The Relevance。of UML in the Development and Documentation of Large Distributed Software Systems

1. Introduction？: The Unified Modeling Language and Its Role in Software Engineering

1.1 Defining UML: Origins and Core Principles have

The Unified Modeling Language (UML) emerged in the late 1990s as a standardized modeling language, providing a visual means to represent the architecture, behavior, and interactions within software systems [1]. Its development was a concerted effort to unify the diverse notations prevalent in object-oriented methodologies at the time. UML effectively integrated the most successful elements from earlier approaches, such as the Object Modeling Technique (OMT), the Booch method, and Objectory [1]. The fundamental aim behind the creation of UML was to foster enhanced clarity, mitigate ambiguity, and improve communication among the various teams involved in software projects [3]. By establishing a common, visual vocabulary, UML sought to provide a more effective way to specify, visualize, construct, and document the often intricate artifacts of software systems. This standardization was particularly crucial in an era where object-oriented principles were gaining prominence, and the need for a unified approach to modeling complex software was becoming increasingly apparent. The convergence of different methodologies into UML reflected an industry-wide acknowledgment of the necessity for a shared visual language to tackle the growing complexity of software development.

1.2 Intended Applications of UML: From Design to Documentation

The scope of UML's intended applications was broad, encompassing the modeling of various types of systems, including distributed ones, as well as supporting activities like analysis, system design, and deployment [3]. It was envisioned as a versatile tool that could be applied across different phases of the software development lifecycle, from the initial high-level design stages through to the implementation and subsequent maintenance of the software [2]. UML diagrams were designed to act as a crucial link between the abstract concepts of high-level design and the concrete details of implementation. This visual representation was intended to facilitate effective communication among all stakeholders involved in a software project, from architects and developers to testers and even non-technical team members [2]. The initial vision for UML was indeed ambitious, positioning it as a universally applicable instrument capable of addressing the diverse needs of software development across various system complexities and throughout the entire project lifecycle. The specific mention of distributed systems within its intended applications is particularly noteworthy, given the user's inquiry about UML's relevance in this domain.

2. The Perceived Decline of UML in Large Distributed Systems: An Examination of Online Discourse

2.1 The Rise of Agile Methodologies and the Shift in Focus

The increasing prominence of Agile methodologies in software development is frequently cited as a significant factor contributing to the perceived decrease in UML usage, particularly in the context of large distributed systems [9]. As Agile approaches gained traction, the meticulous and often detailed nature of UML began to feel like an encumbrance in the fast-paced, iterative cycles characteristic of Agile projects [9]. In such dynamic environments, the creation and maintenance of extensive UML diagrams were sometimes regarded as bureaucratic overhead rather than as genuinely helpful aids in the development process [9]. Agile methodologies place a strong emphasis on working software, continuous delivery, and close collaboration with customers, often prioritizing these aspects over the production of comprehensive documentation, which can include detailed UML diagrams [10]. This shift in focus, driven by the core values of Agile, which emphasize rapid iteration and flexibility, can potentially lead to a diminished emphasis on formal modeling techniques like UML that are often associated with more traditional, sequential development processes. The inherent tension between the detailed planning and documentation often linked to UML and the emergent, adaptive nature of Agile development suggests a fundamental difference in approach that could explain the perceived decline in UML adoption in certain modern software development contexts.

2.2 Challenges of Maintaining UML in Complex and Evolving Systems

The creation and, more critically, the sustained maintenance of UML diagrams for large, intricate software systems, especially those that are distributed, can present substantial practical challenges [7]. These systems often involve thousands of interconnected classes and components, making the initial effort to model them comprehensively a time-consuming and resource-intensive endeavor [7]. Furthermore, software systems, particularly in the domain of large distributed applications, are rarely static; they continuously evolve through updates, new features, and refactoring. As the codebase changes, any existing UML diagrams can quickly become outdated and inconsistent with the actual implementation [2]. Keeping these diagrams synchronized with the evolving reality of the code requires a significant and ongoing investment of time and effort [2]. For systems characterized by a vast number of interconnected components, the resulting UML diagrams can themselves become so complex and unwieldy that they ultimately lose their intended utility as tools for understanding and communication [11]. The dynamic and expansive nature of large distributed systems, therefore, poses a considerable obstacle to the long-term viability and value of detailed UML documentation, as the cost of maintaining accuracy can easily outweigh the perceived benefits.

