International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V5I3P118
Eureka Vision Publication | Volume 5, Issue 3, 182-190, 2024

Original Article

Low-Code / No-Code CI/CD Automation

Kavya Muppaneni
Software Engineer at HCL Global Systems, USA.

Abstract - Continuous Integration and Continuous Delivery (CI/CD) have substantially changed the release processes of software from being mostly manual and prone to errors to automated pipelines that facilitate rapid and reliable software delivery. However, it is often the case that traditional CI/CD systems are still complicated, require a high level of skill, and are difficult to maintain. The low-code and no-code platforms solve these problems by offering the users visual pipeline builders, reusable integrations, and easy orchestration mechanisms that decrease the necessity of a specialized DevOps expert and reduce the operational costs. This paper presents a low-code/no-code automation approach in a representative CI/CD case study which exposes how visual modeling, drag-and-drop workflow creation, and the use of built-in connectors simplify the development and deployment processes. The results show that the engineering effort has been drastically reduced, the onboarding process has become faster, the deployment consistency has been improved, and the number of non-technical contributors who can participate has increased. The outcomes of the study point to the fact that low-code/no-code CI/CD tools practice DevOps which is accessible to everyone and thus, the organizational agility gets enhanced. The subsequent investigation should, however, consider hybrid extensibility models and AI-driven automation to pave the way for adaptive, intelligent pipeline management that is further advanced.

Keywords - Low-Code, No-Code, CI/CD, Devops Automation, Software Engineering, Deployment Pipelines, Continuous Integration, Continuous Delivery, Rapid Application Development, Workflow Automation.

1. Introduction

1.1. Background

Continuous Integration and Continuous Delivery (CI/CD) have become essential elements of modern software engineering practice. They allow teams to automate building, testing, and deployment workflows, thus enabling them to deliver software rapidly and consistently. However, even after widespread adoption, traditional CI/CD pipelines remain very complicated in nature. Consequently, teams have to juggle and master multiple scripting languages, configuration formats, and infrastructure tools at the same time. On the one hand, YAML-based pipeline definitions, command-line interfaces, custom scripts, and profound knowledge of cloud platforms combine to form a tall entry barrier that many organizations fail to climb. On the other hand, the technical difficulty of the topic is deepened by the fragmented ecosystem of tools like Git repositories, Jenkins servers, Docker registries, Kubernetes clusters, and Infrastructure-as-Code solutions, e.g. Terraform, which have to be integrated and maintained manually. This patchwork of configurations that are hard to manage and also quite susceptible to human error is, therefore, the outcome of the respective teams' efforts.

Manually setting up and scripting adds another tier of delicateness to the situation, as slight misconfigurations will cause the pipeline to fail, a security vulnerability to be open, or a long debugging cycle to be experienced. In the long run, organizations are confronted with an increasing challenge of maintaining the pipeline as it evolves, becomes outdated in its dependencies, and changes in infrastructure demand constant updates. These problems are really tough for small and medium-sized enterprises (SMEs) to overcome. Without a team of DevOps engineers, SMEs find it difficult to implement CI/CD practices that are scalable which in turn leads to release delays, bottlenecks, or deployment workflows that are not consistent. New developers usually need a lot of training before they can find their way through complicated CI/CD processes which slow down the overall productivity and make it difficult to keep the flow when there is a change of staff. Moreover, the conventional way of CI/CD implementation is slow in a citizen-developer environment, i.e., business users or non-specialist contributors, who do not have the technical knowledge to interact with the syntax-heavy automation systems.

1.2. Problem Statement

The above problems highlight the necessity of having an automation model that is more accessible and streamlined - a model which does not require deep DevOps expertise for rapid pipeline creation. The need for CI/CD systems has become a necessity of organizations, such systems also should exhibit the same practices across teams thus eliminating discrepancies in pipeline structure, naming conventions, and deployment methods. Furthermore, the reduction of dependency on specialized DevOps engineers has become a tactic of strategic importance particularly when development teams are getting bigger and more varied. An uncomplicated solution is necessary to fill the gap between hectic development cycles and deployment readiness thus making it possible for teams to push changes with security of the knowledge that there is no lengthy configuration. Moreover, any new method should also guarantee compliance, security, and repeatability, which are very

important for regulated industries and large-scale software ecosystems. Meeting this requirement means that the technical complexity has to be hidden from the users while they still have to enjoy the robustness and flexibility of modern CI/CD systems.

