International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 14, Issue 2, 2026

Live Code Share: Real Time Code Collaborator

P Ganesh Kumar[1], Hamsika Karnakota[2], Lakshmi Chetana Kattoju[3]

[1]Assistant Professor; Department Of Information Technology Bhoj Reddy Engineering College For Women Hyderabad India

[2,3]B.Tech Students; Department Of Information Technology Bhoj Reddy Engineering College For Women Hyderabad India

Mail Id; karnakotahamsika@gmail.com ,lakshmichetana11@gmail.com

Abstract

Modern software development frequently involves geographically distributed teams that require continuous coordination while working on shared source code. Conventional workflows based on local development environments, periodic version control commits, and screen-sharing tools often introduce communication delays, merge conflicts, and inefficient collaboration. This paper presents ***Live Code Share****, a cloud-based real-time collaborative coding platform designed to support simultaneous multi-user programming through a browser interface.*

The system is implemented using the MERN technology stack, comprising React.js for the client interface, Node.js with Express.js for backend services, and MongoDB for persistent data storage. Real-time synchronization is achieved through Socket.IO, enabling low-latency bidirectional communication between connected users. The platform provides room-based session management, synchronized editing, code execution support, and shared output visibility.

The proposed solution offers a lightweight and accessible environment suitable for pair programming, technical interviews, classroom instruction, hackathons, and remote team development. Experimental observations indicate reduced coordination overhead and improved collaboration efficiency compared with traditional asynchronous workflows.

Keywords

Real-time collaboration, Collaborative coding, MERN stack, WebSockets, Socket.IO, React.js, Node.js, Cloud IDE, Multi-user editing

Introduction

Software engineering practices have evolved significantly with the rise of distributed teams, remote education, and global development environments. Developers increasingly require tools that allow simultaneous interaction with shared code resources regardless of physical location. However, conventional approaches based on standalone integrated development environments and version control systems are primarily asynchronous in nature. Although these tools effectively maintain source history, they do not natively support live multi-user editing.

As a result, teams often depend on video conferencing and screen-sharing applications to discuss code changes. Such methods permit observation but usually restrict active editing control to a single participant. This limitation decreases productivity, slows decision-making, and creates communication barriers.

To address these challenges, this paper introduces **Live Code Share**, a browser-accessible collaborative programming platform that enables multiple users to edit and review code simultaneously. The application employs WebSocket communication for real-time synchronization and provides isolated collaboration rooms through unique session identifiers. By combining responsiveness, accessibility, and scalability, the platform delivers an effective solution for modern collaborative software development.

Related Work

The development of an effective real-time code collaboration platform requires a comprehensive understanding of current collaborative editing systems, communication technologies, and scalable cloud architectures. A detailed survey of existing research and industrial solutions was conducted to analyze synchronization methods, user interaction models, and implementation strategies. The primary objective of this study is to identify proven techniques and limitations that can guide the design of an improved collaborative coding environment [1].

One of the most influential examples of collaborative systems is found in shared document platforms such as Google Docs and integrated programming tools like Visual Studio Code Live Share. These platforms demonstrate how multiple users can edit the same content simultaneously while maintaining consistency across all connected clients. To achieve this, advanced synchronization models such as Operational Transformation (OT) and Conflict-Free Replicated Data Types (CRDTs) are commonly employed. These approaches minimize conflicts, preserve document integrity, and support smooth concurrent editing with low delay [2].

Modern browser-based code editors such as CodeMirror and Monaco Editor have significantly improved the user experience of online

programming environments. These editors provide essential features including syntax highlighting, automatic indentation, code completion, bracket matching, and support for multiple programming languages. Their modular design allows seamless integration with real-time communication frameworks such as Socket.IO, which enables fast event-driven synchronization between users and servers [3].