2.3 The Preference for Lightweight and Informal Visualizations

In the contemporary software development landscape, many practitioners demonstrate a preference for more lightweight and informal approaches to visualization compared to the rigor of formal UML diagrams [1]. For instance, for conveying architectural overviews or brainstorming design ideas, developers often find informal, hand-drawn diagrams or basic block diagrams to be sufficient and more readily adaptable [1]. In smaller projects or within the context of Agile methodologies, lightweight tools such as simple flowcharts, Kanban boards for workflow visualization, or even the use of sticky notes for quick conceptualization are frequently favored for their speed and inherent flexibility [9]. When it comes to understanding the intricate, low-level behavior of specific components within a system, developers often find that the actual source code itself, or well-maintained API specifications, provides a more direct and authoritative source of information than high-level UML diagrams [11]. This trend suggests a move towards pragmatism in visual modeling, where the perceived value of investing in the creation and maintenance of formal UML diagrams is weighed against the agility and immediacy offered by more lightweight and code-centric approaches. The industry appears to be leaning towards a "just enough documentation" philosophy, where the level of formality in modeling and documentation is carefully considered based on the specific needs of the project, the team, and the context.

3. UML Usage in the Documentation of Open-Source Large Distributed Systems

3.1 Investigating Documentation Practices in Prominent Projects

Research into the documentation practices of open-source software development (OSSD) projects, which often involve the creation and maintenance of large and distributed systems, reveals that the use of UML is not a consistently mandated practice and tends to vary considerably from one project to another [12]. Unlike commercial software development environments where specific processes and documentation standards might be enforced, the adoption of UML in OSSD is typically driven by the individual project's community and their collective perception of its necessity and benefits [12]. In this context, the decision to utilize UML often stems from a recognized need within the developer community for enhanced communication and a desire to codify high-level architectural knowledge in a visual and standardized format [12]. The voluntary nature of contributions in open-source projects likely plays a significant role in shaping documentation choices, with UML being embraced when the developers themselves deem it a valuable tool for their specific project's needs and communication goals. This provides a real-world perspective on how UML is adopted and utilized in the absence of strict organizational mandates, reflecting its perceived utility by practitioners in the field of large, often distributed, software systems.

3.2 Types of UML Diagrams Commonly Found in Open-Source Documentation

Studies examining open-source software development projects indicate that certain types of UML diagrams are more frequently encountered in their documentation than others [2]. Among the most commonly used are class diagrams, which are effective in depicting the static structure of a system by illustrating classes, their attributes, and methods, and use case diagrams, which are valuable for outlining the functional requirements of the system and how users interact with it [2]. Additionally, component diagrams and package diagrams are often employed to visualize the decomposition of the system into modular components and to show the dependencies between these components and packages, providing a high-level view of the system's organization [3]. While sequence diagrams, which model the interactions between objects over time, and activity diagrams, which illustrate process flows and workflows, are sometimes used to explain specific features or intricate operational sequences, their prevalence in open-source documentation appears to be less than that of the structural diagram types [13]. This observed preference for structural diagrams in the documentation of open-source projects suggests that UML might be perceived as particularly beneficial for conveying the fundamental architecture and high-level design of distributed systems, focusing on the system's static organization and primary functionalities rather than detailing every aspect of its dynamic behavior across all possible scenarios.

3.3 The Extent of UML Diagram Correspondence with Implementation Code

An interesting observation from research on UML usage in open-source software development projects is that the UML diagrams found in their documentation often do not maintain a complete and one-to-one correspondence with the actual implementation code [12]. For instance, studies have noted that the number of classes depicted in UML class diagrams is typically less than the total number of classes that exist within the project's source code [12]. This discrepancy suggests that UML, in this context, might be primarily utilized for representing abstract, high-level design concepts or for illustrating specific, key aspects of the system's architecture, rather than serving as an exhaustive and precise visual representation of the entire codebase [12]. This phenomenon reinforces the idea that in practice, particularly within the dynamic and often volunteer-driven environment of open-source projects, UML may function more as a tool for communication and conceptualization of the system's essential structure and functionality, rather than as a form of "living documentation" that is rigorously kept in perfect synchronization with every detail of the implementation. The challenges associated with maintaining such a high level of fidelity between diagrams and code, especially in large and actively evolving systems, likely contribute to this more pragmatic approach to UML usage.

4. UML as a Communication Tool in Development Teams for Large Distributed Systems

4.1 Facilitating Understanding and Collaboration Among Stakeholders

One of the primary intentions behind the development of UML was to establish it as a robust communication tool capable of bridging the gap between various stakeholders involved in software development [2]. This includes not only the technical team members such as developers and designers but also testers, business analysts, and even clients or end-users. By providing a standardized visual language, UML aims to simplify complex technical concepts and intricate system relationships, making them more readily understandable to individuals who may not possess a deep technical background [4]. UML diagrams can serve as a central point of reference, helping teams to align their understanding of the system's intended design, its functionalities, and the overall progress of the project [15]. The inherent strength of UML in this regard lies in its ability to abstract away the often overwhelming details of implementation code and present a higher-level, more conceptual view of the system's architecture and behavior. This abstraction fosters a shared mental model among team members with diverse roles and expertise, thereby facilitating more effective discussions, informed decision-making, and ultimately, better collaboration throughout the software development lifecycle.