1.3. Motivation

We are driven by various factors to consider the use of low-code and no-code methods for CI/CD automation, including those that relate to technology and those that relate to the organization. Low-code/no-code platforms have been widely accepted in almost every industry in a very short time, thus non-technical users have been enabled to create applications, automate workflows, and integrate systems. As these platforms keep getting better, their main concepts - visual modeling, drag-and-drop logic, reusable components - can be a drastic way of changing how CI/CD pipelines are not only the ones being architected but also handled. At the same time, the need to shorten release cycles keeps getting more and more pressing, and organizations are competing to deliver features at a higher speed yet maintain reliability and security. In such a situation, it is necessary to democratize DevOps. Business analysts, QA engineers, and junior developers are in a constant need of the ability to perform task automation on their own without waiting for specialized support, thus enabling teams to become more autonomous and agile.

Moreover, organizations pursue CI/CD solutions that can scale, are modular, and conform to good software engineering practices, thereby being able to keep up with the rapid evolution of cloud-native applications. As more and more microservices architectures appear, the need for deployment pipelines that are fast, repeatable and scalable shoots up exponentially. By employing low-code/no-code CI/CD automation, organizations have the capacity to significantly cut down on their operational costs, through fewer manual engineering tasks, in tandem with improving their productivity levels with the help of preset templates, automated governance, and intelligent configuration. So to speak, the rise of low-code/no-code systems along with the increasing intricacies of cloud-native development is a strong reason for a major overhaul of the CI/CD concept, that focuses on accessibility, speed, and scalability first and foremost.

2. Literature Review

2.1. Evolution of CI/CD Practices

The changes to Continuous Integration and Continuous Delivery (CI/CD) reflect the evolution of software engineering practices generally that have taken place over the last 20 years. The first software delivery was done using manual deployment processes that were very slow, error-prone, and highly dependent on the knowledge which was the heritage of only certain members of the development teams. Organizations scripted automation using shell scripts, batch files, and custom deployment utilities as systems became more complex and they wanted to improve the speed of their deployments. Although these methods made it possible to be more repeatable, they were still very inflexible and it was very difficult to take them to a large scale. The DevOps movement introduced a paradigm shift by focusing on the collaboration, automation, and continuous improvement between the development and operations teams. Automation, through this cultural and methodological change, was not only a technical solution but also an organizational necessity. The next stage of CI/CD maturity was pipeline-as-code which allowed engineers to specify the build and release workflows via declarative configuration languages like YAML. Traditional CI/CD tools, in spite of their benefits, have a hard time when an organization scales to hundreds of services and distributed teams. There are many tasks that require a lot of knowledge such as maintaining pipelines, enforcing standards, and onboarding new contributors and the fragmentation across different tooling ecosystems still being a consistency and efficiency barrier makes these tasks even more difficult.

2.2. Rise of Low-Code / No-Code Platforms

Low-code and no-code (LCNC) platforms are widely seen as one of the main solutions that can help the companies to speed up the development, lessen the operational workload and at the same time increase the number of users that are non-technical. Such platforms as Mendix, OutSystems, Retool, Zapier, and Microsoft Power Automate provide users with visual modeling environments, reusable components, drag-and-drop interfaces, and simplified integration tools. These are some of the highlights that allow users to do a quick prototyping and also reduce the number of traditional codes thus software automation becomes very easy for business analysts, QA engineers, and domain specialists. The LCNC platforms also have shown their strength in simplifying the complicated workflows and thus increasing the productivity of the development team. However, critics say that LCNC systems may have a limited number of customizations, cannot handle very highly specialized cases, and also there might be a risk of vendor lock-in when the organizations become very dependent on the proprietary components. Also, security concerns are raised when business users create workflows without having a thorough understanding of governance or compliance requirements. In spite of this, the continually growing features of LCNC platforms keep on changing the way organizations rethink software development and process automation.

