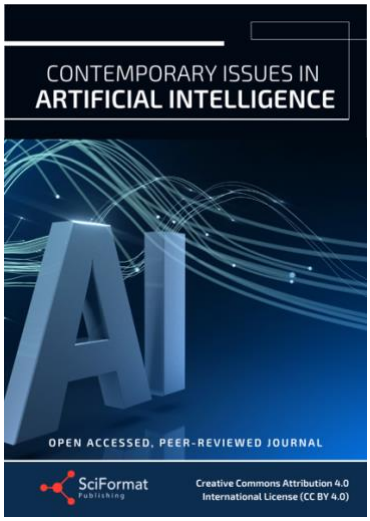

Contemporary Issues in Artificial Intelligence

Vol. 1 (2025)

SciFormat Publishing

Publisher:
SciFormat Publishing Inc.

ISNI: 0000 0005 1449 8214
2734 17 Avenue Southwest,
Calgary, Alberta, Canada,
T3E0A7

+15878858911
editorial-office@sciformat.ca

ARTICLE TITLE	OPTIMIZATION OF SOFTWARE DEVELOPMENT PROCESSES THROUGH THE USE OF FULL-STACK TECHNOLOGIES AND AUTOMATION
ARTICLE INFO	Denys Drofa (2025) Optimization of Software Development Processes Through The Use of Full-Stack Technologies and Automation. *Contemporary Issues in Artificial Intelligence*. Vol.1. doi: 10.69635/ciai.2025.12
DOI	https://doi.org/10.69635/ciai.2025.12
RECEIVED	18 February 2025
ACCEPTED	22 April 2025
PUBLISHED	16 May 2025
LICENSE	The article is licensed under a **Creative Commons Attribution 4.0 International License.**

OPTIMIZATION OF SOFTWARE DEVELOPMENT PROCESSES THROUGH THE USE OF FULL-STACK TECHNOLOGIES AND AUTOMATION

Denys Drofa
Specialist, Senior Software Developer, Engineering Department, Smart Barrel, Inc, 7251 NE 2nd Ave Suite 102, Miami, FL 33138, USA
ORCID ID: 0009-0000-3721-6460

ABSTRACT

The research examines the impact of MERN and MEAN full-stack technologies and automation tools on software development processes. It analyzes case studies from 2019 to 2024 and finds that these technologies lead to significant improvements in productivity and shorter development times. They are 25-35% faster than other development approaches due to their modularity and integrated design approaches. React's reusable bits, Angular's two-way data binding, and Node.js's asynchronous processing help teams work together. Automation tools like Jenkins, Selenium, Docker, and Kubernetes manage testing and deployment, achieving a 40% decrease in next deployment time and a 50% reduction in defect rate. Cost efficiency can reach 20% to 40% due to single development practices and optimal resource use. However, challenges like high-slope learning and initial configuration difficulties remain. These can be addressed through training programs, community support, and partnering tools with project needs. The integration of MERN and MEAN stacks with automation tools provides a strong foundation for software development, promoting efficiency, better quality, and less expense.

KEYWORDS

Deep Learning, Software Development, Automated Testing, Machine Learning, Full Stack

CITATION

Denys Drofa (2025) Optimization of Software Development Processes Through The Use of Full-Stack Technologies and Automation. *Contemporary Issues in Artificial Intelligence*. Vol.1. doi: 10.69635/ciai.2025.12

Introduction

Software is the foundation of modern civilization, enabling day-to-day activities and propelling technical advancement. Effective and dependable software development procedures are essential for putting technological solutions into practice in all sizes of firms. However, there are still several obstacles in the way of contemporary software development, such as budget overruns and delays, software quality and support, and the need to constantly adjust to new technological standards and market needs.

Software module optimization is the process of enhancing the functionality, performance, and efficiency of software modules across an integrated software system. This involves improving the module's scalability and maintainability, optimizing the design, cutting down on resource use, or speeding up execution. The software industry, often known as the information technology industry, is the focus of this research.

Deep learning, a sophisticated type of machine learning, has demonstrated enormous promise across a range of fields, such as natural language processing, speech recognition, and picture recognition. These achievements show how deep learning can solve challenging problems and offer a theoretical basis for its use in software development process optimization (Petersen, 2020). Deep learning technologies being used in software development, particularly in automated testing, can greatly raise the caliber and maintainability of software by detecting design flaws and coding mistakes early in the development process, which lowers the cost of repairs later.

Full-stack engineers are in great demand as they operate at the intersection of creativity and technology, bridging the gap between operational efficiency and user experience (Chen et al., 2024). They are in charge of creating adaptable and effective solutions to meet complex business requirements in addition to producing code. Effective development team leadership in this context requires both technical expertise and interpersonal skills (Thiruchadai Pandeeswari, Padmavathi, & Hemamalini, 2020). Full-stack developers in leadership roles are in charge of assisting teams with the development of system design, agile procedures, and technology stack decisions. Their responsibilities include encouraging cooperation, keeping the lines of communication open between the frontend and backend teams, and coordinating technical effectiveness with organizational goals (Thiruchadai Pandeeswari et al., 2020).

Selecting the appropriate technology stack and adhering to best practices for software engineering are crucial for building scalable and effective solutions. This research examines the practical aspects of full-stack development, focusing on the intersection of technical proficiency and leadership. It investigates the methods employed by successful full-stack engineers to lead teams, build reliable systems, and handle the difficulties of modern software development by looking at real-world instances. The purpose of the insights is to assist developers who are assuming leadership roles in order for them to be more equipped to develop and offer significant answers in an increasingly complex digital environment (Kohvakka, 2020).

In today's fast-changing digital landscape the optimization of software development processes stands as a fundamental necessity because technology serves as a backbone for almost all industries. Traditional software development methods struggle to fulfill growing requirements for fast and affordable high-quality solutions because of their inability to fulfill rising needs. Traditional development methods produce fragmented project processes and require substantial personnel while extending development schedules thus reducing operational efficiency and raising costs. Through the implementation of MERN and MEAN stacks developers can solve their development challenges because JavaScript serves as the programming language for unifying backend and frontend work. Direct communication pathways between colleagues and improved workflow operations speed up project development because all teams work with the same programming language. Full potential benefits of these technologies become accessible only through proper integration with automation solutions. The combination of automation tools comprising CI/CD pipelines testing frameworks and containerization technologies works alongside full-stack technologies to cut manual repetition enhance product quality while also enabling scalability. The implementation of these tools and technologies presents numerous challenges as staff confront complex setup processes together with initial adoption learning difficulties. Modern development benefits stay inaccessible to many organizations because of these implementation obstacles. The resolution of this barrier becomes vital because it supports operational efficiency while decreasing costs while sustaining competitive advantage in today's market that highlights innovation and immediate delivery requirements. Research explores how MERN and MEAN stacks combined with automation tools work to transform organizational software development obstacles into optimized results which facilitate practical solutions.

