelling tools that facilitate modelling activities and agile methodologies.

An additional illustration of a UML framework utilized in agile projects is the ICONIX method, which was suggested by Rosenberg and Scott (2005). Use case diagrams, domain class diagrams, robustness diagrams, and sequence diagrams are the four key UML diagrams that are the center of the ICONIX process, which is a simplified and practical method. These four diagrams serve as the primary artefacts used by the ICONIX process, which directs the software development process from requirements through design and execution. Additionally, the ICONIX method offers some best practices for applying UML in agile projects, like:

* The process is driven by use cases, which serve as both the foundation for design and testing as well as the main source of requirements.
* Analysis of robustness fills up the gaps: The primary components and interactions inside the system are identified, and the use cases are validated, using robustness diagrams.
* Dynamics are displayed in sequence diagrams: Sequence diagrams are used to show how the system's items cooperate and behave.
* Class diagrams display the statistics: Class diagrams are used to specify the characteristics and functions of the system's objects.

Another illustration is Bernhard Rumpe (2017) who explained how an online auction system can be developed using a model-based approach to software development. The following characteristics of agile modelling are demonstrated by the case study:

* Using UML as a high-level programming language capable of producing tests and executable code.
* Models are used in code development, analysis, simulation, and documentation.
* Model transformations are used to advance the architecture and design of the system.
* Automating checks to make sure the models and the system are accurate and of high quality.
* Modelling the requirements and system behavior through a test-first approach.

There are still certain holes or places where UML may improve agile practices, nevertheless, in spite of the studies, frameworks, and best practices that are now in place. As an illustration, a few of the holes or places are:

* How to adapt UML to several agile approaches, including Lean Software Development, XP, Scrum, and Kanban.
* How to include UML into other agile techniques like pair programming, user stories, test-driven development, and retrospectives.
* How to assess the influence of UML on the results of agile projects, like software quality, team performance, and customer happiness.
* How to modify UML to fit new trends and technologies like blockchain, AI, and cloud computing.

These drawbacks or vulnerabilities show that applying UML in agile projects may still be creatively and improved upon.

# DISCUSSION

Agile Modelling is a collection of procedures for documentation and modelling in an agile environment. Using UML (Unified Modelling Language) in an agile setting is known as "agile modelling."

It blends efficient modelling techniques with the ideas of agile development. By offering simple, iterative modelling techniques, it enhances agile methodologies.

UML offers an effective toolkit for software system design and visualization that improves documentation, design, and communication.

**Agile methodologies** are adaptable and team-based project management techniques. Their top priorities includes: including team members at every stage, providing value, and adjusting to change.

The gap between agile methodologies and UML is filled by Agile Modelling using UML. Some common agile methodologies are:

* **Scrum**: An organized framework that organizes work into time-boxed iterations called sprints. It comprises daily stand-ups, product backlogs, sprint backlogs, roles (Product Owner, Scrum Master, Development Team), and artefacts. It is a prescriptive process that has planned meetings and defined roles.
* **Kanban**: Reduces work-in-progress, maximizes cycle time, and visualizes work flow. WIP limitations, flow, and visualization are prioritized in Kanban.
* **XP** stands for "eXtreme Programming," which emphasizes customer cooperation, responsiveness, and high-quality software. XP places a high value on responsiveness and software quality.

Agile methodologies, such as Scrum and Kanban, have become increasingly popular in the software development industry due to their adaptability to changing requirements and flexibility. Nonetheless, a lot of teams and developers are confused about how to successfully integrate UML (Unified Modelling Language) modelling into their agile procedures. A strong toolkit for developing and visualizing software systems is provided by UML, which helps improve documentations, design, and communication.

**Agile Development’s Use of UML**

Prior to discussing integration tactics, let's examine the role that UML plays in agile development:

1. **Visualization**: For developers, product owners, and other stakeholders, UML diagrams offer a consistent visual language. They aid in establishing a common knowledge of the architecture, behavior, and design of the system.
2. **Design**: Using UML, one may create intricate design artifacts like as activity, sequence, and class diagrams. These can be extremely helpful in the development process to help with well-informed design choices.
3. **Documentation**: UML diagrams can act as a lightweight form of documentation that can be updated as the project moves forward, even if agile techniques place a higher priority on working software than extensive documentation.