4.2 Specific UML Diagrams for Communication in Distributed Contexts

Within the realm of large distributed systems, certain types of UML diagrams prove to be particularly valuable for communication due to their ability to represent the unique characteristics and complexities of such architectures [3]. Deployment diagrams, for example, are crucial for visualizing the physical deployment of software components across various hardware nodes in a distributed environment, aiding in the understanding of the system's overall topology and the distribution of responsibilities

[3]. Component diagrams are essential for describing the organization of software components and their dependencies on each other, which is vital for comprehending how different parts of a distributed system interact, often over network interfaces [3]. Sequence diagrams are highly useful for modeling the interactions between different objects or components in a time-ordered manner, thereby clarifying the flow of data and control across the distributed system [2]. Lastly, activity diagrams can effectively model the workflows and processes that span across multiple components of a distributed system, including the representation of parallel processes and synchronization points that are common in such architectures [4]. These specific UML diagram types, therefore, retain significant relevance as communication tools by providing visual representations tailored to the distinct aspects and challenges of distributed systems.

4.3 The Role of UML Compared to Other Communication Methods in Distributed Teams

Development teams working on large distributed systems typically rely on a diverse array of communication tools and methods to coordinate their efforts effectively [17]. These commonly include asynchronous communication channels such as email and various chat platforms like Slack, as well as synchronous methods like video conferencing for meetings and discussions. Additionally, project management tools like Jira and Trello are frequently used to track tasks, share status updates, and manage workflows across distributed team members [17]. Effective communication within distributed teams often emphasizes the importance of over-communicating to compensate for the lack of face-to-face interaction, strategically leveraging different tools based on the context and urgency of the information, and making a conscious effort to schedule virtual "face time" to foster better team cohesion [17]. In this landscape, UML diagrams can play a complementary role by offering a visual and standardized means to represent the system's design and architecture [22]. While UML can be particularly useful for communicating complex design ideas asynchronously, especially in teams spread across different time zones, it is unlikely to entirely replace the need for regular verbal and written communication, which allows for more nuanced discussions, immediate clarifications, and the conveyance of rationale behind design decisions [18]. Therefore, UML should be viewed as one valuable tool within a broader communication toolkit for distributed teams, with its effectiveness contingent on its thoughtful integration with other communication practices and the team's overall proficiency in utilizing it.

5. UML in Academia: An Analysis of Computer Science Curricula

at Top Universities

5.1 UMass Lowell: Software Design Principles and Potential Implicit Inclusion

The undergraduate Computer Science curriculum at UMass Lowell offers students a range of specialized options, including General Computer Science, Cybersecurity, Data Science, and Bio-Cheminformatics [23]. While a review of the publicly available course descriptions for these options [26] does not explicitly mention the Unified Modeling Language by name, it is plausible that the principles and practices of visual modeling, potentially including UML concepts, are implicitly incorporated within courses that focus on broader software design principles. For example, project-based courses, which often require students to design and implement software systems, might necessitate the use of visual design tools and methodologies to conceptualize and communicate their designs. In such contexts, instructors might introduce or expect students to utilize UML diagrams as a standard way to represent system architecture and behavior, even if the course description itself does not specifically list UML as a topic. Therefore, while explicit references to UML are absent in the initial review of the curriculum, its underlying concepts and practical application could still be a part of the educational experience, particularly within courses aimed at developing software design and architecture skills.

5.2 Stanford University: Explicit Coverage in Database and Software Engineering Courses

In contrast to UMass Lowell, the Computer Science curriculum at Stanford University explicitly acknowledges and integrates UML into specific courses [28]. Notably, the course "Databases: Modeling and Theory" directly addresses the data-modeling component of UML, exploring how UML diagrams can be used to represent database schemas and how they are subsequently translated into relational database designs [28]. This indicates a recognition of UML's continued relevance in the domain of database systems. Furthermore, at the graduate level, the course CS446 is dedicated to providing an overview of essential software engineering principles, with a specific focus on modeling software systems using UML as a key topic of instruction [31]. At the undergraduate level, courses such as CS108, titled "Object-Oriented Systems Design," likely incorporate UML as a fundamental aspect of teaching object-oriented design principles and methodologies [30]. The inclusion of UML in these diverse courses, spanning database theory, software engineering, and object-oriented design, demonstrates a clear emphasis on the importance of visual modeling and UML as a valuable skill for computer science students at Stanford. This suggests that the university considers UML a relevant and practical tool for both understanding and designing software systems in specific contexts.