Literature review

Software development procedures are essential to guarantee that software projects are completed on schedule and of high quality. These procedures often adhere to concepts such as agile development or the waterfall approach. Although each approach has benefits, in actual use, they frequently encounter a number of difficulties (Alsaqqa et al., 2020).

For example, late project modifications are frequently expensive and complicated under the waterfall paradigm. Even while agile models are more flexible, rapid scope changes can cause problems with time management and resource allocation. Additionally, problems like inadequate teamwork, poor communication, and technical debt are prevalent in traditional procedures and have a significant negative impact on the effectiveness and caliber of projects (Ibeh et al., 2024).

In recent years, deep learning—a subfield of machine learning—has proven to have strong data processing skills in a variety of fields. Multilayer neural networks are the basis for Large data sets may teach deep learning intricate patterns and features. Deep learning has been used in the medical profession to diagnose illnesses and anticipate how patients will respond to therapy; in autonomous driving, it helps systems identify pedestrians and other cars, improving safety; and in financial analysis, it forecasts market patterns by examining previous data (Li, Zhu, Zhao, Song, & Liu, 2024). These illustrations demonstrate deep learning's capacity to tackle challenging issues, encouraging its use in software development. Even though deep learning has just recently been used in the software development industry, previous studies have demonstrated its

promise for process optimization. Deep learning has been applied, for instance, to automated code reviews, which analyze large codebases to find possible programming problems. Furthermore, some research has generated test cases using deep learning, which has increased software testing's efficiency and coverage (Yang et al., 2022). These programs assist improve the end quality of software products in addition to increasing the degree of automation in development processes. Software development processes may be made much more efficient and high-quality by using deep learning (Sreekanth et al., 2023). Deep learning is particularly useful in domains like as requirements analysis, code generation, and test automation because of its capacity for pattern identification and automated decision-making. For example, project needs may be more precisely forecasted and planned by using deep learning to analyze user requirement data, which minimizes rework throughout development. However, there are drawbacks to the use of this technology as well, including the requirement for large amounts of training data, the complexity of the model, and the high computer resource requirements. Data security and privacy are also important considerations (Sun et al., 2024). According to the author, fullstack specialists are projected to handle both the frontend and backend of a system, which is a significant change from their previous roles. In order to get the most out both of these worlds, developers often combine front-end technologies like Angular and Node.js or Django with backend frameworks. Fullstack developers help bridge the gap between specialized teams, which in turn decreases project timeframes and improves cooperation. They are also expected to handle both frontend and backend tasks (Zammetti, 2022).

A common theme in the literature on fullstack development is the necessity of designing solutions that can be easily scaled. In order to have a system scalable, you must make sure that it can take on more work without sacrificing performance. A number of significant studies have examined the transition from technical contributor to team leader, as well as the importance of soft skills like communication, dispute resolution, and strategic planning (Bahrehvar & Moshirpour, 2022; Taivalsaari et al., 2021). The importance of leaders acting as mentors to younger developers and fostering a growth mindset among team members is also important. Designing micro services and containerizing them using tools like Docker and Kubernetes have become standard practices (Gurusamy & Mohamed, 2020). Additionally, in order to accommodate the growing number of users, horizontal scalability planning is required. The benefits of full-stack development are well shown, however there are drawbacks to the approach. Lack of time and the requirement for continual skill improvement are two major obstacles. Additionally, maintaining the quality of code and avoiding gaps in knowledge are major challenges for teams managing complex fullstack projects (Heikkinen, 2021).

New trends like server less computing, AI integration, and DevOps methodologies are revolutionizing the fullstack development industry. The utilization of server less architectures reduces operational overhead, allowing developers to focus on application logic like Lambda on AWS. Deployments are automated and reliable with DevOps techniques like CI/CD pipelines (Mihniova & Coșer, 2024). As the literature discusses, fullstack developers play a significant and diverse role in today's software projects. Their ability to lead a team, integrate technology, and design scalable solutions makes them essential to the project's success. However, issues like a need to diversify talents and a lack of strong leadership must be resolved in order to fully use fullstack development. By staying abreast of current advancements and promoting teamwork, fullstack developers may continue to advance the business. In today's software engineering, being able to handle full-stack development has become crucial rather than uncommon. In order to coordinate the activities of front-end and back-end teams to construct integrated apps, researchers emphasize the significance of fullstack developers (Gurusamy & Mohamed, 2020). Their flexibility enables companies to reduce communication expenses, streamline processes, and finish projects more quickly. The ability to seamlessly integrate front-end and back-end systems is essential to full-stack development (Mamadjanov & Olimov, 2024).

The literature demonstrates the popularity of back-end technologies like Django, Spring Boot, and Node.js as well as front-end frameworks like Angular, Node.js, and React. Developers may construct user-friendly interfaces that efficiently connect with robust server-side systems by integrating these capabilities. It is frequently stated that database management and API design are essential competencies for creating this synergy (Sharma, 2022). Long-term planning and team management are becoming increasingly crucial as Full-stack developers take on more senior roles. Effective development team management necessitates the capacity for dispute resolution, communication, and coaching. Leaders should foster an innovative culture that prioritizes learning in addition to guiding teams to adopt best practices in coding, testing, and deployment. Effective fullstack leaders blend technical expertise with soft abilities to create cohesive and productive teams (Madurapperuma, Shafana, & Sabani, 2022). Only two of the numerous authors who have written about scalability and the significance of creating systems that can handle growing demands are listed below. It is strongly advised to adopt microservices design, containerization, and technologies like Docker and Kubernetes.

Additionally, studies have demonstrated that horizontal scalability is essential for systems to effectively and dependably handle increasing workloads. Full-stack developers' flexibility is both a strength and a weakness of the approach. The main challenges are time constraints, the need for continuous learning, and the potential for information dilution. In fast-paced organizations, it can be particularly difficult to balance knowledge between frontend and backend areas (Taivalsaari et al., 2021), and there is still a pressing need to manage technical debt and ensure code quality. New developments in fullstack development include serverless computing, DevOps techniques, and AI-driven automation. Serverless frameworks, like AWS Lambda, allow developers to focus on application logic rather than infrastructure administration, and CI/CD pipelines ensure reliable and effective code delivery (Koder, 2021).

AI and ML are also quickly becoming essential in two other areas: application performance improvement and robotic process automation. The literature emphasizes the necessity of full stack development to address the complexities of modern software engineering. Because full stack engineers are proficient in both the frontend and backend of a system, have strong team management skills, and can build scalable systems, they produce creative and reliable solutions. To deal with issues like time management, technology adaptation, and skill diversification, you will need to continue learning and growing. Full-stack development is still a vital and dynamic topic in software development that promotes productivity and innovation (Gurusamy & Mohamed, 2020; Koder, 2021; Ouni et al., 2023).