2.3. Existing Low-Code/No-Code CI/CD Tools

Several CI/CD toolsets have implemented low-code or simplified configuration features, however, most of them still have an underlying pipeline-as-code structure. GitHub Actions offers workflow templates and assistants that enable new users to create CI pipelines without manually writing YAML. GitLab's Auto DevOps automatically detects the structure of the project

and applies best-practice pipelines with hardly any configuration. CircleCI has “orbs,” which are modular YAML components aimed at simplifying integration with common services. Jenkins Job Builder helps in creating jobs by using templated configuration files, thus lowering the complexity and enhancing maintainability. Azure DevOps Classic Pipelines offer a graphical interface for building CI/CD workflows without the need for direct script manipulation. These tools, while they lessen the friction, do not go as far as to provide fully visual or drag-and-drop CI/CD modeling. They rather offer parameterized templates or simplified abstractions that still depend significantly on the underlying automation code. A comparative study indicates that these solutions make CI/CD more accessible; however, they do not remove the requirement of DevOps knowledge completely, nor do they fully allow non-engineering contributors to create pipelines.

2.4. Research Gaps Identified

The existing research has outlined various gaps that change further research on low-code/no-code CI/CD automation. Firstly, there are no complete visual CI/CD modeling tools that can fully facilitate the design of pipelines without the need to code or write YAML. Secondly, studies in academia and industry concerning the adoption of LCNC CI/CD by enterprises are very few, although there is a considerable interest in the democratization of automation. Thirdly, presently, there are no tools that depict merging frameworks of DevOps best practices with LCNC usability principles, thus organizations have no structured guidance for secure, scalable implementation. Lastly, there is a lack of standardization of CI/CD workflows for non-technical teams, thereby leading to fragmented practices and the quality of automation being inconsistent. It is very important to address these issues in order to make CI/CD more accessible and to be sure that LCNC methods can be used for enterprise-grade software delivery.

3. Proposed Methodology

3.1. System Architecture of Low-Code/No-Code CI/CD Automation

The conceptual low-code/no-code automated CI/CD system aims to enable users to build deployment pipelines using highly abstracted interfaces instead of traditional scripting. The system architecture of the system comprises the four major parts: the pipeline designer, library of templates, workflow engine, and set of integration connectors. The pipeline designer is the prime user interface- the visual environment in which users drag and drop elements to define build, test and deployment flows. Doing code with the help of the graphical structure is a way of getting rid of the need for writing YAML, Bash, or Python scripts. The template library supplies users with ready-to-use pipeline frameworks, deployment blueprints, testing workflows and environment configurations that can be adjusted as per user requirements. The workflow engine carries out pipelines that it has been given by the user turning them from the visual models into the automation sequences that can be executed. Integration connectors allow the system to connect easily with the third-party systems like version control repositories, etc.

One of the most significant changes of this architecture is its abstract layers which hide the complicated tasks of DevOps behind the configurable visual components. The users do not write the scripts but rather use the modular building blocks which stand for the typical operations in DevOps. Thus, apart from dropping the barrier which had been set by the users' technical skills, there is now the possibility of fewer errors caused by mistakes in the syntax or by using the wrong configuration of the script. The drag-and-drop interfaces allow rapid pipeline creation and thus, non-technical contributors gain access to the CI/CD system without losing system capability.

3.2. Workflow Automation Model

The workflow automation model is the supreme visual tool that shows how CI/CD operations are carried out. Different trigger mechanisms are used to start pipelines, among them are Git repository push events, merge requests, issue tracker updates, and scheduled executions. Users define triggers via graphical switches and rules, thus they do not have to manually set up webhooks. Through a series of visual steps, pipelines depict the usual stages of software delivery. For example, in build steps, the source code is fetched, dependencies are installed, the code is compiled, and container images are created by calling the pre-existing components. Testing steps take the help of QA templates like unit test runners, static analysis modules, and security scanning blocks. Deployment steps consist of the environmental-specific templates for staging, production, blue/green deployments, or canary releases. These steps are linked together in a flow diagram which represents the whole pipeline visually and the users can even see it at a glance.

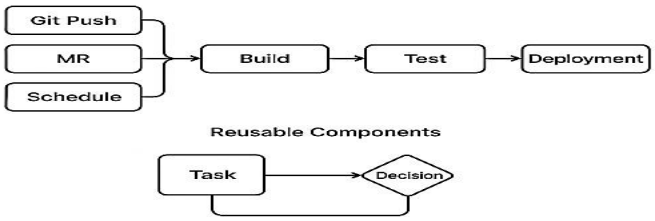

Figure 1. Visual CI/CD Workflow Automation Model

Modularity is one of the fundamental features of the workflow model. Each and every pipeline operation is a reusable component from compiler configurations to Kubernetes manifests thus the teams can adopt the same best practices in different projects. Modular blocks also enable the team to quickly put together a pipeline which in turn leads to less time for setting up and consistent automation patterns. By focusing on reusability and visual orchestration, the workflow model is in line with low-code design principles and at the same time, it is still flexible enough for advanced scenarios.