From the server-side perspective, many scalable collaborative applications are built using the MERN stack, consisting of MongoDB, Express.js, React.js, and Node.js. React.js enables responsive and component-based frontend development, while Express.js simplifies backend API management. Node.js is especially suitable for real-time systems because of its asynchronous, event-driven runtime, which efficiently handles numerous simultaneous connections. MongoDB offers flexible document-oriented storage for session details, user records, shared files, and collaboration metadata [4].

Maintaining consistency during simultaneous editing remains one of the major technical challenges in collaborative systems. When multiple users modify the same section of code concurrently, conflicts may occur due to overlapping operations or delayed network transmission. OT and CRDT techniques are widely adopted to address these issues, though challenges such as fault tolerance, state recovery, and ordering of remote operations still require careful design. Some systems additionally integrate lightweight version management features inspired by distributed repositories, enabling rollback and change tracking [5].

Security is another critical component of cloud-based collaboration platforms. Since multiple users interact within shared environments, authentication and authorization mechanisms must be implemented to prevent unauthorized access. Technologies such as JSON Web Tokens (JWT) are frequently used for secure login sessions, while Role-Based Access Control (RBAC) helps define permissions for hosts, editors, and viewers. For systems supporting code execution, sandbox environments are often deployed to isolate user programs and protect server resources [6].

Recent developments in collaborative programming platforms include the integration of artificial intelligence tools for productivity enhancement. AI-based assistants can generate code suggestions, identify syntax errors, recommend optimizations, and support debugging in real time. Such capabilities improve efficiency and reduce development time. In addition, cloud deployment services such as Vercel and Render provide scalable infrastructure for hosting modern frontend and backend applications with minimal operational complexity [7].

In summary, previous research and existing commercial tools demonstrate that successful collaborative systems depend on efficient synchronization algorithms, responsive interfaces, secure access control, and scalable cloud deployment. By combining these established concepts with modern web technologies, the proposed Real-Time Code Collaborator seeks to provide a reliable, secure, and user-friendly environment for simultaneous multi-user programming with high consistency and low latency [8].

Architecture

System Architecture

System architecture represents the overall structural design of the proposed Real-Time Code Collaborator and illustrates how individual components interact to achieve the desired functionality. It provides a high-level view of the system by defining the relationship between users, client interfaces, server modules, databases, and external services. The purpose of system architecture is to establish a clear and organized framework that supports efficient implementation, maintenance, and future expansion.

In the proposed platform, users access the application through a web browser and interact with the collaborative editor interface. The client-side application communicates with the backend server through HTTP requests for standard operations and WebSocket connections for real-time synchronization. The server processes user requests, manages collaboration rooms, coordinates editing events, and handles code execution requests. Persistent data such as user information, room details, and session history are stored in the database.

The architecture ensures that all modules function together in a coordinated manner while satisfying both functional and non-functional requirements. Functional requirements include user registration, room creation, code editing, synchronization, and output generation. Non-functional requirements include low response time, scalability, reliability, and security.

A well-defined architecture also assists in identifying design risks at an early stage, improving planning and reducing implementation complexity. It serves as a blueprint for developers by guiding the integration of frontend technologies, backend services, database systems, and cloud deployment resources.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