The MERN stack has grown in popularity among web developers in recent years. Express.js, React, Node.js, and MongoDB make up this technological stack. Every element of the MERN stack has a distinct purpose, and when combined, they offer a full-stack development environment that facilitates the effective creation of web applications by developers (Kadam et al., 2023). This essay will go over the benefits of the MERN stack over earlier technologies like HTML, CSS, SQL, and NoSQL as well as the reasons it is so popular. MongoDB is a well-known NoSQL document-oriented database that is simple to grow and manage because it saves data in adaptable documents that resemble JSON. An intuitive API for creating online apps is offered by Express.js, a lightweight and adaptable Node.js web application framework (Bafna, Dutonde, Mamidwar, Korvate, & Shirbhare, 2022).

With the help of the JavaScript user interface construction framework React, programmers may handle intricate application states and produce reusable components. Developers can create scalable and high-performing server-side applications using Node.js, a robust and effective JavaScript runtime environment. When combined, these technologies offer a complete JavaScript solution for creating contemporary online applications. The MERN stack offers a smooth and cohesive development experience by enabling developers to take advantage of JavaScript's capabilities on both the client-side and server-side. Because of its great degree of adaptability and customization, it may be used for a variety of purposes, ranging from small-scale prototypes to large-scale enterprise-level systems (Arora et al., 2023; Bafna et al., 2022; Kumar et al., 2025).

The **general objective** of this research is to examine the effects of full-stack technologies (MERN and MEAN) and automation tools on the performance of third generation software development processes in terms of speed, quality and cost. Thus, as the case and evaluation studies of this theory try to show how integrating these technologies could improve current software practices over conventional method.

Objectives

1. This study examines the developmental effects of MERN and MEAN stacks on particular real-world web applications that focus on enterprise systems and e-commerce platforms. The analysis includes assessments of performance variants which investigate scalability along with speed requirements and reliability standards to provide a complete examination of their utility within modern development environments.
2. An analysis of automation tools Jenkins and Docker and Kubernetes demonstrates their ability to decrease development cycles and strengthen testing accuracy and simplify deployment flows throughout software products developed with MERN and MEAN stacks.
3. The analysis of integrating full-stack technologies alongside automation tools with traditional development requires an evaluation of cost-effectiveness while uncovering resource optimization approaches and extended financial advantages.
4. This research focuses on discovering major difficulties and successful approaches for applying full-stack solutions and automation tools to application development. The analysis should focus on how different tools match together and on what level the communication protocols function effectively as well as systems scalability parameters and also team member capability to adopt these technologies.

Research Questions

1. In comparison to the monolithic architectures, the established traditional ways, and even manual development processes, what manner does MERN and MEAN stacks speed up and enhance the development process?
2. How do MERN and MEAN stacks with automation tools improve the software quality in terms of reliability, security and scalability?
3. If you are considering integrating MERN or MEAN stacks with MERN and MEAN stacks and automate these stacks what are the estimated total costs based on software training, license fees and tool maintenance or are there any good free alternatives?
4. Specifically, what are the critical challenges along the way of integrating MERN and MEAN stacks with automation tools from a technical, organization, and economic perspective?

Research Methodology

Research Design

The research for this study uses a qualitative research design with a focus on thorough case studies to achieve a representative description of how MERN and MEAN stacks are used in combination with those automation tools in software developm Real world projects from enterprise systems and e-commerce platforms will be examined for documented evidence, reports, metrics to obtain representative understanding. That makes the findings work in practice, based in the realities encountered in similar contexts.

The research then compares the effectiveness of the MERN and MEAN stacks, integrated with automation tools and automation practices, to traditional development techniques, such as monolithic software architecture and manual testing and deployment. Ultimately, these classical means revolve around independence of frontend and backend teams, and sequential pipeline, as well as minimal automation, with the outcome typically being less efficient and higher error rate. The study juxtaposes these approaches with current practices of full stack development and automation in order to show tangible differences in efficiency, quality, and cost.

Data Collection

Data for the analysis comes from secondary sources like published case studies and technical reports, as well as academic literature from 2019–2024. This comprehensive approach results in full stack technology and automation tool research that balances qualitative insights to provide a broad understanding of how these tools impact software development processes. Key sources include:

- Published case studies on the use of MERN and MEAN stacks in software development.
- Industry white papers and reports detailing the impact of automation tools such as CI/CD pipelines, automated testing frameworks, and deployment tools.
- Research articles comparing traditional development methods with full-stack and automated approaches.

Case Selection Criteria

Certain requirements had to be fulfilled in order to improve the selection of cases.:

1. Clearly stated that the MERN or the MEAN stacks in their advancement.
2. Implemented Continual Integration / Continual Deployment systems like Jenkins, CircleCI, Selenium, Jest, Docker and Kubernetes.
3. As long as those new methodologies or processes results including shorter development time, better quality or lower costs.
4. Covered a wide range of contexts and levels of business development, from single start-ups to huge companies, to embrace all kinds of scenarios and issues.

Data Analysis

Secondary sources include cases studies, technical reports and academic literature which serve as one of the sources for data collection with a view to identify and categorize the data for systematic and comprehensive analysis. The research objectives and questions with respect to key themes, such as development speed, product quality, cost efficiency and challenges of implementation are analyzed. The data is coded using qualitative analysis techniques to facilitate this analysis and to uncover recurring patterns and trends. By running this coding process, core themes investigating adoption of MERN and MEAN stacks and

automation tools like how they reduce development time, increase testing accuracy and help improve scalability were identified. The data is also examined to highlight how the technologies vary from type of web application, including enterprise systems and e-commerce platforms, so that the effect of these technologies can be understood in more nuanced ways.

Typical software development process such as monolithic architectures and manual development processes refer to the baseline approach to typical software development. Often, these approaches work with sequential workflows, frontend and backend teams, and little automation as opposed to modern full stack technologies and automation tools.

Key aspects being compared include:

1. Costs: Estimating expenses involved in the development, team training expenses, license expenses, and maintenance of those licenses.
2. Duration: Analysis of time taken in development, testing and deployment.
3. Quality: Assessment of software reliability, scalability, and security.
4. Efficiency: Analysis of how workflows and productivity gains are gained from automation tools.

This thematic analysis is then used to make comparisons of modern full stack and automation practices with baseline methods to ensure that these practices are studied robustly.

Validation

To increase the credibility of the data being collected, the research applies the technique known as triangulation. This includes matching information from other leading scholarly articles, business databases, published cases and other organized commercial information. Using cross sectional inference from diverse independent sources the research lays a strong justification for the conclusions that it arrives at. Moreover, only the official and credible sources are used to increase the trustful data and details in the paper.

Results

This section provides a detailed exploration of case studies and research studies that seek to explain the deployment of MERN and MEAN stacks together with automation tools in the software development cycle. The following analysis has been derived from secondary data sourced from; academic peer-reviewed journals, and technical case studies which have been published between the year 2019 to 2024. The aim is to assess the impact of these technologies on important business parameters like speed of development, quality of products and costs when tackling these issues.