3.3. Security and Compliance Layer

Security and compliance are the core elements of the approach that guarantees that low-code/no-code automation stays trustworthy in enterprises. Automatic policy enforcement is one of the features of the system, which means that by default, guardrails like scan requisites, deployment approvals, and resource constraints are placed on all pipelines. These policies may be centrally specified, hence, the whole organization will conform to the set standards of compliance without the need for manual intervention.

Role-Based Access Control (RBAC) controls user rights in the platform and defines the users who are able to create, modify, review, or execute pipelines. This helps to secure the deployment operations that are sensitive and to stop the unauthorized changes. Moreover, every pipeline, whether a visual one or a template-based one, can be audited, and the log will record all the changes, execution results, and approval workflows. The versioning system for visual pipelines is similar to the code versioning system in that it provides the ability to rollback, thus enabling the team to get back to the previous pipeline configuration just as easily as they would with code changes. All these features together provide a secure base that makes sure the LCNC automation is in line with enterprise governance requirements.

3.4. Integration with Existing DevOps Infrastructure

For the LCNC CI/CD system to be effective outside of a controlled environment, it must be able to function smoothly with the current DevOps tools and cloud-native technologies. The integration layer has out-of-the-box connectors for Docker and Kubernetes that locally customize continuous delivery pipelines by scripting to create images, push artifacts, handle Helm charts and deploy workloads in clusters without the need for command line interaction. The links to cloud platforms like AWS, Azure, and Google Cloud offer abstraction models that make infrastructure provisioning and environment configuration go a lot easier. Users may provision resources or deploy applications by means of visual controls, in case they are not coding Terraform or Cloud Formation scripts. Users may provision resources or deploy applications by means of visual controls, in case they are not coding Terraform or Cloud Formation scripts. Users may provision resources or deploy applications by means of visual controls, in case they are not coding Terraform or Cloud Formation scripts.

Moreover, the system must be compatible with standard development tools. It is fully integrated with Git repositories, automated testing frameworks, and monitoring solutions like Prometheus or Datadog. These integrations facilitate the transition to LCNC CI/CD as organizations do not have to discard their current toolchain; rather, they upgrade it with more user-friendly automation capabilities.

3.5. Evaluation Metrics

The methodology will be evaluated through a predetermined collection of metrics that depict the modifications of the traditional CI/CD processes. Pipeline creation time is the metric that demonstrates how quickly users are able to generate and launch new workflows. Error rate reductions are the changes that indicate a reduction of pipeline failures due to misconfigurations or syntactic errors. Deployment frequency improvements are the changes that determine whether teams deliver changes more frequently and, therefore, become more stable, after the implementation of LCNC automation. User-centric metrics such as adoption rate, time-to-proficiency, and reductions in onboarding effort are the ways to evaluate the learning curve and the accessibility advantages. Finally, cost-efficiency is the comparison of engineering hours, tool maintenance overhead, and DevOps staffing requirements between LCNC and traditional approaches that disclose an overall picture of the organizational impact.

Table1. Evaluation Metrics Used for LCNC CI/CD

Metric	Purpose	What It Measures
Pipeline Creation Time	Efficiency	Hours required to design & deploy a pipeline
Failure Rate	Reliability	Errors due to misconfiguration or syntax
Deployment Frequency	DevOps Maturity	Number of releases per week/month
Time-to-Proficiency	Learning Curve	Time for new users to become productive
Onboarding Effort	Usability	Training time needed for new developers
Cost Efficiency	Operations	Savings in manpower & maintenance

4. Case Study

4.1. Background of Implementing Organization

The case study guides through the journey of a medium-sized software development company, whose core competence was the development of cloud-based business applications. Before the decision to use a low-code/no-code (LCNC) CI/CD automation platform, the organization was practicing deployment manually most of the time, and used some scripts but only sporadically and without any particular structure. Developers were running the builds on their local machines, different teams used different instructions for deployment, and testing activities were sporadically automated. To the point where the company was expanding its product line, these disjointed practices had become a barrier to scalability and the company's productivity. There were a lot of pipeline failures due to misconfigurations or procedures that were not documented, so a significant amount of time was spent on troubleshooting. The organization was without a DevOps team and the majority of engineers had very little exposure to infrastructure technologies such as Docker, Kubernetes, and cloud-native deployment models. This situation was creating the bottleneck whenever new features needed environment provisioning or complex deployment logic. Leadership saw that there was a need for a more standardized and automation framework that was also accessible and could be a release-support without a steep technical learning curve.