5.3 Carnegie Mellon University: UML as Part of Software Construction and Design

Carnegie Mellon University's School of Computer Science also integrates UML into its curriculum, particularly within courses focused on software construction and design [32]. The course "Principles of Software Construction: Objects, Design and Concurrency" includes a dedicated module that covers the essentials of UML, aptly titled "Just enough UML" [33]. This inclusion suggests that UML is considered a relevant tool for teaching the fundamental principles of software construction and design. Additionally, the course 17-214, titled "Applying UML and Patterns," explicitly focuses on both UML and design patterns, indicating a deeper exploration of these concepts within the curriculum [32]. Moreover, the Saylor Academy's software engineering course, which utilizes materials developed by Carnegie Mellon University, also incorporates UML as part of its teaching on software modeling [34]. The presence of UML in these various educational contexts within and associated with Carnegie Mellon underscores its perceived value in equipping computer science students with practical skills in software design and communication through visual modeling techniques. This reinforces the idea that UML continues to be considered a significant component of software engineering education at this institution.

5.4 Massachusetts Institute of Technology (MIT): Less Explicit Mention in Core CS Courses

A review of the core Computer Science course descriptions available in the MIT course catalog does not reveal explicit mentions of the Unified Modeling Language [35]. The curriculum at MIT appears to place a strong emphasis on foundational concepts such as algorithms, data structures, theoretical computer science, and system design, with course descriptions focusing primarily on these areas. While courses related to software design are offered, the provided snippets do not indicate a specific focus or requirement on learning or using UML. It is possible that the principles of visual modeling and system representation are covered within these courses, but perhaps through different notations or with a less explicit emphasis on the UML standard itself. The absence of direct references to UML in the core Computer Science curriculum descriptions at MIT might suggest a different pedagogical approach or a prioritization of other fundamental computer science concepts over the explicit teaching of UML. This observation highlights that while software design principles are undoubtedly important in computer science education, the specific tools and notations emphasized can vary among top universities.

5.5 Technische Universität München (TUM) (via edX): Explicit Inclusion in Software

Engineering Essentials

The "Software Engineering Essentials" course offered by Technische Universität München (TUM) through the edX platform explicitly includes the Unified Modeling Language as a key component of its curriculum [38]. The course description lists "Unified Modeling Language (UML)" as one of the core topics that participants will learn and apply. This explicit inclusion of UML as a fundamental aspect of software engineering education at TUM demonstrates its continued relevance in the academic context, particularly in a European university setting. The course aims to teach students how to apply UML modeling, along with other essential software engineering practices like agile methods, object-oriented programming, and project management techniques, in the development of complex software systems. This indicates that UML is still considered a valuable and necessary skill for aspiring software engineers at TUM, highlighting its enduring importance in the field of software engineering education internationally.

Table 1: UML Coverage in Computer Science Curricula at Selected Universities (Section 5)

University	Explicit UML Mention	Relevant Courses
UMass Lowell	No	Project Courses (potential implicit inclusion)
Stanford University	Yes	Databases: Modeling and Theory, Software Engineering (CS446), Object-Oriented Systems Design (CS108)
Carnegie Mellon	Yes	Principles of Software Construction, Applying UML and Patterns (17-214), Software Engineering (via Saylor Academy)
MIT	No	Software Design courses (potential implicit inclusion)

TUM (via edX)	Yes	Software Engineering Essentials

6. Advantages and Disadvantages of Using UML in Modern Distributed Systems Development

6.1 Advantages of UML in Distributed System Contexts

Despite the challenges and criticisms, UML offers several distinct advantages when applied to the development of modern distributed systems [3]. One of the most significant benefits is the ability of UML diagrams to provide a clear visualization of the inherent complexity of distributed systems [3]. By representing the architecture, components, and interactions visually, UML can simplify understanding for all stakeholders involved. Furthermore, UML serves as a standardized visual language, which enhances communication and collaboration across diverse teams working on distributed systems, including architects, developers, and operations personnel [3]. Specific UML diagram types are particularly advantageous in this context. Deployment and component diagrams are crucial for defining and standardizing the architecture of distributed solutions, illustrating the physical distribution of applications and their interdependencies [3]. Sequence and activity diagrams are effective in modeling the dynamic behavior and process flows within and between distributed components, aiding in the comprehension of data flow and control mechanisms [2]. Additionally, UML can assist in planning for scalability and ensuring consistent designs that are easier for multiple developers and new team members to understand and follow [14]. Finally, up-to-date UML documentation, especially class and sequence diagrams, has been shown to improve code comprehension, reduce maintenance time, and facilitate collaboration during software maintenance efforts [2]. These advantages highlight that, while not without its drawbacks, UML can still provide significant value in the development and documentation of intricate distributed systems.