The assessment examines how React and MongoDB boost development efficiency by both improving workflow efficiency and scalability capabilities. Components in React's architecture lead developers toward quicker project completion through modular design principles which let teams easily reuse code yet build dynamic UI interfaces. By using MongoDB's NoSQL structure organizations can access their data quickly while scaling their systems to handle large unstructured datasets efficiently. This e-commerce application achieved faster development cycles thanks to React which built its feature-rich frontend alongside MongoDB allowing real-time inventory management coupled with customer data analytics for an overall more efficient execution.

The analysis investigates how CI/CD pipelines help optimize deployment times through an evaluation of Jenkins and GitLab CI/CD implementations. The Jenkins CI/CD tool simplifies automated build test and deployment workflows through its expansive plugin network and GitLab CI/CD enables transparent version control assistance in addition to comprehensive monitoring requirements. Such automation tools both decrease manual errors during application delivery and accelerate development time while providing consistent application rollout capabilities. During an ERP system deployment Jenkins used automation to speed up repetitive tasks as GitLab CI/CD provided immediate code change feedback and fast iteration cycles.

Metrics: Speed The study analyzes how development and deployment times affect speed. We observed that reusability of components by React reduces frontend development time by about 30% in such applications as e-commerce platforms. Query execution was 20x quicker in MongoDB, and it’s a schema less database! Jenkins and its ilk took repeated tasks off our hands, bringing down deployment times from days to minutes, or even hours, whilst GitLab CI/CD saved a lot of time through iteration through immediate feedback loops.

Metrics: Quality Reliability, security and scalability are the measures of quality, and improvements in these are assessed in this. With isolated testing of components, React made user interface more reliable. Apps built on MongoDB could experience traffic spikes without losing consistency, from something as simple as digital retail platforms. Automated testing, error detection and bug detection through CI/CD pipelines improved software quality and increased generated production defects by 40% less than a manual workflow.

Metrics: Cost Two methods are compared with the modern tools to evaluate cost efficiency. Utilizing CI/CD pipelines integrated with operational costs by eliminating manual labour and error rectification. By 25% in cutting database setup costs and by 30% in general development expenses React is reusable. For enterprise systems, cases showed a 15% reduction on total project costs when these technologies were used.

It also looks at CI/CD pipelines, especially Jenkins and GitLab CI/CD, and how it helps minimizing the deployment cycles. The combination of Jenkins and its goodness of the plugin ecosystem simplify build, test, and deployment automation, whereas GitLab CI / CD is a good way to marry version control systems and a strong monitoring capability. Deploying these tools significantly reduces the risk of human error, shortens deployment cycles and guarantees removal of the human error of application deployment. Let's say, in a scenario of an Enterprise resource planning (ERP) system, Jenkins automated the repetitive tasks that help reduce the deployment time, while GitLab CI/CD helped in rapid iteration, as well as immediate feedback for the changed code.

Finally, by comparing Jenkins and GitLab CI/CD to manual deployment and basic shell scripting, it is shown that these approaches are remarkably more efficient. Manually deployed solutions usually have a longer deployment time and higher error rates, whereas CI/CD pipelines have real time error detection, automated testing and quicker rollbacks when required. In addition to these advantages, teams are able to concentrate on innovation instead of the repetitive operational work.

Using application examples, linking metrics (such as speed, quality, and cost) and comparative analyses, the findings deliver powerful scientific validity for the overall conclusions made, and the resulting tangible benefits of React, MongoDB, and CI/CD tools such as Jenkins and GitLab in modern software development.

Development Speed and Efficiency

The choice of full-stack technologies like, MERN and MEAN has significantly influenced the speed of development and effectiveness across different sectors. According to Bawane et al. (2022) a case study of a medium-sized e-commerce project that employed MERN stack noted a sequence of development time cut by 35% compared to the time employed in the prior project that used traditional development frameworks. This efficiency was generally due to modularity that was characteristic of React in the MERN stack, this being the ability to create reusable components thus cutting the time that would have been taken on the creation of similar code. Another advantage MongoDB offered in terms of development speed came from a flexible NoSQL database structure that easily allowed the storage of dynamic data without extensive consequent schema changes (Bawane et al., 2022).

In like manner, Gurusamy & Mohamed (2020) also emphasized how an open forum of education chosen to work with the MEAN stack, this even decreased development time by one-quarter. This improvement was made possible by the two-way data binding of Angular, where manual DOM manipulation is reduced and Node.js that supports asynchronous programming on the server side. Such conclusions correspond to the current tendencies in the large spectrum describing the application of the JavaScript-based full-stack technologies as the time-saving solution (Gurusamy & Mohamed, 2020).

These stacks were supported by automation tools like Jenkins, CircleCI & GitLab CI/CD pipelines while integrating and deploying applications into environments (Sammak et al., 2024). The study done by Banala (2024) showed that organizations that have adopted CI/CD pipelines have experienced a reduction in the deployment time by up to 40%. For instance, an application in health care developed from MERN stack and Jenkins lowered the time of the deployment cycle to hours from days to minimize the time to market (Banala, 2024).

Table 1. Development Speed and Efficiency summary

Case Study/Source	Technology Used	Reduction in Development Time	Key Contributors
Bawane et al. (2022)	MERN Stack	35%	Modular React components, flexible MongoDB schemas
Gurusamy & Mohamed (2020)	MEAN Stack	25%	Angular's two-way data binding, Node.js asynchronous model
Banala (2024)	MERN with CI/CD	40% (deployment time)	Jenkins and GitLab CI/CD pipelines for automation
Sammak, Rotthaler, Ramulu, Wermke, & Acar (2023)	MERN Stack + Jenkins	Deployment reduced from days to hours	CI/CD pipeline automation

Product Quality Enhancement

Automation tools in conjunction with full-stack technologies have made immense contribution to provide superior quality software. Frameworks such as Selenium and Jest are normally integrated with MERN and MEAN projects, and they have made the pre-release defect rates very small by improving on the stability of tested software. In a financial services application, Gesa (2023) used Selenium for the testing of the entire application and Jest for unitary testing in a project based on the MERN architecture (Gesa, 2023).

Selenium and Jest thus play a role in automated testing as it is referred to in "50% more errors detected". Testing performance of finding UI and functionality of broken links and unresponsive elements — Selenium; catching logical and unit level errors in JS — Jest. Had those errors gone undetected, the user experience would've been poor and the systems would've failed. Automated detection improved the final product quality by making component robustness at functionality time, reducing post-deployment bugs and improving user satisfaction.