**Integration Techniques**

1. **Start simple**

Take a basic approach to UML modelling at first. Avoid overloading your agile teams up front with intricate graphics and copious documentation. Start with a few critical diagrams that solve pressing issues. For example, class diagrams can be used to illustrate important software components, and user narrative maps can be used to illustrate user journeys.

1. **Just-In-Time Modeling**

Agile methods place a strong emphasis on adapting to change. When it comes to UML modelling, apply the same principle: only draw diagrams when necessary. For example, use a UML diagram to explain and record a particularly difficult user story or architectural choice that you come across.

1. **Collaboration is key**

One team member shouldn't be in charge of all UML diagrams. Promote cooperation between architects, product owners, developers, and other stakeholders. To ensure that everyone's thoughts are taken into account, the entire team can take part in the creation and review of UML diagrams.

1. **Use Digital Tools**

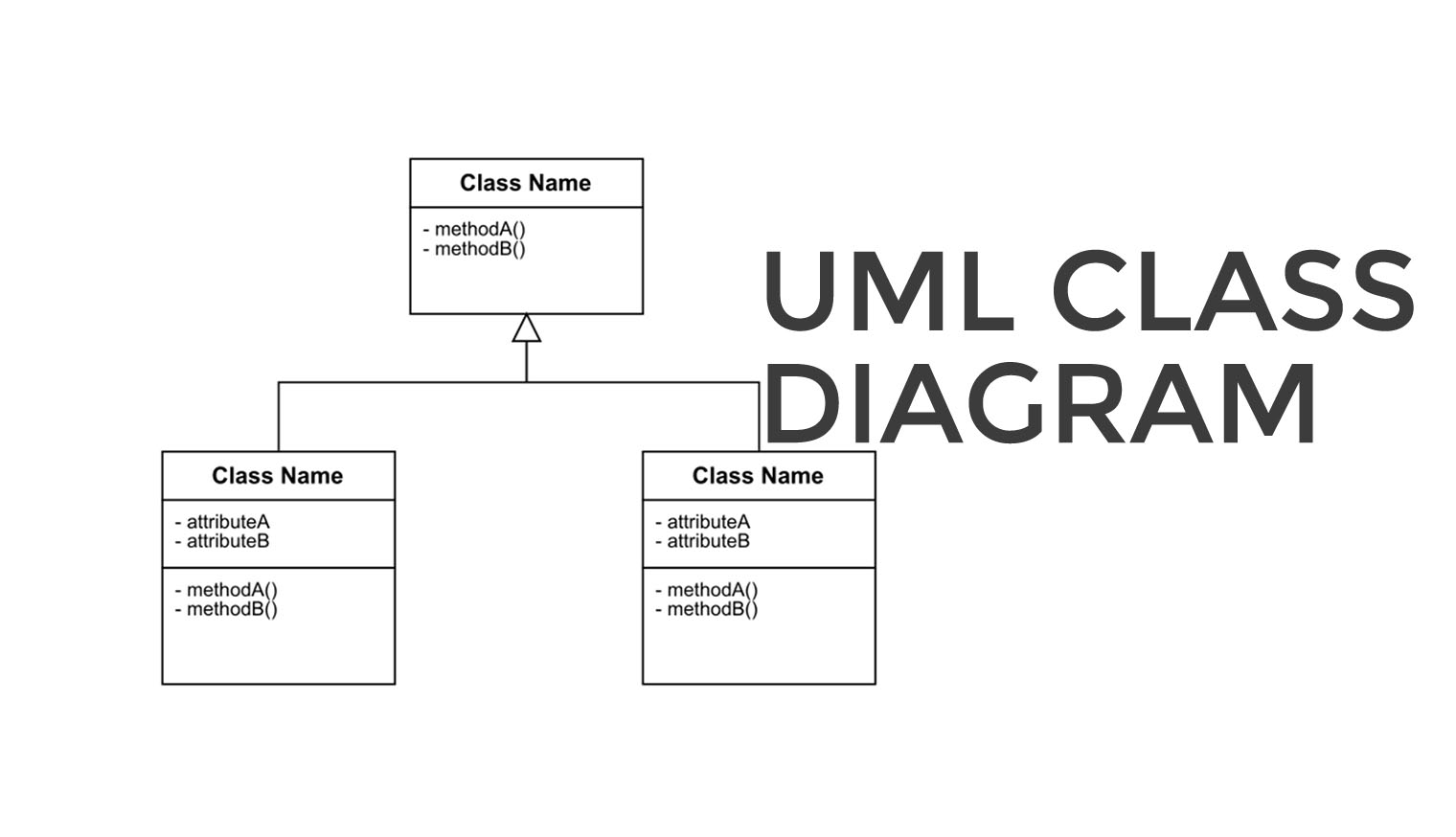
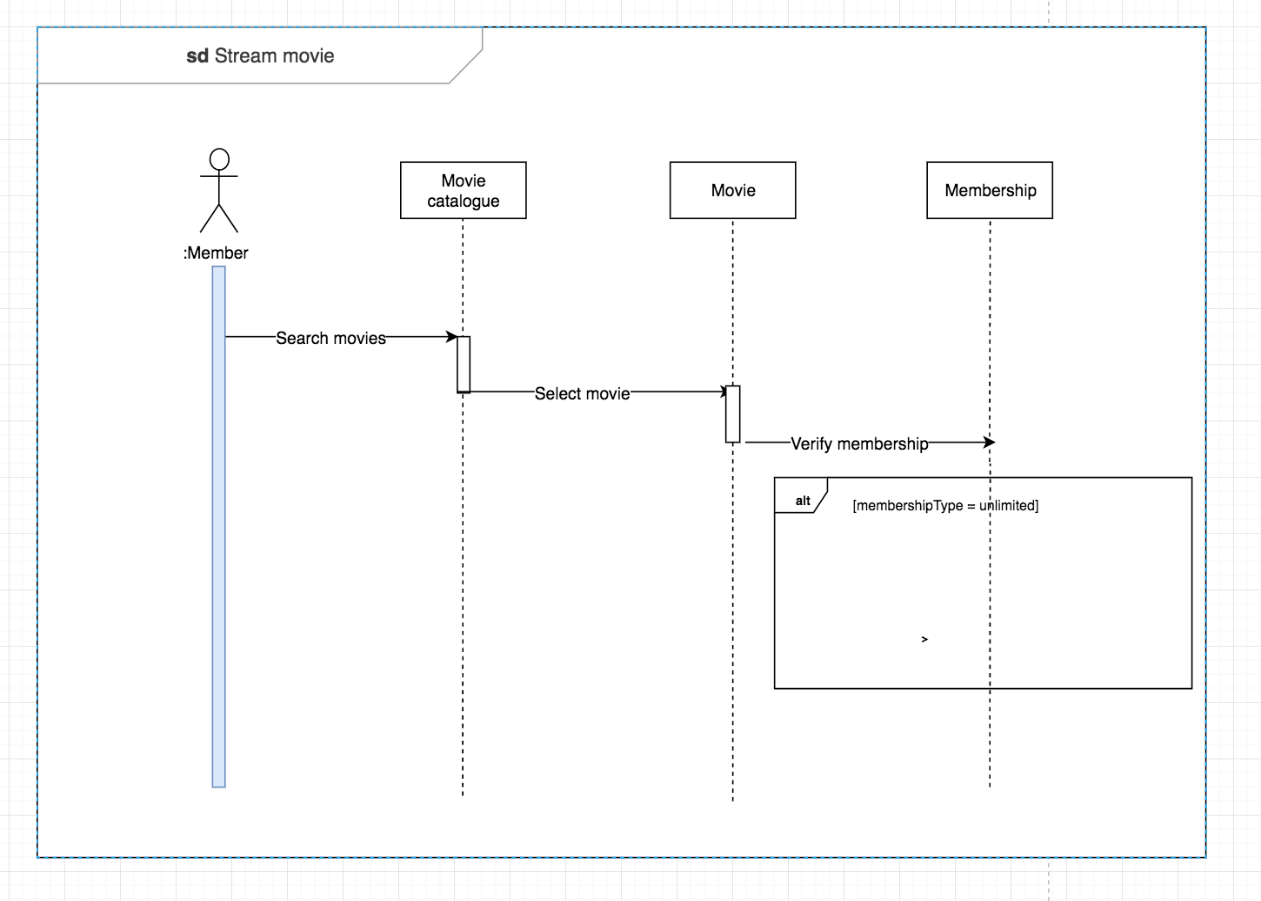
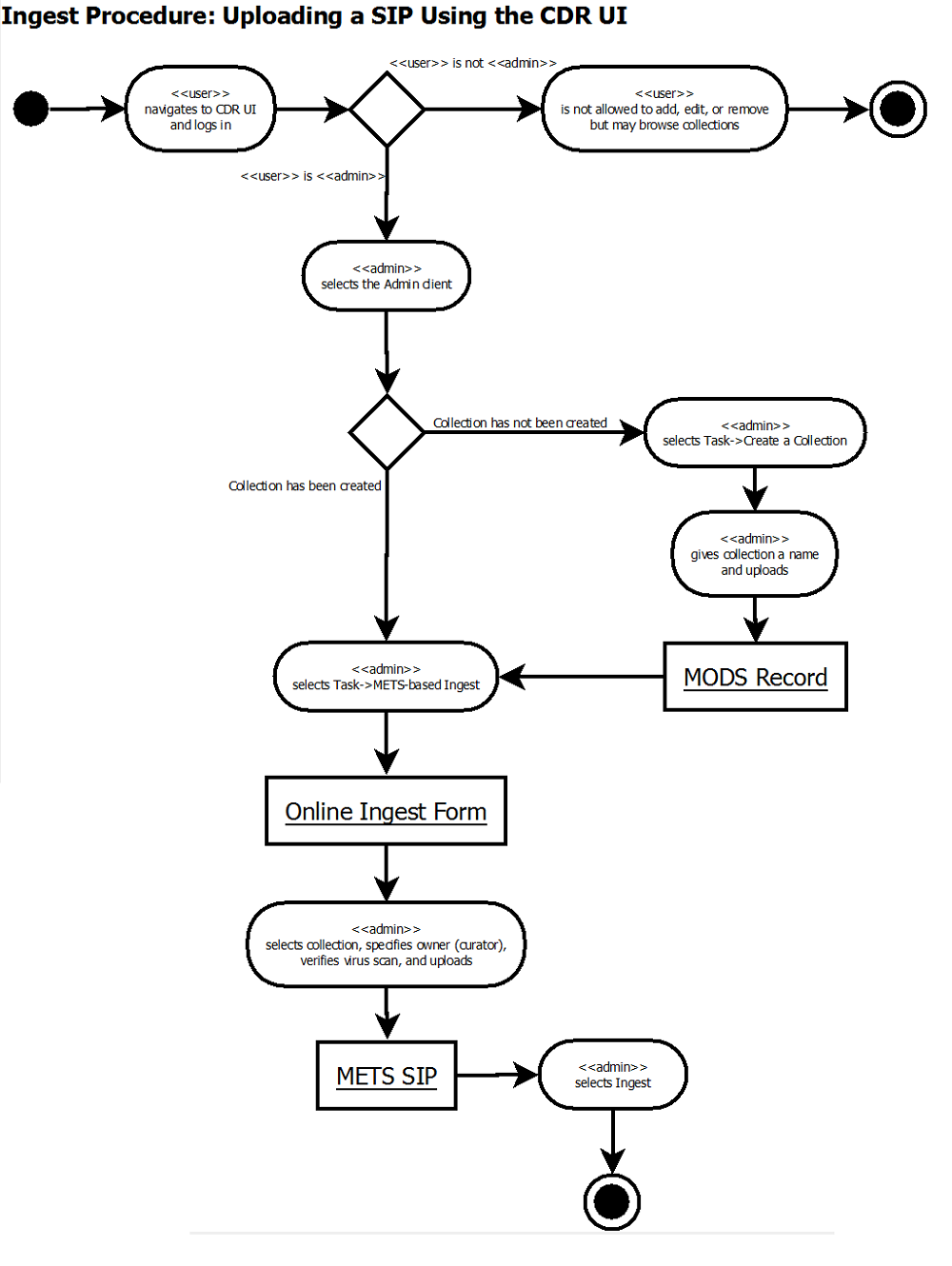
Use UML modelling tools such as Jira or Trello that are well-integrated with agile project management solutions. With the use of these tools, you can ensure that UML diagrams are updated as the project moves forward and ease the process of developing and sharing them.