6.2 Disadvantages and Limitations of UML for Modern Distributed Systems

Despite its benefits, the use of UML in the context of modern distributed systems development also presents several disadvantages and limitations [7]. A primary concern is the time and effort required to create and, more importantly, maintain detailed UML diagrams, especially for large and rapidly evolving distributed systems [7]. Given the dynamic nature of these systems and the prevalence of Agile development methodologies, UML diagrams can quickly become outdated and inconsistent with the actual codebase, diminishing their utility and potentially leading to confusion [2]. The comprehensive nature of UML, with its various diagram types and notations, can also be complex and presents a steep learning curve for some team members, potentially hindering its widespread adoption [7]. For smaller or less complex distributed systems, the formality and level of detail offered by UML might be perceived as excessive and unnecessary overhead [7]. Furthermore, the interpretation of UML diagrams can sometimes be subjective, leading to potential ambiguities and misunderstandings within a team [7]. UML may also lack specific constructs needed to effectively model certain critical aspects of distributed systems, such as user interfaces, serialization protocols, mechanisms for object persistence, and fault tolerance strategies [39]. The emphasis on upfront design and extensive documentation often associated with traditional UML usage can also clash with the iterative and emergent nature of Agile development practices commonly employed for distributed systems [9]. Finally, creating UML diagrams that can effectively represent the sheer scale and complexity of very large distributed systems, with their numerous components and intricate interconnections, can be challenging and may result in diagrams that are too cumbersome to be practically useful [40]. These disadvantages underscore the need for a careful and balanced approach to using UML in modern distributed systems development, considering its limitations alongside its potential benefits.

Table 2: Advantages and Disadvantages of UML for Distributed Systems (Section 6)

Category	Points	Supporting Snippets
Advantages	Visualization of complexity, enhanced communication, architectural blueprinting (deployment & component diagrams), modeling dynamic behavior (sequence & activity diagrams), consistency and scalability planning, improved code comprehension and maintenance.	[2]
Disadvantages	Time and effort overheads, potential for diagrams to become outdated, complexity and learning curve, overkill for certain projects, ambiguity in interpretation, limited support for specific distributed system	[2]

	concerns (UI, serialization, persistence, fault tolerance), clash with Agile principles, scalability issues with diagrams for very large systems.	

7. Alternative Modeling and Documentation Techniques

In response to the challenges and limitations associated with UML, especially in the context of large distributed systems, several alternative modeling and documentation techniques have gained prominence [9]. In Agile environments, where speed and flexibility are paramount, lightweight tools such as simple flowcharts, Kanban boards for visualizing workflows, and even the use of sticky notes for rapid brainstorming and conceptualization are often favored over the more formal and detailed UML diagrams [9]. For conveying high-level architectural overviews, many practitioners prefer using basic block diagrams, which are often less constrained by formal standards and can be quickly adapted to specific needs, rather than adhering strictly to UML component diagrams [11]. Furthermore, in many modern development practices, the source code itself is increasingly viewed as the most accurate and up-to-date form of documentation, with developers sometimes finding well-written and commented code, along with comprehensive API specifications, to be more informative than external UML diagrams, which can easily fall out of sync with the implementation [11]. For specific aspects of distributed systems, such as performance evaluation, techniques like queueing network models, computation structure models, and hierarchical performance modeling are employed to analyze communication and computation delays caused by the system's architecture [41]. To address the complexities of distributed system behavior and ensure correctness, formal modeling languages like TLA+ and lightweight simulation techniques are utilized to describe and verify system properties at various levels of abstraction [44]. In the domain of machine learning, which often involves large distributed systems for training models, techniques like data parallelism and model parallelism have their own distinct architectural considerations and documentation needs [43]. Lastly, in service-oriented architectures (SOA), domain-specific modeling languages and model weavers are used to define the interactions and relationships between loosely coupled services [45]. These alternative techniques reflect a trend towards more specialized, lightweight, or code-centric approaches to modeling and documenting large distributed systems, often chosen based on the specific needs and context of the project.