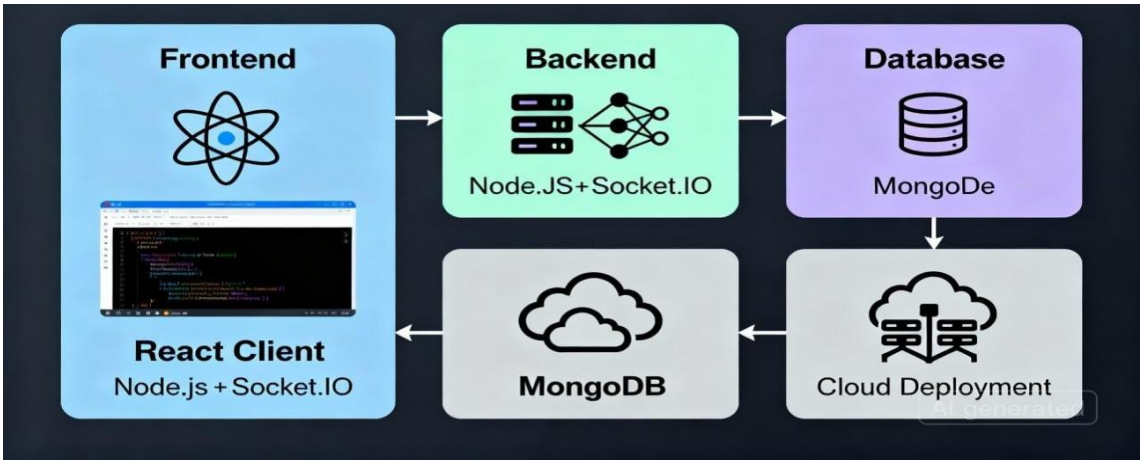


Fig.1 System Architecture

Technical Architecture

The technical architecture of the Real-Time Code Collaborator is designed to provide a responsive, scalable, and secure environment for simultaneous multi-user programming. It combines modern web technologies with event-driven communication mechanisms to deliver seamless collaborative editing and code execution.

The frontend layer is developed using React.js along with HTML and CSS. This layer provides an interactive user interface where participants can create rooms, join sessions, write code, observe real-time changes, and view execution results. React.js enables reusable components and fast rendering, improving the overall user experience.

The backend layer is implemented using Node.js and Express.js. Node.js supports asynchronous and non-blocking execution, making it highly suitable for handling many concurrent users. Express.js is used to manage APIs, routing, authentication logic, and room/session operations.

Real-time communication is established using Socket.IO over WebSocket connections. This module enables immediate synchronization of code changes, cursor activity, user presence, and optional chat messages among connected users. Whenever one participant edits the code, the modification is transmitted instantly to all others in the same room.

MongoDB is used as the database layer for storing user credentials, collaboration room data, saved code, session history, and other metadata. Its document-oriented structure offers flexibility and efficient data retrieval for dynamic collaborative applications.

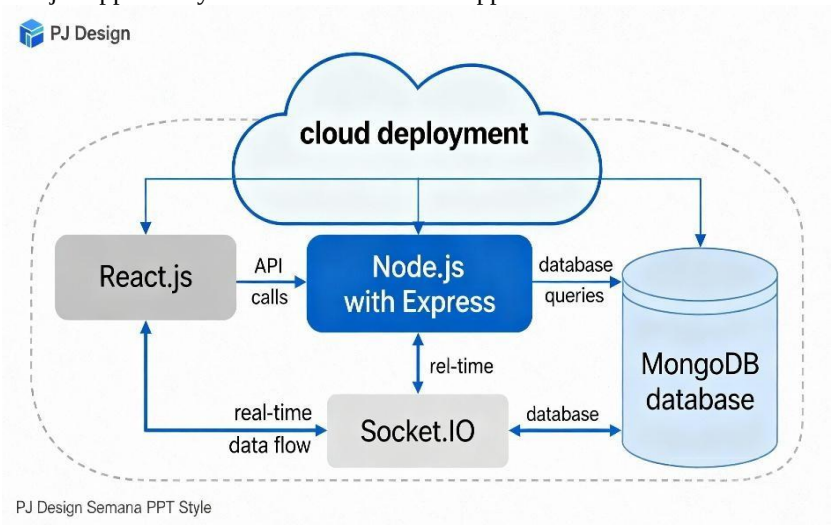


Fig. 2 Technical Architecture

Data Flow Diagram

A Data Flow Diagram (DFD), also known as a bubble chart, is a graphical representation used to describe the movement of data within a system. It illustrates how input data enters the system, how it is processed through various functional components, and how output information is generated. DFDs are widely used during system analysis and design because they provide a clear understanding of data movement and interactions among system elements. In the Real-Time Code Collaborator, the DFD helps model the relationship between users, the collaborative editor, backend services, database storage, and the code execution engine. It identifies the major processes performed by the system, the data stores used, the external entities interacting

with the platform, and the flow of information between them.