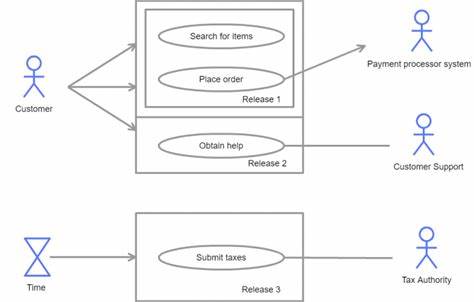
1. **Iterate and Refactor**

Iterate on your UML diagrams in the same way that you would with your code. Reexamine and rework your UML diagrams as the project progresses to ensure they remain consistent with the program as it stands. By doing this, documentation may be kept current.

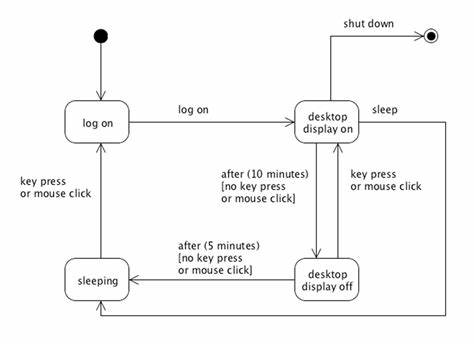
**UML Diagrams for Agile Teams**

Various UML diagrams have different uses in agile development.

* **Class Diagrams**: These show the classes, attributes, and relationships between them in your software's static structure. They are useful for comprehending the overall architecture and creating data models.
* **Sequence diagrams**: are useful for visualizing the dynamic behavior of a system, particularly when examining how various actors or components interact with one another. Sequence diagrams are useful for comprehending intricate user stories.
* **Activity Diagrams**: These show how a system's workflow and control flow are organized. They work incredibly well for outlining the steps in a particular process or user story.
* **Use Case Diagrams**: Use case diagrams are a useful tool for identifying and documenting various user roles and their interactions with the system when working with user stories.



* **State Diagrams**: State diagrams are useful for recording and visualizing complicated state transitions in your software.



**Choosing Appropriate UML Diagrams for Agile Processes**

Several UML diagrams can be used at different phases of an Agile software development process to satisfy particular demands and improve stakeholder and team communication. When to use some of the most popular UML diagrams is shown here:

**Class Diagrams**:

* When to Use Them: Class diagrams are usually used in the early phases of a project to define the architecture and data models of the system.
* Purpose: Use them to depict the software's static structure, which includes the classes, their characteristics, and the connections between them.
* Scenarios: Class diagrams come in handy for discussing high-level system architecture or designing the underlying data structure.

**Sequence Diagrams:**

* When to Use Them: When you want to see how several actors or components interact with one another, sequence diagrams come in handy, especially in the development stage.
* Purpose: Use them to demonstrate the dynamic behavior of your system, such as the order in which objects send messages or invoke methods on one another.
* Scenarios: Complex user stories or scenarios involving several system components can be better understood and documented with the use of sequence diagrams.

**Activity Diagrams**:

* When to Use Activity Diagrams: Because of their versatility, activity diagrams can be applied to all stages of a project, including requirement analysis, design, and testing.
* Purpose: Use them to depict business processes, workflows, and the control flow inside a system.
* Scenarios: Activity diagrams, such as flows of user interaction or business processes, are useful for recording and illustrating the steps in a particular process.

**Use Case Diagrams:**

* When to Use Them: Use case diagrams are usually made in the early phases of a project, most frequently when gathering requirements.
* Use them to specify various user roles, how they interact with the system, and what kind of high-level functionality the system offers.
* Scenarios: Use case diagrams are a useful tool for identifying and documenting features or user stories that require implementation**.**

**State Diagrams:**

* When to Use Them: During the design and development phase of software development, state diagrams come in handy when your program includes intricate state transitions.
* Use them to see an object's states and the ways in which it changes between them in response to circumstances or events.
* Scenarios: State diagrams are a useful tool for simulating the behavior of individual parts or objects that undergo changes between different states.

Keep in mind that flexibility and adaptability are promoted by agile development. The particular requirements of your project should determine which UML diagrams to use and when. Achieving a balance between providing the team with just the right amount of documentation to aid in development and not overloading them with needless diagrams is crucial. Throughout the Agile process, regular cooperation and communication between team members and stakeholders will assist you in determining how best to use UML diagrams.