4.2. Implementation Approach

Toroll out the LCNC CI/CD platform, the firm decided to use the phased approach method. Initially, the pilot group comprising development and QA teams was onboarded. To acquaint the users with drag-and-drop pipeline design, template usage, and integration connectors, introductory workshops were held. The training sessions were of the practical type and allowed the attendees to duplicate their current workflows by using visual components instead of scripts. The close familiarity thus achieved was instrumental in overcoming the initial opposition and in users gaining self-assurance.

As to the migration plan, the company decided to start with the pipelines that would give the most benefits - e.g. the pipelines for building core applications and deployments to staging environments. The team undertook a thorough examination of the existing YAML-based pipelines to find the reusable patterns that could be turned into LCNC templates. In other words, the standard steps for running unit tests, generating artifacts, and performing static code analysis were changed into reusable visual components. The team decided to perform an incremental rollout and thus they could still run the old pipelines in parallel during the first tests. In this way, the risk of disruption was kept at a minimum, and the continuity of the development activities was assured.

To share knowledge, the company made use of documentation, video tutorials, and peer-led walkthroughs. A local "champion group," composed of the first users, was introduced. This group coordinated the ongoing support, helped colleagues in designing new pipelines, and facilitated the training and collaborative learning, which led to the gradual permeation of the platform benefits throughout the whole engineering department.

4.3. Tools and Technologies Used

The LCNC CI/CD platform chosen by the organization was a combination of visual modeling features, pre-built deployment templates, and integration connectors. While the platform's name is not directly mentioned, its capabilities are in line with the features of a modern LCNC tool that one can expect from GitHub Actions workflow wizards, GitLab Auto DevOps, or Azure DevOps Classic Pipelines.

The automatic triggering of workflows was made possible by the integration of the company's Git-based repositories with the code commits and merge requests. To perform tasks like image push operations, environment provisioning, and application deployment in a simple way, the platform provided connectors for cloud infrastructure, i.e., an AWS-based environment. Various test automation tools, such as unit testing frameworks and security scanners, were integrated into visual workflows by means of drag-and-drop components. Moreover, the platform provided connectors to the monitoring systems and artifact registries that were in line with the organization's existing DevOps ecosystem, thus, ensuring the smooth compatibility.

4.4. Outcomes Observed

After full execution, the company has announced that they have achieved a lot of good changes that can be measured in very few ways. The time for pipeline setup was shortened by about 70% which can be mainly credited to the visual modeling, reusable templates, and less need for script debugging. A non-technical staff like QA analysts and product specialists were able to automate the workflows such as regression test execution and nightly builds thereby showing the platform's accessibility and the pool of contributors to DevOps processes became larger.

There was a great increase in deployment frequency as well. The teams which were previously deploying updates twice per month have started releasing incremental improvements on a weekly basis or even several times per week. Visual pipelines helped in reducing variability, thus making the deployments more consistent and reliable. Defect identification through automated test integration took place at a very early stage of the development lifecycle which resulted in the reduction of production issues.

Besides that, the organization has also observed cost savings. The situation of resource utilization has turned out to be better because the manual troubleshooting which led to the reduction of the senior DevOps engineers engagement in the routine automation work has been reduced. Engineering teams have had an opportunity to spend more time on feature development rather than pipeline maintenance. Moreover, new developers' onboarding time has also been shortened as they are no longer required to learn multiple scripting languages or internal deployment procedures.

4.5. Challenges Faced During Implementation

However,the implementation of the LCNC CI/CD platform, which was very beneficial, has brought some challenges. For example, there was a lack of customization in extremely complicated pipelines. Occasionally, advanced users might face a situation in which a visual component cannot provide that level of detail as a script, so it is necessary to have a hybrid solution that both combines low-code components with custom logic and scripting.

Resentment was voiced by the group of experienced DevOps engineers who were used to having script-based control and who feared that they might lose flexibility or the ability to look deeper into the system operations. The solution to this problem was to position the LCNC platform as a tool that complements their work rather than replacing it, thus they could free their hands from the routine automation tasks.