The main external entities include users who create or join collaboration rooms, write code, execute programs, and receive synchronized updates. The primary processes include session management, code synchronization, compilation request handling, and result delivery. Data stores include user information, room details, saved code, and collaboration history.

The DFD demonstrates how user input is accepted through the frontend interface, transferred to the backend server, synchronized across connected participants, sent to the compiler service when execution is requested, and returned as output to all users.

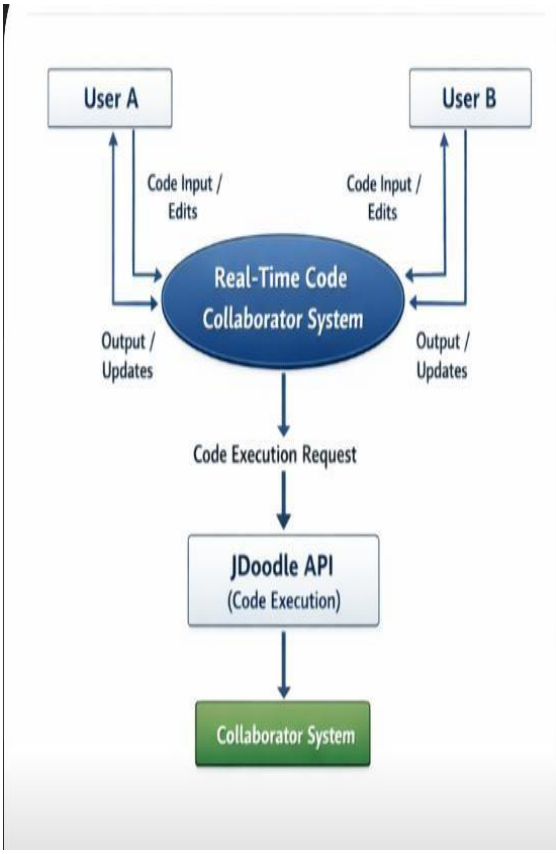


Fig. 3; Data Flow Diagram

Methodology

The Real-Time Code Collaborator is developed to support simultaneous programming activities among multiple users in a shared browser-based environment. The methodology combines collaborative editing mechanisms, low-latency communication, and cloud-based code execution to provide an efficient coding experience.

Initially, the system establishes a collaboration room where users can create a new session or join an existing one using a unique room identifier. Each room functions as an isolated workspace where all participants interact with the same source code file.

The frontend provides an editor interface for writing and modifying code. Whenever a participant performs an edit operation, the change is transmitted to the backend server and instantly broadcast to all other connected users. This synchronization process is implemented using WebSockets through Socket.IO, ensuring minimal delay and continuous consistency among users.

When a user requests code execution, the current source code along with the selected programming language is forwarded to the JDoodle API or an equivalent online compiler service. The compiler executes the code within a secure remote environment and returns output, warnings, or error messages. These responses are displayed to all users in the shared session.

Implementation

Express.js

Express.js is a lightweight and flexible backend web framework used for developing the server-side functionality of the Real-Time Code Collaborator. It is primarily responsible for handling HTTP requests and responses exchanged between the client interface and the database. In the proposed system, Express.js is used to build REST APIs for user authentication, room creation, session management, and communication with external services. It simplifies route handling and middleware integration, which improves code organization and maintainability. Express.js also works efficiently with Node.js and Socket.IO, making it suitable for real-time collaborative applications. Overall, it serves as a core component of the backend architecture.

React.js

React.js is used to develop the frontend user interface of the application. It is a component-based JavaScript library that enables the creation of reusable and maintainable UI elements. In this project, React.js is used to design pages such as login screens, room joining interfaces, code editor layouts, and output consoles. It offers a fast and responsive experience by updating only the components that change, rather than reloading the entire page. React.js also supports state management, which is essential for tracking editor content, connected users, selected programming languages, and real-time events. Its efficient rendering mechanism enhances usability and overall user interaction.