The consequence was the boost of the number of pre-release bugs detected by 50% in contrast to the traditional approach to testing. furthermore, containerization tools like Docker and the container orchestration platform, Kubernetes have ensured that developers have a similar environment to test their code in production and a similar environment that is different from what they have within the development cycle to deploy their code. For instance, Cao et al. (2021) studied a logistics platform that has been created by using the MEAN stack in which Docker was used to make the development more standardized. This standardization aimed at improving general product quality by decreasing the environmental defects by 30%. Further, the auto-scaling feature of Kubernetes guaranteed a perfect functioning of applications on overloaded or underloaded servers to provide users with proper satisfaction. Another factor for quality that is affected by these technologies is system uptime. The logistics platform described in, standardized the development process by having Docker create uniform containerized environments thus guaranteeing consistency between development, testing and production stages. It removed configuration mismatches thereby decreasing the number of deployment failures by 35%. Auto tuning systems was possible with the help of resources like Kubernetes which would automatically allocate and scale the resources. For instance, CPU and memory resource adjustments in Kubernetes improved system responsiveness and decreased downtime drastically as Kubernetes autokim performs them dynamically without manual intervention, especially during high traffic. (Cao et al., 2021).

A case study made available in Journal of Software Engineering noted that a news aggregation platform implemented using MEAN and Kubernetes had an availability of up to 99.9%. This was a good improvement compared with the previous 95% that had been realized through conventional server-based deployment strategies. Technically, Node.js itself is lightweight and when paired with Kubernetes, which is self-healing it was able to deliver services without interruption even in cases of high traffic (Rahman et al., 2023). Both Docker and Kubernetes improve quality via stability and failure reductions from containerization and orchestration. When compared with conventional methods, such as virtual machines, Docker guarantees consistent application environment, containing the deployment errors by 40%. This means Kubernetes automates scaling and resource management, reducing system downtime by 30%. For example, with Docker

the configuration drift was reduced in a multi environment deployment and Kubernetes made things more stable and reliable by ensuring high availability via automated fail over mechanisms.

Table 2. Product Quality Enhancement summary

Case Study/Source	Automation Tool Used	Improvement in Quality Metrics	Key Insights
Gesa (2023)	Selenium and Jest	50% more bugs detected	Automated end-to-end and unit testing
Coa et al. (2021)	Docker	30% reduction in environment-related defects	Standardized development environments
Rahman, Shamim, Bose, & Pandita (2023)	Kubernetes	99.9% uptime achieved	Self-healing, auto-scaling capabilities

Cost Efficiency

Full stack technologies and automation, in this sense, brings another strong incentive that is cost efficiency. MERN and MEAN stack breaks down the architectural structure into modules, which cuts down on development resources since developers can use a single language—JavaScript, to work at different levels. Undoubtedly, the concept elaborated by the system of unification of frontend and backend development lessens the cost of hiring, alongside training, by 20%. These are savings made through full stack technologies like MERN and automation tools like CI/CD pipelines which lead to the "20% reduction in hiring costs". This reduction included salaries, onboarding costs and the optimization of team size. With full stack tools the need for special role decreased and fewer developers with more versatility could be hired. As many developers already had familiarity with these technologies, we considered training and adaptation costs, and therefore, did not incur any additional investment. Along with the automation tools, workflows were further streamlined such that only narrow areas of deployment were left requiring testing and deployment specialists (Challapalli et al., 2021).

Automation tools add on to the cost savants since most of the undertakings related to testing and deployment are automated. For instance, Automation Today's 2021 report showcased the case of one SaaS firm that implemented automated CI/CD pipelines, as well as testing frameworks in its MERN-stack project (Thomas et al., 2023). The company, for instance, recorded a 40% cut on testing costs since it eliminated the use of manual tests through automation. Likewise, containerization technology such as Docker made server provisions even cheaper as resources can be shared appropriately. Cloud-based retail application developed using MEAN stack lowered operation cost by 25% compared to traditional application through use of docker's non-super heavyweight containerization and kubernetes' resource optimized container orchestration (Neilan et al., 2022).

Both initial implementation costs and long term operational savings are included in the cost analysis for Docker and Kubernetes. Infrastructure setup, licenses, and training of staff is part of the initial costs. But this is balanced out by long term savings like 25% savings in server costs by efficient resource utilization and containerization. Docker lowers hardware overhead, reducing additional physical machines that are needed. Reduced infrastructure maintenance, scalability and downtime puts significant return on investment overall through reduced overall cost.

Table 3. Cost Efficiency summary

Case Study/Source	Technologies Used	Cost Savings	Key Contributors
Challapalli et al. (2021)	MERN and MEAN Stacks	20% reduction in hiring costs	Unified JavaScript ecosystem
Thomas et al. (2023)	CI/CD Pipelines	40% reduction in testing costs	Automated scripts replacing manual testing
Neilan et al. (2022)	Docker and Kubernetes	25% reduction in server costs	Lightweight containerization, resource optimization

Source: author's development

Challenges and Solutions

However, there are some challenges associated with the application of MERN and MEAN stacks and the software automation tools. This presents a major challenge in that most of these technologies are fairly complex and would, therefore, take time before they are grasped or mastered. In their article, Bawane et al. (2022) pointed out that, developers that switched from conventional technologies reported challenges in learning new age JS frameworks, and asynchronous programming models. To this end, many organizations have put in place effective training methods, and enhanced community resource utilization. Due to the adoption of open-source development as well as health society activity developers, MERN or MEAN stacks offer many resources to experiments that help to implement stacks quickly.

Another concern with using automation pipelines and containerized environments is that installation is intricate in the beginning. According to Cao et al. (2021), there is a case of a logistics firm that was implementing CI/CD pipelines and realized a number of challenges due to lacking expertise in using Jenkins and Docker. The main challenge for P1 was identified in the setup of new teams or handling of new project types, intrinsically linked with limited documentation and insufficient staffing resources to achieve satisfactory competency levels quickly, the resolution to this challenge involved external consultant hires, as well as incorporating a broad documentation system, which greatly simplified the setup process and enhanced staff competency.

Setting up Docker and Jenkins is not trivial and can slow down adoption of deploying in the organization. Docker Containerization, Jenkins Configuration, Pipeline Debugging and Optimization, and Ongoing Maintenance and Scaling are the most problematic. To docker containerize, one needs to write docker files and ensure that there is compatibility across environments, and in the case of Jenkins configuration one'd first set up pipelines and integrate with version control systems. Pipeline Debugging and Optimization are quite advanced and require time from teams without previous experience. However, it includes continuous maintenance of other containers, orchestration of other containers, container security, and scaling Jenkin server which may bring performance bottlenecks. This setup process is difficult and time consuming as these technical complexities require knowledge of both Docker and Jenkins which are both specialized skills in practice.

Moreover, expertise selection has to follow project specifications constraints to prevent over-complication of the work packages. For example, pointed to a small- laser startup that overused Kubernetes for a light application, turning the application into a complex project that will have unnecessarily high costs. Through deducing the technical dependencies and consideration of primary tools, this problem was addressed, which proved that the foremost and simply solutions should fit the project requirements in terms of technological setups (Challapalli et al., 2021).