8. Case Studies: Real-World Examples of UML Usage (or

Avoidance)

While explicit case studies detailing extensive UML usage in contemporary large distributed systems are not prominently featured in the provided material, several instances and research directions offer insights into its application [42]. The PERMABASE project, for example, explored the adaptation of UML for specifying the physical environment of distributed systems, indicating a past effort to leverage UML's visual modeling capabilities in this domain [53]. Books like "Designing Concurrent, Distributed, and Real-Time Applications with UML" utilize case studies to illustrate the application of UML-based methodologies, such as COMET, for designing these types of complex systems [50]. While these examples might not represent the latest trends, they highlight historical applications of UML in distributed system design. Real-world examples of prevalent large distributed systems, such as Netflix, Amazon, Uber, and Airbnb, are often cited in discussions about distributed architectures [49]; however, the snippets do not provide specific details regarding their use of UML in their development processes. Research focusing on open-source projects, which frequently involve large and distributed systems, offers some empirical evidence of UML usage, showing a tendency towards employing class and use case diagrams for high-level conceptualization and knowledge sharing within the community [12]. Additionally, a thesis documents the application of UML for modeling a complex, software-intensive simulation for a naval "digital ship," which can be considered a form of a distributed system, showcasing a specific instance of UML's utility in a sophisticated and intricate domain [54]. These examples, while varied in context and recency, suggest that UML's application in large distributed systems is not entirely absent but might be more selective and context-dependent in modern software development practices.

9. Conclusion: The Relevance of UML in the Age of Distributed Systems

The analysis of online discussions, open-source documentation practices, team communication methods, university curricula, advantages and disadvantages, alternative techniques, and case studies suggests that the relevance of UML in the development and documentation of large distributed software systems is nuanced and has evolved over time. While UML may not be the universally adopted standard it once was, particularly in the face of Agile methodologies and the increasing complexity and dynamism of distributed systems, it continues to hold value in specific contexts. Certain UML diagram types, such as deployment, component, sequence, and activity diagrams, remain useful for visualizing and communicating specific aspects of distributed system architecture and behavior.

The rise of Agile has led to a preference for lightweight and informal visualizations in many cases, and the challenges of maintaining UML diagrams in rapidly evolving systems are significant. Consequently, practitioners often favor simpler tools or rely more directly on code and API specifications for detailed understanding. However, research on open-source projects indicates that UML, especially class and use case diagrams, is still used for high-level design and communication when deemed beneficial by the community.

UML's role as a communication tool remains relevant, particularly for fostering a shared understanding among diverse stakeholders. While distributed teams rely on a variety of communication methods, UML can complement these by providing a standardized visual representation of system design. The presence of UML in the curricula of several top universities, including Stanford, Carnegie Mellon, and TUM, indicates its continued importance in software engineering education, suggesting that future generations of developers will still be familiar with its principles and applications.

The advantages of UML in visualizing complexity, enhancing communication, and aiding in architectural blueprinting, especially for distributed systems, cannot be overlooked. However, the disadvantages, including time overhead, the potential for diagrams to become outdated, and its limitations in modeling certain distributed system concerns, necessitate a judicious and balanced approach to its use. Alternative modeling and documentation techniques have emerged, reflecting a trend towards more specialized, lightweight, or code-centric approaches.

Case studies, while not overwhelmingly abundant for modern large distributed systems, still point to instances where UML has been successfully applied, particularly in architectural design and system modeling.

In conclusion, UML is not obsolete in the realm of large distributed software systems, but its application is more selective and strategic than it might have been in the past. Its relevance lies in its ability to provide standardized visual representations for specific aspects of system design and communication, particularly when the benefits outweigh the costs of creation and maintenance. The decision to use UML should be driven by a careful consideration of the project's needs, the team's expertise, and the specific communication and documentation goals.