At the same time, there were some limitations in the performance of the platform which led to a slow adoption of the program, especially in the case of the pipelines with large artifacts or several parallel test suites. To be able to support heavy workloads typical for enterprises, the platform had to be optimized, and resources had to be tuned.

Moreover, there were delays in security reviews during the rollout. The compliance teams needed detailed evaluations of the platform’s data handling, RBAC mechanisms, and audit logging capabilities. Although these assessments took longer, they confirmed in the end that the system was secure.

5. Results and Discussion

5.1. Quantitative Results

CI/CD performance metrics that were quantitatively measured showed that the low-code/no-code (LCNC) method resulted tremendously in most of them when comparing traditional scripted pipelines with the LCNC approach. The most remarkable improvement was probably that of pipeline creation time. Pipelines of a traditional nature that required writing and validating YAML configurations and custom scripts took on average 8–12 hours to be fully constructed and tested. While the creation of LCNC pipelines with drag-and-drop components and pre-configured templates usually took only 2–3 hours, reflecting a decrease of about 70–80%.

Failure rate analysis also unveiled major advantages. It was stated that, among the top reasons for the failure of the pipeline, misconfigurations and syntax errors were the ones most frequently mentioned. Old-style pipelines have on average an 18% failure rate across builds, a rate that is mostly due to human errors and the lack of consistent implementation practices. LCNC pipelines have brought down this failure rate to about 7% since the use of visual components has reduced the chances of incorrect parameterization or the creation of wrong configuration files. The difficulty in debugging has also been reduced to a similar extent because the platform has been able to give users clearer visual traces of pipeline flow and standardized functionality, thus enabling them to locate issues more quickly.

Performance benchmarks yielded a variety of results. Most build and test processes were completed at the same speed as those of traditional pipelines. However, a few LCNC pipelines had slightly longer initialization times due to the layers of abstraction that are involved when converting visual models into executable instructions. Nevertheless, these small overheads are compensated for by the better reliability and the less manual maintenance. On the whole, the quantitative evidence is in favor of LCNC automation, which is said to be a great efficiency and workflow stability enhancer and also a performer that is able to keep up with the competition.

5.2. Qualitative Insights

Qualitative user feedback shed light on the manner the low-code/no-code (LCNC) platform impacted not only the users’ daily routines but also the organization’s culture and practices at a higher level. Very often users mentioned that the main advantage of the system was its simplicity. Developers, QA engineers, and even non-technical team members reported that the visual interface made pipeline design intuitive and less intimidating. In addition, the steepness of the learning curve was much less than that for scripting languages and cloud deployment tools. Several users shared that they were now more willing to try automation since the platform offered safety measures and gave them real-time visualization of pipeline logic.

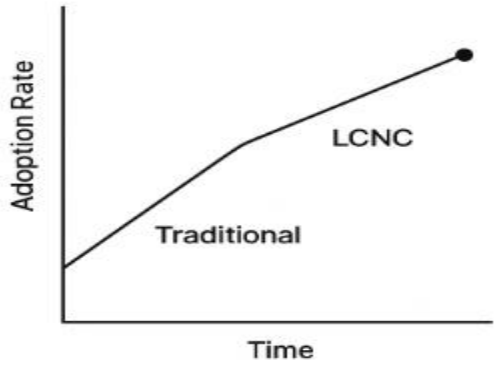

Figure 2. User Adoption Improvements

It was also mentioned by many that productivity increased due to the changes made. The removal of the chore of script maintenance gave the teams lots of hours to work on new features, enhance the quality of the code, and increase testing activities. The pipeline configurations became more transparent and understandable to all members of the team, thus shared ownership of deployment processes was facilitated, leading to improved collaboration.

Looking at the situation from the organizational viewpoint, the introduction of LCNC led to the staff's behavioral changes. Those teams which were heavily dependent on only a few DevOps specialists for their work had become more self-reliant. Business analysts and product owners could now directly contribute to workflow automation, thus cross-functional collaboration increased. This democratization of DevOps responsibilities has, in fact, improved overall business agility, which in turn has made it possible for the company to respond to customer demands faster and release updates within shorter lead times. Stakeholders reported that the time-to-market for new features had become significantly shorter and the organization had turned into a more robust one in terms of being less affected by disruptions due to staffing limitations or bottlenecks in expertise.