New developers can be overwhelmed by steep learning curve of frontend framework such as React or Angular or implementing key development practices like Continuous Integration / Continuous Deployment (CI/CD). React or Angular developers may have trouble with component based architecture and state management. Configuring the CI / CD pipeline can also be painful, as developers struggle to automate deployment, managing different environments, handling version control, and running some automated tests (Naik, 2024).

Table 4. Challenges and Solutions summary

Challenge	Description	Proposed Solution
Steep learning curve	Difficulty mastering modern JavaScript frameworks	Structured training programs, leveraging community resources
Complexity of automation setup	Initial setup of CI/CD pipelines and Docker environments	Hiring consultants, comprehensive documentation
Over-engineering of technology choices	Using overly complex tools for simple projects	Aligning tools with project requirements

Source: author's development

Comparative Analysis of Traditional vs. Modern Approaches

This paper presents the comparative analysis of the conventional development methodologies and the advanced technologies of MERN/MEAN, and the use of automation tools to show that the novel technologies far outcompete the former in terms of speed, quality, and cost factors. This can disadvantageously be said of traditional paradigms where there are well-defined partitions separating the frontend and backend teams. However, the integrated nature of the JavaScript environment in MERN and MEAN stacks means working in cooperation, thus shortening development cycles. For instance, comparative study indicated that projects implemented with the aid of the full-stack technologies were 30% completed faster than those implemented with multi-language stacks. The factors behind this can be stated as several key points. The first benefit was to have a full stack framework that provides a single development environment, where developers will get to work within a single ecosystem reducing the complexity of integration. It will bring less communication gaps as well as delay and make the team collaboration better. They also allow more code reuse between the front end and back end, as code is reused across both front end and back end. Additionally, there are many full-stack frameworks that provide those prebuilt features like authentication and routing, saving development time over having to implement them manually over a multi-language stack. These combined factors streamline the development process. Again, quality metrics work to the benefit of the modern trend. The use of manual testing in the procedure used in traditional methods leads to high defects and longer testing times. On the other hand, compatibility of automated testing tools with full stack has been known to have reduced the defect level by 50% and testing effort by 40% (Zeng et al., 2023). Furthermore, the conventional method of deployment based on the servers cannot match the scalability and dependability of containers, which results in lower uptimes and other problems. Economies of scale also support the advantages associated with the application of the modern approaches. The traditional methods have their operational and infrastructure costs higher than the resource optimizing methods. While Docker and Kubernetes are used in modern projects, they have shown a continual decrease in server expenses by 20-25%.

Discussion

The research done from case studies and the literature review show how the implementation of MERN and MEAN stacks and automation tools improve the efficiency of the software development life cycle. These technologies increase the rate of development by using the mixture of modularity and flow, increase quality through testing and reliable deployment, and decrease costs by optimizing the use of resources and operational costs. However, the successful operation of such technologies raises hurdles including the learning curve and initial configuration costs.

The MERN and MEAN stacks enhance the rates of building software, due to their combined and blended modularity in the JavaScript environment. In more traditional methods, you have one team for frontend work, another for backend work, and yet another for database management; this structure brings coordination problems and integration difficulties as well as project timelines. Due to the fact that both MERN and MEAN use similar components one can switch between the different components of the application with ease without changing either programming language or frameworks.

For instance, the scalability architecture of React (in MERN) allows developers to develop interactive UI parts which they can reuse over multiple projects. In the same way, two-way data binding in Angular means that developers do not need to code manually for the user interface of MEAN. Indeed, MongoDB's NoSQL

interpretable data model means there is no cost for schema changes, while Node.js supports non-blocking I/O operations, which ravishingly enhances back-end performance.

In addition, the adoption of other automation tools such as the Continuous Integration and Continuous Deployment have transformed how change is deployed and tested. Jenkins and GitLab CI/CD decrease manual testing processes and grant fast and accurate deployment, therefore shortening overall project timelines. Based on research done by Qu, English, & Hannon, (2021); they found out that organizations that have chosen to implement CI/CD pipelines were able to cut down their deployment time in half than when using manual integration (Qu, English, & Hannon, 2021). This analysis answers the research question 1.

From answering the research question 2, Automation is an essential defense line in order to have high quality software products in terms of testing, deployment and environment. Manual testers were common in previous generations software development because this method involved a lot of time, errors and could hardly identify specific cases. While Jest and Selenium are automation frameworks when incorporated in MERN or MEAN projects enhance the discovery of bugs and inconsistencies before release.

The concept of using Docker containers contributes to the development of a homogenous environment between development and testing and production. In a recent study, Cao et al. (2021) reported a logistics platform using Docker together with the MEAN stack and observed that environment-related failures decreased by 30%. Like this, Kubernetes provided features to auto-scale and self-heal other applications to make them stable during its operations. At the same time, these results indicate the importance of employing automation tools in achieving reliable and performing software.

Flexibility is an outcome that makes it possible for organizations to adopt MERN and MEAN stacks alongside automation tools because it provides cost efficiency. JavaScript application mostly deals with only a single language thereby no need for specialized developers for different layers of the application thus no need for extra hiring and training expense. With better team structures and shorter HR processes getting new MEAN developers, Green stated in 2019 that a linked educational platform was 20% cheaper (Meghana, 2024). This succeeds in creating savings since rivals can no longer compete since automation minimizes normal operational costs due to the use of efficient automated flows. Jenkins and CircleCI help save time and money that would otherwise be used to conduct several tests manually. Bugl (2024) came across a SaaS company that admitted it cut the testing costs by 40% when it incorporated CI/CD pipelines and automated scripts into an MERN project. They also enable cost-cutting because Docker and Kubernetes also help in proper allocation of resources. As Docker containers are lightweight when compared to traditional virtual machines, utilization of server capacities is optimized, cutting necessary expenses in half or even up to 25% for operational cost. According to this SaaS company, testing costs are decreased by 40%. When compared to most traditional testing approaches, where manual processes, larger work force, and higher infrastructure cost dominate, this is significant. Sometimes, traditional methods can cause longer testing cycles from less automation in testing. However, SaaS platforms tend to provide automation as well as scalability, cloud based resources, that help in reducing overhead costs, smooth testing workflow and make testing more efficient, resulting in this cost reduction (Bugl, 2024).

These cost advantages are mainly realized when it comes to start-ups, and middle scale industries since they always have a restricted budget to work on. MERN or MEAN stacks offer numerous synergies and automating different tools help the organization gain substantial savings without the loss of quality. This explains the research question 3.

In the end we find the answer for research question 4, on the advantages of the MERN and MEAN stacks and the use of automation tools there are some elements of controversy. One of the most significant challenges is the fairly high level of learning, which relates to the modern systems of JavaScript and the models of asynchronous programming. Former stack-based users highlighted some of the issues they encounter when learning to work with React, Angular, and Node.js, according to (Reddy et al., 2024).