Whether utilizing Scrum or Kanban, integrating UML modelling into agile software development helps improve documentation, design, and communication without sacrificing agility. Keep in mind that collaboration, iteration, and lightweight design are crucial. UML diagrams ought to work in tandem with your agile procedures and adjust to your project's evolving requirements. When used carefully, UML may be a great tool for creating high-caliber software inside an agile framework.

**Values and Trade-offs**

**Visualization and Communication:**

* Value: UML diagrams give developers, product owners, and stakeholders a consistent visual language for visualization and communication. They aid in communicating system interactions, architecture, and design.
* Difficulties/Trade-offs: Agile prioritizes working software and teamwork over extensive documentation. It's possible that there are situations when updating UML diagrams quickly is not worth the effort.

**Requirements Elicitation and Analysis:**

* Value: UML diagrams help clarify and improve requirements. Activity diagrams, class diagrams, and use cases aid stakeholders in understanding the behavior of the system.
* Issues/Trade-offs: It can be difficult to strike a balance between thorough documentation and agility. Diagrams that are too complicated could impede agility.

**System Design and Architecture:**

* Value: Class and component diagrams from UML are useful tools for system architecture design. They show components, relationships, and classes.
* Difficulties/Trade-offs: Emergent design, or iterative architecture evolution, is encouraged by agile. Updates to UML diagrams may be necessary frequently as design choices shift.

**Modelling Behavior and Interactions:**

* Value: State machine diagrams and sequence diagrams are useful tools for modelling interactions and dynamic behavior. They support the identification of synchronization spots, concurrency problems, and bottlenecks.
* Difficulties/Trade-offs: System dynamics knowledge is necessary to create correct sequence diagrams. Regular behavioral adjustments may result in maintenance costs.

**Validation and Testing:**

* Value: Test case design is guided by UML models. Class diagrams guarantee thorough coverage, whereas state machine diagrams aid in identifying test cases.
* Difficulties/Trade-offs: It can be difficult to make sure that UML models and changing code are in alignment. Testers could be misled by antiquated diagrams.

**Code Generation and Reverse Engineering:**

* Values: Class diagrams can be used by UML tools to produce skeleton code. By using reverse engineering, UML diagrams can be made from already written code.
* Problems/Trade-offs: Code generated might not always match UML models exactly. Reverse-engineered schematics may be missing some information.

**Maintenance and Evolution:**

* Value: UML helps to understand existing systems during maintenance. It highlights dependencies, interfaces, and effect regions.
* Challenges/Trade-offs: Keeping diagrams current as the system evolves needs effort; neglecting updates can result in outdated documentation.

When applying UML in Agile contexts, think about concentrating on simple diagrams that provide quick benefits. Strike a balance between the Agile idea of adapting to change rather than adhering to a plan and the requirement for documentation.

# CONCLUSION

**Summary of findings:**

* Adaptability, teamwork, and iterative development are given top priority by agile approaches like Scrum, Kanban, and XP.
* UML-based agile modeling improves documentation, design, and collaboration in an agile setting.
* The design and behavior of systems are visually represented by UML diagrams, such as class and sequence diagrams.
* UML and agile approaches work well together to support efficient development processes.

**Effectiveness of UML Integration with Agile Processes:**

* Advantages:

Clarity: Teams can better comprehend system behavior, linkages, and structure by using UML diagrams.

Alignment: Iterative design and minimal documentation are agile principles that are in line with UML.

UML promotes cooperation between stakeholders, clients, and team members.

* Problems:

Above: Over-modeling might make a person less agile. It's important to maintain balance.

Learning Curve: Teams must comprehend the practical applications of UML notation.

Adaptability: When requirements change, UML models ought to change too.

**The impact for Software Development Teams**:

* Education and Acceptance:

Teams should be trained in the fundamentals of UML and how to apply it to agile projects.

Promote the usage of UML techniques gradually.

* Context Is Important

Select UML diagrams (class diagrams for architecture, sequence diagrams for behavior, etc.) in accordance with the requirements of the project.

UML procedures should be modified to the team's needs.

* Working Together and Communicating:

During sprint planning, design talks, and code reviews, use UML as a communication tool.

Utilizing UML, developers, testers, and product owners may work together across roles.

To sum up, the use of UML into agile processes enables teams to efficiently conceptualize, communicate, and create software. Succeeding in modern software development requires striking a balance between modeling discipline and adaptability.