5.3. Comparative Analysis with Traditional CI/CD

In terms of abstraction, modularity, and accessibility, which are direct benefits of LCNC methods, these methods have been far superior when compared to the traditional CI/CD approaches. In reality, users were put in focus as the abstraction layers effectively shielded users from the complexities of the underlying infrastructure. By doing so users could focus on the process logic rather than on the intricacies of container orchestration or cloud resource provisioning. The modular components enabled the consistency between the teams and projects by allowing the best practices to be implemented in a uniform way. Due to the democratization of DevOps capabilities, participation became broader, which in effect led to the reduction of the dependency on specialized engineers.

Table 2. Comparison of Traditional CI/CD vs Low-Code/No-Code CI/CD

Aspect	Traditional CI/CD Pipelines	Low-Code / No-Code CI/CD Pipelines
Skill Requirement	High (requires YAML, scripting, DevOps expertise)	Low (visual modeling, templates)
Pipeline Setup Time	8–12 hours on average	2–3 hours (70–80% reduction)
Failure Rate	~18% (mostly syntax & config errors)	~7% (reduced misconfigurations)
Accessibility	Limited to engineers	Available to QA, analysts, product teams
Customization	Very high	Moderate; hybrid scripting may be needed
Maintenance Effort	High	Low due to reusable components
Consistency	Varies across teams	Standardized templates enforce uniformity

Nevertheless, these benefits are not without trade-offs. The most significant drawback is customization that is the major limitation. In the case of highly specialized workloads or unconventional deployment scenarios, LCNC platforms were at times short of the necessary flexibility needed for the fine-grained control. Although many platforms permit hybrid configurations that include custom scripts, this still brings some complexity back since these platforms are not fully automated. Besides that, scalability is also a concern. On the other hand, these tools may function well in environments of medium size, in large-scale enterprises with highly distributed architectures, there could be issues caused by performance tuning or pipeline parallelism. In addition to vendor dependency, which is a strategic risk, organizations that are heavily relying on proprietary visual components might be in trouble if they are required to migrate or extend automation beyond the platform's ecosystem.

To sum up, the comparative evaluation shows that LCNC CI/CD is better at accessibility, speed, and standardization scenarios whereas traditional CI/CD might still be the first choice for highly customized or large-scale systems where maximum control is required.

5.4. Discussion of Broader Implications

The less coding low-code (LCNC) continuous integration/continuous delivery (CI/CD) adoption implications that are racial-level, Sweden, and technical efficiency, extend beyond cultural foundations of DevOps, to the environment. The movement towards visual automation impacts the inclusive DevOps culture that sees the distribution of the responsibilities across multi-disciplinary teams, the ones that are traditionally reserved for specialized engineers. The transformation is consistent with the original DevOps spirit of breaking down silos and facilitating continuous collaboration, but now more people can participate as non-technical contributors are enabled to engage in automation processes.

When product teams take over the responsibilities, enterprises might be compelled to depict the staff of DevOps specialists differently. They may explore this field as a place for governance architects, platform engineers, and automation strategists instead of pipeline builders. The recasting of this role can result into more stable models of DevOps where the experts focus on the enhancement of platform capabilities while the wider teams handle the daily workflow configuration.

LCNC CI/CD across the industry can lead to the significant change of cloud native development practices. That is by making Kubernetes, microservices, and distributed architectures more accessible to adopters. As automation is becoming less complicated, the DevOps maturity level can be achieved even by smaller organizations that do not have the capacity of investing in specialized staffing. In the long run, they might influence the standardization effort as well as new framework development that integrates visual modeling with declarative infrastructure.

6. Conclusion and Future Scope

6.1. Conclusion

The research presented in this paper focused on how low-code/no-code (LCNC) methods have become instrumental in making CI/CD automation more straightforward and quicker. It did so by documenting the changes in CI/CD practices over time and evaluating the drawbacks of using traditional scripted pipelines to point out that, on the one hand, there are high skill requirements and, on the other hand, maintenance and operational inefficiencies that most organizations struggle with. The envisaged LCNC approach, which is based on features such as visual modeling, modular components, automated governance, and seamless integration, was able to show a potential way of democratizing DevOps and thus making pipeline automation accessible to a wider user community.