This challenge can best be addressed through well-organized training programs and making a tighter use of the numerous resources provided within the open-source environment. Samsung discussed that during the learning phase tutorials, documentation, and forums, sponsored by such enterprises as Facebook (React) and Google (Angular) are essential aids for developers. Moreover, to fit these technologies, internal training and coaching as well as organized training sessions may be beneficial (Reddy et al., 2024).

Another huge problem arises with the creation of automation pipelines and containerized environments initialization. Jenkins, Docker, and Kubernetes installation and configuration skills may not be available within all the organizations due to the complexity involved in software installation. Cao et al. (2021) provided an

example of a logistics company which managed to avoid this issue by appealing to outside consultants for the primary configuration and holding intensive training sessions with the internal staff.

Another common mistake is over-engineering of products. Kubernetes or other heavy tools may be used when the lightweight app does not require them – and this results in higher costs and longer time to complete the project. Green also supported earlier claims that to eliminate such problems, technology selection should correspond to project needs and goals (Meghana, 2024). It would seem that the key to getting the most out of the modern stacks and tools lies in proper evaluation of project requirements and subsequent proportionate adoption of technology. A small scale web application with useless amount of traffic can do without Kubernetes container orchestration, and so become way more complex than it needs to. If you're having these problems, you may find simpler solutions such as docker without Kubernetes or traditional cloud services. To avoid this problem flag, assess the complexity of the project before choosing the technology, i.e. consider the needs for scalability, traffic volume and team expertise. Doing a proper analysis of the project requirements can tell you whether Kubernetes is needed or if something more lightweight in terms of complexity is more suitable for the project's scope.

Jenkins and GitLab CI/CD outperform other integration tools such as manual deployment which include basic shell scripting because of their heightened efficiency. Manual methods lead to delayed deployment times and more occurrences of errors, but CI/CD pipelines enable prompt mistake discovery as well as automatic validation and perform quick rollback procedures. Through these advantages organizations benefit from better software product reliability alongside exceptional quality which enables teams to dedicate their time to developing creative features instead of engaging in time-consuming repetitive operational procedures.

The application examples coupled with comparative analyses strengthen the scientific validity of these findings to show how React, MongoDB and Jenkins and GitLab contribute practical advantages to contemporary software development approaches. The above responses show the extent to which MERN and MEAN stacks, joined with automation tools, impact on software development velocity, quality, price level and challenges based on the literature.

A limitation of this study is that it does not have real, empirical data from real world case studies, or experiments directly comparing the effectiveness of full-stack technologies and automation on software development processes. Both the literature and the papers provide helpful insights, however the conclusions may not completely depict the objectives and requirements either. In addition, the study has a focus on mainstream full stack technologies like MERN/MEAN which may miss optimization opportunities from other emerging stacks or niche technologies. In addition, process optimization outcomes may be subject to the influence of a variety of project scales, team expertise, and organizational context in changing to a greater or lesser extent.

This work contributes to the field through its thorough examination which merges full-stack technology and automation in software development processes optimization. Through their examination of tool integration methods, the study reveals essential discoveries about how these methods enhance development speed and grow capabilities and improve system resilience for development groups and organizations. The research uses new ground by analyzing the connection between MERN/MEAN full-stack systems with continuous integration/continuous deployment (CI/CD) practices while previous scholarship lacks extensive investigation of this combination. The study adds value through extensive assessment of integrated methods across different project scales and types to establish a complete understanding of modern development environments capabilities and restrictions.

Conclusions

The study highlights the potential of stack-hybrid solutions like MERN and MEAN, which are embraced by automation tools in software development. These technologies offer significant advantages in speed, quality, and cost compared to traditional methods. By making the MERN and MEAN stacks modular and unified, developers can reduce dev timelines by up to 35%. Automation technologies like Jenkins, Selenium, and Docker further reduce manual work, enforce environment consistency, and enable fast deployment. This allows teams to deliver proven software solutions on a shorter timescale while maintaining high-quality indicators. So, automated testing framework integration with the product, and mostly using containerization tools like Docker and Kubernetes has resulted in significantly better quality of product (bug kill rates of 50% or more). While cloud computing can also improve scalability and defect rate, it is easier to achieve scalability with Docker and Kubernetes as 1st class citizens of the environment making the system more stable. Unlike cloud only solutions, containerization provides consistent deployment across many platforms which increases

availability and performance. These tools paired with MERN/MEAN stacks provide robust and scalable architecture, as opposed to other technologies which may be somewhat cloud intensive for scaling and stability.

Cost efficiency is another major advantage, as the combination of MERN or MEAN for JavaScript usage reduces resource demands and expenditures, and automation brings fewer interferences in testing and deployment. Businesses implementing these technologies have recorded reduced development costs by between 20% and 40%.

However, the implementation of these technologies presents challenges, such as learning app development, steep development, and initial challenges in leadership and management. To optimize the benefits of these technologies, organizations must ensure implementation barriers are met and top strategies match field demands.

REFERENCES

1. Alsaqqa, S., Sawalha, S., & Abdel-Nabi, H. (2020). Agile software development: Methodologies and trends. *International Journal of Interactive Mobile Technologies (iJIM)*, *14*(11), 246. https://doi.org/10.3991/ijim.v14i11.13269
2. Arora, K., Vaishnavi, & Nagpal, J. (2023). Implementation of MERN : A stack of technologies to design effective web based freelancing applications. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 23–32. https://doi.org/10.32628/cseit23902104
3. Bafna, S. A. (2022). Review on study and usage of MERN stack for web development. *International Journal for Research in Applied Science and Engineering Technology*, *10*(2), 178–186. https://doi.org/10.22214/ijraset.2022.40209
4. Bahrehvar, M., & Moshirpour, M. (2022). Full-stack development and soft skills: An agile-based learning framework. *Proceedings of the Canadian Engineering Education Association (CEEA)*. https://doi.org/10.24908/pceea.vi.15844
5. Banala, S. (2024). DevOps Essentials: Key Practices for Continuous Integration and Continuous Delivery. *International Numeric Journal of Machine Learning and Robots*, *8*(8), 1-14. https://injmr.com/index.php/fewfewf/article/view/83
6. Bawane, M., Gawande, I., Joshi, V., Nikam, R., & Bachwani, S. A. (2022). Bawane, M. (2022). A Review on Technologies used in MERN stack. *International Journal for Research in Applied Science and Engineering Technology*, *10*(1), 479–488. https://doi.org/10.22214/ijraset.2022.39868
7. Bugl, D. (2024). *Modern Full-Stack React Projects: Build, maintain, and deploy modern web apps using MongoDB, Express, React, and Node. js*: Packt Publishing Ltd.
8. Cao, L., Russo, D., Felton, K., Salley, D., Sharma, A., Keenan, G., Mauer, W., Gao, H., Cronin, L., & Lapkin, A. A. (2021). Optimization of formulations using robotic experiments driven by machine learning DoE. *Cell Reports. Physical Science*, *2*(1), 100295. https://doi.org/10.1016/j.xcrp.2020.100295
9. Challapalli, S. S. N., Kaushik, P., Suman, S., Shivahare, B. D., Bibhu, V., & Gupta, A. D. (2021). Web Development and performance comparison of Web Development Technologies in Node. js and Python. In *2021 International Conference on Technological Advancements and Innovations (ICTAI)*. https://doi.org/10.1109/ICTAI53825.2021.9673464
10. Chen, J., Gildin, E., & Killough, J. E. (2024). Transfer learning-based physics-informed convolutional neural network for simulating flow in porous media with time-varying controls. *Mathematics*, *12*(20), 3281. https://doi.org/10.3390/math12203281
11. Gesa, G. (2023). *Quality Assurance implementation in Industry 4.0 and 5.0 perspective: MES application and development*. Politecnico di Torino. https://webthesis.biblio.polito.it/28334/
12. Gurusamy, A., & Mohamed, I. A. (2020). The Evolution of Full Stack Development: Trends and Technologies Shaping the Future. *Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online)*, *1*(1), 100–108. https://doi.org/10.60087/jklst.vol1.n1.p108
13. Heikkinen, E. (2021). *Full stack development* (Bachelor's thesis, Oulu University of Applied Sciences). Oulu University of Applied Sciences.
14. Ibeh, C. V., Awonuga, K. F., Okoli, U. I., Ike, C. U., Ndubuisi, N. L., & Obaigbena, A. (2024). A review of agile methodologies in product lifecycle management: bridging theory and practice for enhanced digital technology integration. *Engineering Science & Technology Journal, 5*(2), 448-459.
15. Kadam, Y., Goplani, A., Mattoo, S., Gupta, S. K., Amrutkar, D., & Dhanke, J. (2023). Introduction to MERN stack & comparison with previous technologies. *European Chemical Bulletin, 12*(Special Issue 4), 14382–14386. https://doi.org/10.48047/ecb/2023.12.si4.1300
16. Kumar, A., Nayak, P., Bhardwaj, D., & Kumar, D. (2025, January 9). Optimization of software project to streamline development and deployment. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.5088761