Node.js

Node.js is a server-side JavaScript runtime environment used to execute backend logic outside the browser. It is employed together with Express.js to manage application services, process requests, and coordinate real-time collaboration features. In this project, Node.js handles session creation, user communication, API integration, and event processing. Its asynchronous and non-blocking

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026

architecture allows the system to support many users simultaneously without performance degradation. Node.js is also highly compatible with Socket.IO, enabling smooth real-time communication. It plays a significant role in ensuring scalability and fast server performance.

Socket.IO

Socket.IO is a real-time communication library that enables bidirectional data exchange between the client and the server. It is the primary technology used for live collaboration in this project. Through Socket.IO, code changes made by one participant are instantly transmitted to all other users connected to the same room. It also manages events such as user joining, user leaving, room updates, language selection changes, and output broadcasting. Socket.IO ensures low-latency communication and reliable synchronization even when multiple users interact concurrently. It is the key technology that enables seamless collaborative coding.

Pseudo Code (Paragraph Form)

The implementation begins by initializing the server environment using configuration variables stored in the environment file. Required modules such as Express.js, HTTP, Socket.IO, CORS, Axios, and TypeScript support libraries are imported. An Express application is created, and middleware functions are enabled to allow cross-origin requests and JSON data handling. An HTTP server is then created using the Express application, and Socket.IO is attached to this server to provide real-time communication features.

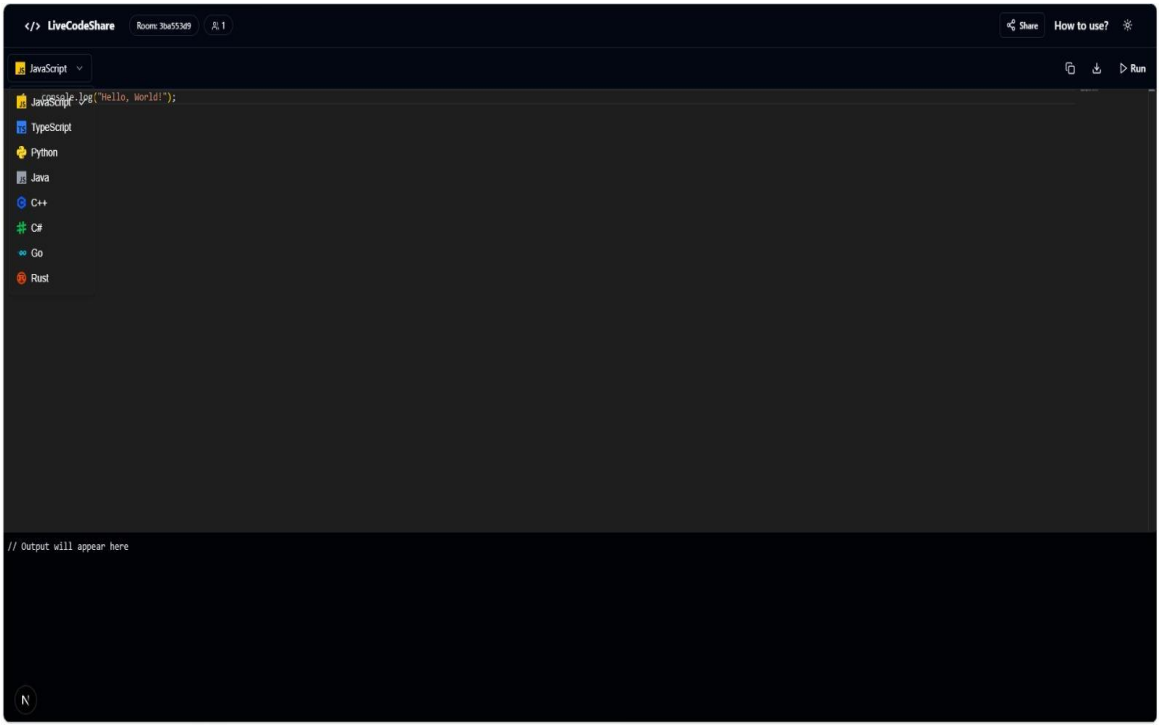
Once the server starts, it listens for incoming socket connections from users. Whenever a new user connects, a unique socket identifier is generated and the connection is registered. The system maintains a room-based structure in which users can join specific collaboration sessions using a room ID. When a participant joins a room, the server adds that user to the room list, broadcasts a notification to other participants, and updates the number of active users in the room.