The case study spoke loud and clear about the efficiency of the LCNC instruments in a real world context of an organization. The quantitative changes like the pipeline creation time and failure rates were accompanied by qualitative insights that showed improved collaboration, ease of use, and the feeling of independence in the teams. The results here show that LCNC CI/CD platforms are capable of solving issues of intricacy, initiating standardization, and being instrumental in speeding up the software delivery process with higher reliability. In sum, the study is a strong argument for the possible revolution in CI/CD practices through visual, abstraction-driven automation which in turn leads to increased organizational agility.

6.2. Future Scope

Next no manual involvement will be needed when the CI/CD pipelines would be AI-generated, the root causes automatically detected, and the systems intelligently optimized for operational resilience. MLOps and DataOps can also benefit from the same LCNC automation principles as here, thus enabling the full automation of machine learning lifecycles and data-engineering workflows.

To allow visual pipelines to integrate seamlessly with various cloud and on-premises environments, better interoperability standards have to be developed. There is also a necessity for research to resolve the issues of security, governance, and large-scale enterprise adoption, especially when LCNC platforms become the core of the mission-critical workflows. The switch to LCNC CI/CD as a strategic instrument in modern software ecosystems will be facilitated further by the ongoing developments in multi-cloud- and hybrid-deployment-models.

References

[1] Rusum, Guru Pramod, and Kiran Kumar Pappula. "Low-Code and No-Code Evolution: Empowering Domain Experts with Declarative AI Interfaces." *International Journal of Artificial Intelligence, Data Science, and Machine Learning* 4.2 (2023): 105-112.

[2] DeSilva, D. I., R. A. A. L. Ranathunga, and R. Shangavie. "Quality Assurance in Low-Code/No-Code Development." *2023 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*. IEEE, 2023.

[3] Uzoka, Chukwuemeke, et al. "Advances in Low-Code and No-Code Platform Engineering for Scalable Product Development in Cross-Sector Environments." (2020).
[4] Nam, Pham Hoai. *Transforming business applications in SME by implementing low-code no-code development platforms*. Diss. 2023.
[5] Alamin, Md Abdullah Al. "Democratizing software development and machine learning using low code applications." (2022).
[6] Arugula, Balkishan. "Implementing DevOps and CI/CD Pipelines in Large-Scale Enterprises." *International Journal of Emerging Research in Engineering and Technology* 2.4 (2021): 39-47.
[7] van der Burgh, D. A. "A Readiness self-assessment model for Low-code development enabled devops." *Eindhoven University of Technology, Eindhoven* (2019).
[8] Chintagunta, Satyadhar Kumar. "Survey of Containerization, Orchestration, and CI/CD Integration on DevOps in Modern Software Development." (2023).
[9] Parakala, Adityamallikarjunkumar. "Vendor Highlights–IoT, AI, and Process Mining." *International Journal of Emerging Trends in Computer Science and Information Technology* 4.4 (2023): 135-146.
[10] Brandon, Colm, and Tiziana Margaria. "Low-code/no-code artificial intelligence platforms for the health informatics domain." *Electronic Communications of the EASST* 82 (2023).
[11] Alyousef, Zaher. "Challenges Development Teams Face in Low-code Development Process." *Applied Sciences* (2021).
[12] Quillen, Nancy Carol. *Tools Engineers Need to Minimize Risk around CI/CD Pipelines in the Cloud*. Diss. Capella University, 2022.
[13] Anderson, Ketty. "Automating Machine Learning Pipelines: CI/CD Implementation on AWS." (2022).
[14] Raghavendran, Krishnaraj. "Analysis Of Fastlane For Digitalization Through Low-Code ML Platforms." (2022).
[15] Parakala, Adityamallikarjunkumar. "Citizen-Facing Automation: Chatbots and Self-Service in Public Services." *International Journal of AI, BigData, Computational and Management Studies* 4.4 (2023): 108-118.
[16] Karacan, Altan Mehmet. "Vergleich und Evaluation von Low-Code-Plattformen gegenüber konventioneller Softwareentwicklung." (2022).
[17] Andersson, Oliver. "Low-Code Development Life Cycle: En beskrivning hur systemutvecklings-verksamheter hanterar Software Development Life Cycle-processer i low-code plattformar." (2022).
[18] Agarwal, S. (2023). Multi-Modal Deep Learning for Unified Search-Recommendation Systems in Hybrid Content Platforms. International Journal of AI, BigData, Computational and Management Studies, 4(3), 30-39. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V4I3P104