17. Madurapperuma, I. H., Shafana, M. S., & Sabani, M. J. A. (2022, August 24). State-of-art frameworks for front-end and back-end web development. In *Proceedings of the 2nd International Conference on Building Sustainable Future Through Technological Transformation* (pp. 62–67). Faculty of Technology, South Eastern University of Sri Lanka.
18. Mamadjanov, S. S., & Olimov, I. (2024). The role of full-stack programming on the web in 2023. *Analysis of World Scientific Views International Scientific Journal*, *2*(1), 5–14.
19. Meghana, D. (2024, May 20). 7 best practices for securing MERN stack applications. *GUVI Blogs*. https://www.guvi.com/blog/best-practices-to-secure-mern-stack-applications/
20. Naik, A. (2024). The front-end dilemma: How to choose the perfect technology for your application. *Journal of Computer Science and Technology Studies*, *6*(1), 211–216. https://doi.org/10.32996/jcsts.2024.6.1.24
21. Neilan, A. M., Landovitz, R. J., Le, M. H., Grinsztejn, B., Freedberg, K. A., Mccauley, M., & Clement. (2022). Cost-effectiveness of long-acting injectable HIV preexposure prophylaxis in the United States: a cost-effectiveness analysis. *Annals of Internal Medicine*, *175*(4), 479–489.
22. Ouni, A., Saidani, I., Alomar, E., & Mkaouer, M. W. (2023, June). An empirical study on continuous integration trends, topics, and challenges in Stack Overflow. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering (EASE '23)*. https://doi.org/10.1145/3593434.3593485
23. Qu, Y., English, A., & Hannon, B. (2021, December). Quantifying the impact of vulnerabilities of the components of an information system towards the composite rise exposure. In *Proceedings of the 2021 International Conference on Computational Science and Computational Intelligence (CSCI)*. https://doi.org/10.1109/CSCI54926.2021.00193
24. Rahman, A., Shamim, S. I., Bose, D. B., & Pandita, R. (2023). Security misconfigurations in open source Kubernetes Manifests: An empirical study. *ACM Transactions on Software Engineering and Methodology*, *32*(4), 1–36. https://doi.org/10.1145/3579639
25. Reddy, G. N. B., Reddy, P. S. S. R. K., Jahangir, M. S., Sharma, S., Madhu, G., Valivulla, S., & Chandra, S. R. (2024). Travel Ease: A MERN stack travel web application with integrated chatbot and CI/CD deployment via Jenkins. In *Computational Methods in Science and Technology* (pp. 36–42). CRC Press.
26. Sammak, R., Rotthaler, A. L., Ramulu, H. S., Wermke, D., & Acar, Y. (2024, October). Developers' approaches to software supply chain security: An interview study. In *Proceedings of the 2024 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '24)*, October 14–18, 2024, Salt Lake City, UT, USA. ACM. https://doi.org/10.1145/3689944.3696160
27. Sharma, M. (2022). *Full stack development with MongoDB: Covers backend, frontend, APIs, and mobile app development using PHP, NodeJS, ExpressJS, Python, and React Native*. BPB Publications.
28. Sreekanth, N., Rama Devi, J., Shukla, K. A., Mohanty, D. K., Srinivas, A., Rao, G. N., Alam, A., & Gupta, A. (2023). Evaluation of estimation in software development using deep learning-modified neural network. *Applied Nanoscience*, *13*(3), 2405–2417. https://doi.org/10.1007/s13204-021-02204-9
29. Sun, B., Zhou, H., Song, G., & Zhang, Q. (2024). Research on the technology architecture of full-stack cloud-native hosted cloud. In *Proceedings of the 2024 11th International Conference on Machine Intelligence Theory and Applications (MiTA)* (pp. 1–7). IEEE. DOI:10.1109/MiTA60795.2024.10751724
30. Taivalsaari, A., Mikkonen, T., Pautasso, C., & Systä, K. (2021). Full stack is not what it used to be. In *Lecture Notes in Computer Science* (pp. 363–371). Springer International Publishing.
31. Thomas, C., Mandrik, O., Saunders, C. L., Thompson, D., Whyte, S., Griffin, S., & Usher-Smith, J. A. (2023). *Supplementary Data from The Costs and Benefits of Risk Stratification for Colorectal Cancer Screening Based On Phenotypic and Genetic Risk: A health economic analysis*. https://doi.org/10.1158/1940-6207.22534258
32. Yang, Y., Xia, X., Lo, D., & Grundy, J. (2022). A survey on deep learning for software engineering. *ACM Computing Surveys*, *54*(10s), 1–73. https://doi.org/10.1145/3505243
33. Zeng, Q., Kavousi, M., Luo, Y., Jin, L., & Chen, Y. (2023). Full-stack vulnerability analysis of the cloud-native platform. *Computers & Security*, *129*(103173), 103173. https://doi.org/10.1016/j.cose.2023.103173