During collaboration, whenever a user edits the source code, the updated content is sent to the server through a code-change event. The server immediately forwards the modified code to all other users within the same room, ensuring synchronized editing. Similarly, when a participant changes the selected programming language, the updated language preference is broadcast to all connected members of that session.

If any execution failure occurs, such as compilation errors or API communication problems, the server captures the exception and sends an appropriate error message to the room. This ensures users receive immediate feedback.

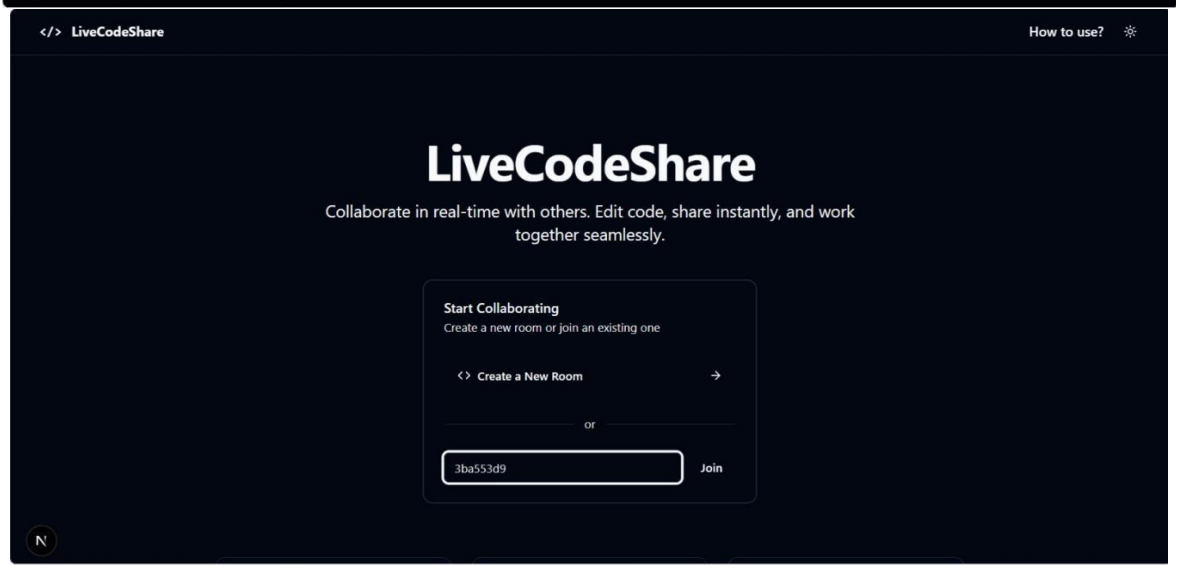
When a user disconnects from the platform, the server removes that participant from the corresponding room. It then informs the remaining users that a participant has left and updates the active user count. If no users remain in a room, the room is automatically removed from memory to optimize server resources.

Screenshots



Main Page

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 2, 2026



Output Display

Conclusion

The Real-Time Code Collaborator project successfully demonstrates the development of a web-based collaborative programming platform that enables multiple users to write, edit, and execute code simultaneously in a shared environment. By utilizing modern web technologies and real-time communication mechanisms, the system provides instant synchronization of code changes with minimal latency, even when multiple participants are working concurrently. This capability significantly improves team productivity and reduces the delays commonly experienced in traditional asynchronous development workflows.