Works cited

1. Unified Modeling Language - Wikipedia, accessed March 23, 2025, https://en.wikipedia.org/wiki/Unified_Modeling_Language
2. Unified Modeling Language (UML) and Its Contribution to Software Maintenance Efficiency, accessed March 23, 2025, https://www.researchgate.net/publication/389585787_Unified_Modeling_Language_UML_and_Its_Contribution_to_Software_Maintenance_Efficiency
3. What is Unified Modeling Language (UML)? - Visual Paradigm, accessed March 23, 2025, https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
4. Unified Modeling Language (UML) Diagrams - GeeksforGeeks, accessed March 23, 2025, https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/
5. Introduction to UML. do you want to improve the software... | by Amina Rafaqat - Medium, accessed March 23, 2025, https://medium.com/@aminarafaqat12345678/introduction-to-uml-cf83bfeef950
6. UML diagrams vs. simulation in software design, accessed March 23, 2025, https://softwaresim.com/blog/uml-diagrams-vs.simulation-in-software-design/
7. Advantages and Disadvantages of UML: Explained in Detail - The Knowledge Academy, accessed March 23, 2025, https://www.theknowledgeacademy.com/blog/advantages-and-disadvantages-of-uml/
8. What are the advantages and disadvantages of UML? - GeeksforGeeks, accessed March 23, 2025, https://www.geeksforgeeks.org/what-are-the-advantages-and-disadvantages-of-uml/
9. Does It Still Have a Place?. Ah, UML — the Unified Modeling... | by Dave LumAI - Medium, accessed March 23, 2025, https://medium.com/@DaveLumAI/uml-does-it-still-have-a-place-71be1330ab31
10. Is UML Still Relevant Today? How Is it Used in an Agile Environment?, accessed March 23, 2025, https://managedagile.com/is-uml-still-relevant-today/
11. Why isn't UML actually used in industry? : r/AskComputerScience - Reddit, accessed March 23, 2025, https://www.reddit.com/r/AskComputerScience/comments/1dmmq9/why_isnt_uml_actually_used_in_industry/
12. UML Usage in Open Source Software Development : A Field Study - CEUR-WS, accessed March 23, 2025, https://ceur-ws.org/Vol-1078/paper3.pdf
13. UML usage in open source software development : A field study - ResearchGate, accessed March 23, 2025, https://www.researchgate.net/publication/289640651_UML_usage_in_open_source_software_development_A_field_study
14. UML for Distributed System - Codemia, accessed March 23, 2025, https://codemia.io/knowledge-hub/path/uml_for_distributed_system
15. What is a UML Diagram? | Different Types and Benefits - Miro, accessed March 23, 2025, https://miro.com/diagramming/what-is-a-uml-diagram/

16. Modeling a Distributed System Using Deployment Diagram - Visual Paradigm Guides, accessed March 23, 2025, https://guides.visual-paradigm.com/modeling-a-distributed-system-using-deployment-diagram/
17. Distributed Engineering Teams: Best Practices - Integrio Systems, accessed March 23, 2025, https://integrio.net/blog/distributed-engineering-teams-best-practices
18. 10 Communication Tips For Distributed And Offshore Teams - Praxent, accessed March 23, 2025, https://praxent.com/blog/distributed-offshore-team-communication-tips
19. Effective Distributed Team Communication Guide | Best Practices & Tips - Web Help Agency, accessed March 23, 2025, https://webhelpagency.com/blog/a-guide-on-effective-distributed-team-communication/
20. Communication Tools for Distributed Software Development Teams - RTI International, accessed March 23, 2025, https://www.rti.org/sites/default/files/resources/rti_cpr07_virtualteam.pdf
21. Effective Communication in Globally Distributed Scrum: A Model and Practical Guidance, accessed March 23, 2025, https://ajis.aaisnet.org/index.php/ajis/article/view/4501
22. On the Usage of UML Diagrams in Open Source Projects - USI – Informatics, accessed March 23, 2025, https://www.inf.usi.ch/lanza/Downloads/MSc/Rome2023a.pdf
23. Computer Science Major | Miner School of Computer and Information Sciences | Kennedy College of Sciences | UMass Lowell, accessed March 23, 2025, https://www.uml.edu/sciences/computer-science/programs/ugrad/computer-science-major.aspx
24. Computer Science Major | UMass Lowell, accessed March 23, 2025, https://www.uml.edu/catalog/undergraduate/sciences/departments/computer-science/computer-science-major.aspx
25. Computer Science Major | Miner School of Computer and Information Sciences | Kennedy College of Sciences | UMass Lowell, accessed March 23, 2025, https://www.uml.edu/sciences/computer-science/programs/ugrad/computer-science-major.aspx/1000
26. Computer Science | UMass Lowell, accessed March 23, 2025, https://www.uml.edu/catalog/undergraduate/sciences/departments/computer-science/degree-pathways/dp-cs-data-science-2020.aspx
27. Computer Science | UMass Lowell, accessed March 23, 2025, https://www.uml.edu/catalog/undergraduate/sciences/departments/computer-science/degree-pathways/dp-cs-general-2020.aspx
28. StanfordOnline: Databases: Modeling and Theory - edX, accessed March 23, 2025, https://www.edx.org/learn/databases/stanford-university-databases-modeling-and-theory
29. StanfordOnline: Databases: Modeling and Theory - edX, accessed March 23,