# 

# RECOMMENDATIONS

1. **Encourage Just-in-Time UML Modeling:**

Contextual Relevance: Software systems are visually represented by UML (Unified Modeling Language) diagrams. Encourage teams to develop UML diagrams just-in-time, when they are most needed, as opposed to creating them all at once.

The incremental approach Depending on the stage of the project, start with the most important diagrams (such as use case, class, or sequence diagrams). Diagrams should be added to or modified as requirements change.

1. **Encourage cooperation among teammates**:

Cross-Functional Communication: Agile approaches place a strong emphasis on teamwork. Promote frequent communication between stakeholders, including product owners, testers, and developers.

Pair Modeling: Take into account having pair modeling sessions when drawing UML diagrams. Together, the two team members brainstorm, create, and polish the diagrams. This lessens silos and encourages mutual understanding.

Feedback Loops: Request feedback frequently and in advance. As a team, check over UML diagrams frequently to make sure they are accurate and in line with the project's objectives.

Steer clear of overhead: In the event that requirements change, excessive advance modeling may result in lost effort. Diagrams are kept in line with the changing system thanks to just-in-time modeling.

1. **Leverage Project Management Platforms with Integrated Digital UML Tools:**

Select the Appropriate Tool: See which cutting-edge UML modelling tools work well with project management systems (like Jira, Trello, or Azure DevOps). Aspects like as export capabilities, version control, and real-time collaboration should be looked for.

1. **Advantages of Integration**

Efficiency: Teams may generate, share, and edit UML diagrams in the same environment that manages project activities thanks to integrated tools, which expedite the process.

Traceability: Establish a direct connection between UML artifacts (such class or activity diagrams) and user stories, tasks, or requirements. This guarantees alignment and improves traceability.

Visibility: UML diagrams are readily accessible to project managers and stakeholders in addition to other project artifacts, which promotes transparency.

1. **UML tool examples include:**

Lucidchart: A cloud-based UML-compatible diagramming application.

Visual Paradigm: Provides integration with several project management applications, UML modeling, and teamwork.

Draw.io: An open-source, free application for drawing UML diagrams (now a component of diagrams.net).

Keep in mind that flexibility and agility are essential. Instead of impeding communication, UML modelling should facilitate it. Teams can adhere to agile principles and produce UML artifacts that are effective by using these tips.

# BIBLIOGRAPHY

Ambler, S. (2002, August 14). *Agile modeling: Effective practices for extreme programming and the Unified Process*. Google Books.

Ambler, S. W. (2004). *The object primer - agile model-driven development with UML 2.0*. Scribd

Andrews, A., Runeson, P., & France, R. (2004, June). *Requirements Trade-offs During UML Design.* <https://www.researchgate.net/publication/220344291_Trade-off_Analysis_for_Requirements_Selection>

Beedle, M., Bennekum , A. van, Cockburn, A., Cunningham, W., Fowler, M., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. c, Schwaber, K., Sutherland, J., Thomas, D., Mellor, S., Grenning, J., & Beck, K. (2001). *Manifesto for Agile Software Development.*

Digital, M., & Bowes, J. (2020, September 18). *Kanban vs Scrum vs XP – an agile comparison*. Manifesto. <https://manifesto.co.uk/kanban-vs-scrum-vs-xp-an-agile-comparison/>

GfG. (2024, January 8). *Unified modeling language (UML) diagrams: An introduction*. GeeksforGeeks. <https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/>

Rosenberg, Doug & Stephens, Matt & Collins-Cope, Mark. (2005). *Agile Development with ICONIX Process: People, Process, and Pragmatism*. 10.1007/978-1-4302-0009-3.

Rumpe, B. (2017). *Agile modeling with UML: Code generation, testing, refactoring* (1st ed.). Springer Vieweg.

Ryu, M., Truong, H.-L., & Kannala, M. (2021, February 15). *Understanding quality of analytics trade-offs in an end-to-end machine learning-based classification system for Building Information Modeling - Journal of Big Data*. SpringerOpen. <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00417-x>

Vpadmin. (2023a, September 12). *Integrating UML modeling into Agile Software Development: A guide for Scrum and kanban teams*. Visual Paradigm Guides. <https://guides.visual-paradigm.com/integrating-uml-modeling-into-agile-software-development-a-guide-for-scrum-and-kanban-teams/>