The backend of the system, developed using Node.js and Express.js, efficiently manages room creation, user sessions, request handling, and communication services. The frontend, implemented using React.js, provides a responsive and user-friendly interface consisting of a shared code editor, collaboration controls, and an output console. Real-time synchronization is achieved through Socket.IO, which ensures that any modification made by one participant is immediately reflected on the screens of all connected users. Furthermore, integration of the JDoodle API allows secure execution of code in multiple programming languages without requiring local compiler installation.

Although the system performs effectively, certain limitations still exist. Dependence on third-party code execution services may affect availability or response time, and performance may be impacted under heavy user load or unstable network conditions. These challenges present opportunities for future improvements and optimization.

Future Scope

The future scope of the Real-Time Code Collaborator project is extensive, and the system can be further enhanced into a complete professional collaborative coding platform. One major direction for future development is the integration of artificial intelligence features that can provide real-time code suggestions, detect syntax errors, recommend improvements, and assist users during programming tasks. Such intelligent features would increase productivity and make the platform more useful for beginners as well as experienced developers.

The security of the system can also be strengthened by implementing advanced authentication methods such as OAuth login, multi-factor authentication, and role-based access control. These features would help ensure that only authorized users can access collaboration rooms and perform editing operations. In addition, encrypted communication and secure data storage can further improve system reliability.

Another important enhancement is the development of an internal sandbox-based code execution engine instead of relying completely on external APIs. This would provide faster execution, greater privacy, improved scalability, and better control over supported programming languages. The system can also be extended to support additional languages, frameworks, and development environments.

10. References

[1] E. Adar, J. Fogarty, D. S. Weld, B. Lafon, and J. Wobbrock, “CodeHelpers: Unobtrusive Assistance for Novice Programmers Doing Creative End-User Programming,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2004, pp. 575–582.

[2] M. Ancona, A. Fantechi, S. Gnesi, and G. Ristori, “Deadlock Detection in Distributed Systems through Abstractions,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 32, no. 1, pp. 1–42, 2010.

[3] C. Bessiere, E. Freuder, and J. C. Régin, “AC3: Arc Consistency Algorithms,” *Artificial Intelligence*, vol. 32, no. 3, pp. 263–297, 1987.

[4] K. Boersma and H. Holone, “Performance in Software Development: A Systematic Literature Review,” *Information and Software Technology*, vol. 48, no. 6, pp. 495–506, 2006.

[5] J. Brandt, P. Guo, J. Lewenstein, M. Dontcheva, and S. Klemmer, “Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010, pp. 1589–1598.

[6] S. Chong and J. Reppy, “A Practical Concurrent Garbage Collector for a Multithreaded Implementation of ML,” in *ACM SIGPLAN Notices*, vol. 28, no. 6, pp. 140–150, 1993.

[7] J. Daubert and G. Donaldson, *The Design and Implementation of the 4.4BSD Operating System*. Reading, MA, USA: Addison-Wesley, 1996.

[8] S. G. Elbaum, H. N. Chin, M. B. Dwyer, and J. Dokulil, “JCrasher: An Automatic Robustness Tester for Java,” *Software Testing, Verification and Reliability*, vol. 14, no. 2, pp. 105–135, 2004.

[9] M. Fischer, B. Gärtner, N. Lynch, and A. Russell, “BFT Protocols under Fire: On Solvability and Security,” in *European Symposium on Research in Computer Security*, Berlin, Heidelberg: Springer, 2016, pp. 311–329.

[10] T. Berners-Lee, R. Fielding, and H. Frystyk, “Hypertext Transfer Protocol – HTTP/1.1,” Internet Engineering Task Force (IETF), RFC 2616, 1999.