2025, https://www.edx.org/course/modeling-and-theory?source=aw&awc=6798_1606479168_ba4acf678a5d8fef1d86aa2ee0e686a3&utm_source=aw&utm_medium=affiliate_partner&utm_content=text-link&utm_term=539905_Benzinga
30. Computer Science Department Courses | Stanford University Bulletin, accessed March 23, 2025, https://bulletin.stanford.edu/departments/COMPUTSCI/courses
31. Description of course CS446 - Stanford InfoLab, accessed March 23, 2025, http://infolab.stanford.edu/cs446/
32. 17-214: Principles of Software System Construction, accessed March 23, 2025, https://cmu-17-214.github.io/
33. Principles of Software Construction: Objects, Design and Concurrency Just enough UML, accessed March 23, 2025, https://www.cs.cmu.edu/~charlie/courses/15-214/2014-spring/slides/09b-uml.pdf
34. CS302: Software Engineering | Saylor Academy, accessed March 23, 2025, https://learn.saylor.org/course/view.php?id=788
35. Electrical Engineering and Computer Science (Course 6) - MIT Bulletin, accessed March 23, 2025, https://catalog.mit.edu/subjects/6/
36. Course 6: Electrical Engineering and Computer Science IAP/Spring 2025, accessed March 23, 2025, https://student.mit.edu/catalog/m6a.html
37. Computer Science and Engineering (Course 6-3) - MIT Bulletin, accessed March 23, 2025, https://catalog.mit.edu/degree-charts/computer-science-engineering-course-6-3/
38. TUMx: Software Engineering Essentials - edX, accessed March 23, 2025, https://www.edx.org/learn/software-engineering/technische-universitat-munchen-software-engineering-essentials
39. UML Advantages and Disadvantages | PDF | Unified Modeling Language - Scribd, accessed March 23, 2025, https://www.scribd.com/document/211538904/UML-advantages-and-disadvantages
40. Representing Software Architectures For Large Scale Systems: The Layered Package Diagram, accessed March 23, 2025, https://www.crystalclearsoftware.com/publications/2001/RepresentingArchitectures.html
41. Hierarchical Performance Modeling for Distributed System Architectures - ResearchGate, accessed March 23, 2025, https://www.researchgate.net/publication/232650518_Hierarchical_Performance_Modeling_for_Distributed_System_Architectures
42. USING UML TO MODEL DISTRIBUTED SYSTEM ARCHITECTURES - Mara Nikolaidou, accessed March 23, 2025, https://mara.dit.people.hua.gr/publications/caine05.pdf
43. Distributed Training: Guide for Data Scientists - neptune.ai, accessed March 23, 2025, https://neptune.ai/blog/distributed-training
44. Curated list of resources on testing distributed systems - Andrey Satarin, accessed March 23, 2025, https://asatarin.github.io/testing-distributed-systems/

45. Model-driven Generative Techniques for Scalable Performabality Analysis of Distributed Systems - Jeff Gray, accessed March 23, 2025, https://gray.cs.ua.edu/pubs/ngs06.pdf
46. How We Use Formal Modeling, Lightweight Simulations, and Chaos Testing to Design Reliable Distributed Systems | Datadog, accessed March 23, 2025, https://www.datadoghq.com/blog/engineering/formal-modeling-and-simulation/
47. DISTRIBUTED COMPUTING AND MODELING & SIMULATION: SPEEDING UP SIMULATIONS AND CREATING LARGE MODELS, accessed March 23, 2025, https://www.informs-sim.org/wsc11papers/014.pdf
48. Distributed System Simulation Methods For Model-Based Product Development - DiVA portal, accessed March 23, 2025, https://www.diva-portal.org/smash/get/diva2:872653/FULLTEXT01.pdf
49. Distributed Architecture: 4 Types, Key Elements + Examples - vFunction, accessed March 23, 2025, https://vfunction.com/blog/distributed-architecture/
50. Designing Concurrent, Distributed, and Real-Time Applications With Uml - Amazon.com, accessed March 23, 2025, https://www.amazon.com/Designing-Concurrent-Distributed-Real-Time-Applications/dp/0201657937
51. Examples and Applications of Distributed Systems in Real-Life - GeeksforGeeks, accessed March 23, 2025, https://www.geeksforgeeks.org/examples-and-applications-of-distributed-systems-in-real-life/
52. Using UML to Model Distributed System Architectures. - ResearchGate, accessed March 23, 2025, https://www.researchgate.net/publication/220923234_Using_UML_to_Model_Distributed_System_Architectures
53. UML specification of distributed system environments, accessed March 23, 2025, http://csis.pace.edu/~marchese/CS865/Papers/content.pdf
54. Combat System Modeling: Modeling Large-Scale Software and Hardware Application Using UML - VTechWorks, accessed March 23, 2025, https://vtechworks.lib.vt.edu/server/api/core/bitstreams/ce03578a-21b1-45ec-b9f5-36600e7c2c9a/content